11. ÓRA

A DBM függvények használata

Ha nem is férünk hozzá valamilyen SQL adatbáziskezelőhöz (mint a MySQL vagy az Oracle), majdnem biztos, hogy valamilyen DBM-szerű adatbázisrendszer rendelkezésünkre áll. Ha mégsem, a PHP akkor is képes utánozni számunkra egy ilyen rendszer működését. A DBM függvények segítségével lényegében név–érték párokat tárolhatunk és kezelhetünk.

Noha ezek a függvények nem biztosítják számunkra egy SQL adatbázis erejét, rugalmasak és könnyen használhatók. Mivel a formátum igen elterjedt, az e függvényekre épülő kód általában hordozható, noha maguk a DBM adatokat tartalmazó állományok nem azok.

Ebben az órában a következő témákkal foglalkozunk:

- Megtanuljuk a DBM adatbázisok kezelését.
- Adatokkal töltünk fel egy adatbázist.
- Visszanyerjük adatainkat egy adatbázisból.
- Elemeket módosítunk.
- Megtanuljuk, hogyan tároljunk bonyolultabb adatokat egy DBM adatbázisban.

DBM adatbázis megnyitása

A DBM adatbázisokat a `dbmopen()` függvénnyel nyithatjuk meg, amelynek két paramétert kell átadnunk: a DBM fájl elérési útvonalát és a megnyitás módjára vonatkozó kapcsolókat. A függvény egy különleges DBM azonosítóval tér vissza, amelyet aztán a különböző egyéb DBM függvényekkel az adatbázis elérésére és módosítására használhatunk fel. Mivel a `dbmopen()`egy fájlt nyit meg írásra vagy olvasásra, a PHP-nek joga kell, hogy legyen az adatbázist tartalmazó könyvtár elérésére.

A `dbmopen()` függvénynek a 11.1. táblázatban felsorolt kapcsolókkal adhatjuk meg, milyen műveleteket kívánunk végrehajtani az adatbázison.

11.1. táblázat A dbmopen() kapcsolói

Kapcsoló	*Jelentés*
`r`	Az adatbázist csak olvasásra nyitja meg.
`w`	Az adatbázist írásra és olvasásra nyitja meg.
`c`	Létrehozza az adatbázist (ha létezik, akkor írásra/olvasásra nyitja meg).
`n`	Létrehozza az adatbázist (ha már létezik ilyen nevű, törli az előző változatot).

A következő kódrészlet megnyit egy adatbázist, ha pedig nem létezne a megadott néven, újat hoz létre:

```
$dbm = dbmopen( "./adat/termekek", "c" ) or
➥ die( "Nem lehet megnyitni a DBM adatbázist." );
```

Vegyük észre, hogy ha nem sikerülne az adatbázis megnyitása, a program futását a `die()` függvénnyel fejezzük be.

Ha befejeztük a munkát, zárjuk be az adatbázist a `dbmclose()` függvénnyel. Ez azért szükséges, mert a PHP automatikusan zárolja a megnyitott DBM adatbázist, hogy más folyamatok ne férhessenek hozzá a fájlhoz, mialatt a tartalmát olvassuk vagy írjuk. Ha az adatbázist nem zárjuk be, a várakozó folyamatok azután sem érhetik el az adatbázist, amikor már befejeztük a munkát. A `dbmclose()` függvény paramétere egy érvényes DBM azonosító:

```
dbmclose ( $dbm );
```

Adatok felvétele az adatbázisba

Új név–érték párt a `dbminsert()` függvénnyel vehetünk fel az adatbázisba. A függvénynek három paramétere van: egy érvényes DBM azonosító (amelyet a `dbmopen()` adott vissza), a kulcs és a tárolandó érték. A visszatérési érték 0, ha sikeres volt a művelet; 1, ha az elem már szerepel az adatbázisban; és `-1` bármilyen más hiba esetén (például írási kísérlet egy csak olvasásra megnyitott adatbázisba). A `dbminsert()` már létező elemet nem ír felül.

A 11.1. programban létrehozzuk és megnyitjuk a `termekek` nevű adatbázist és feltöltjük adatokkal.

11.1. program Adatok felvétele DBM adatbázisba.

```
 1: <html>
 2: <head>
 3: <title>11.1. program Adatok felvétele DBM adatbázis-
    ba</title>
 4: </head>
 5: <body>
 6: Termékek hozzáadása...
 7:
 8: <?php
 9: $dbm = dbmopen( "./adat/termekek", "c" ) or
     die( "Nem lehet megnyitni a DBM adatbázist." );
10:
11: dbminsert( $dbm, "Ultrahangos csavarhúzó", "23.20" );
12: dbminsert( $dbm, "Tricorder", "55.50" );
13: dbminsert( $dbm, "ORAC AI", "2200.50" );
14: dbminsert( $dbm, "HAL 2000", "4500.50" );
15:
16: dbmclose( $dbm );
17: ?>
18: </body>
19: </html>
```

Az adatbázisba illesztés során az összes érték karakterlánccá alakul, így a termékek árainak megadásakor idézőjeleket kell használnunk. Természetesen az adatbázisból kiolvasás után ezeket az értékeket lebegőpontos számokként is kezelhetjük, amennyiben szükséges. Vegyük észre, hogy nem csak egyszavas kulcsokat használhatunk.

Ha ezek után meghívjuk a dbminsert() függvényt egy olyan kulcsértékkel, amely már létezik az adatbázisban, a függvény az 1 értéket adja vissza és nem módosítja az adatbázist. Bizonyos körülmények között pontosan erre van szükség, de előfordulhat olyan eset is, amikor módosítani szeretnénk egy meglévő adatot, vagy ha nem található ilyen kulcs az adatbázisban, új adatot szeretnénk felvinni.

Adatok módosítása az adatbázisban

A DBM adatbázisban a bejegyzéseket a dbmreplace() függvénnyel módosíthatjuk. A függvény paraméterei: egy érvényes DBM azonosító, a kulcs neve és az új érték. A visszatérési érték a hibakód: 0, ha minden rendben volt és −1, ha valamilyen hiba lépett fel. A 11.2. példában az előző program egy új változata látható, amely a kulcsokat korábbi meglétüktől függetlenül felveszi az adatbázisba.

11.2. program Elemek felvétele vagy módosítása DBM adatbázisban

```
 1: <html>
 2: <head>
 3: <title>11.2. program Elemek felvétele vagy módosítása
           DBM adatbázisban</title>
 4: </head>
 5: <body>
 6: Termékek hozzáadása...
 7: <?php
 8: $dbm = dbmopen( "./adat/termekek", "c" )
 9:             or die( "Nem lehet megnyitni a DMB adatbá-
                   zist." );
10: dbmreplace( $dbm, "Ultrahangos csavarhúzó", "25.20" );
11: dbmreplace( $dbm, "Tricorder", "56.50" );
12: dbmreplace( $dbm, "ORAC AI", "2209.50" );
13: dbmreplace( $dbm, "HAL 2000", "4535.50" );
14: dbmclose( $dbm );
15: ?>
16: </body>
17: </html>
```

A program működésének módosításához mindössze át kell írnunk a dbminsert() függvényhívást dbmreplace()-re.

Adatok kiolvasása DBM adatbázisból

Egyetlen elemet a `dbmfetch()` függvény segítségével olvashatunk ki az adatbázisból. Ebben az esetben két paramétert kell átadnunk: egy létező DBM azonosítót és az elérni kívánt kulcs nevét. A függvény visszatérési értéke egy karakterlánc, a kulcshoz tartozó érték lesz. A `"Tricorder"` elem árát például a következő függvényhívással kérdezhetjük le:

```
$ar = dbmfetch( $dbm, "Tricorder" );
```

Ha `"Tricorder"` elem nem található az adatbázisban, a `dbmfetch()` egy üres karakterlánccal tér vissza.

Nem mindig ismerjük azonban az adatbázisban található kulcsokat. Mit tennénk például akkor, ha ki kellene írni a böngészőablakba az összes terméket és a hozzájuk tartozó árakat, anélkül, hogy „beleégetnénk” a programba a termékek nevét? A PHP biztosít egy módszert, amellyel az adatbázisban szereplő összes elemen végiglépkedhetünk.

Az adatbázis első kulcsát a `dbmfirstkey()` függvénnyel kérdezhetjük le. A függvény paramétere egy érvényes DBM azonosító, visszatérési értéke pedig a legelső kulcs. Természetesen ez nem feltétlenül egyezik meg az elsőként beillesztett adattal, ugyanis a DBM adatbáziskezelők gyakran saját rendezési eljárást használnak. Miután megkaptuk a legelső kulcsot, az összes rákövetkező elemet a `dbmnextkey()` függvény ismételt hívásával kérdezhetjük le. A függvény paraméterként szintén egy érvényes DBM azonosítót vár, visszatérési értéke pedig a következő kulcs a sorban. Ha ezeket a függvényeket együtt használjuk a `dbmfetch()`-csel, az adatbázis teljes tartalmát kiolvashatjuk.

A 11.3. példaprogram a `termekek` adatbázis teljes tartalmát kiírja a böngészőbe.

11.3. program DBM adatbázis összes bejegyzésének kiolvasása

```
1: <html>
2: <head>
3: <title>11.3. program DBM adatbázis összes bejegyzésének
4: kiolvasása</title>
5: </head>
6: <body>
7: A Hihetetlen Kütyük Boltja
8:    a következő izgalmas termékeket kínálja
9:    Önnek:
```

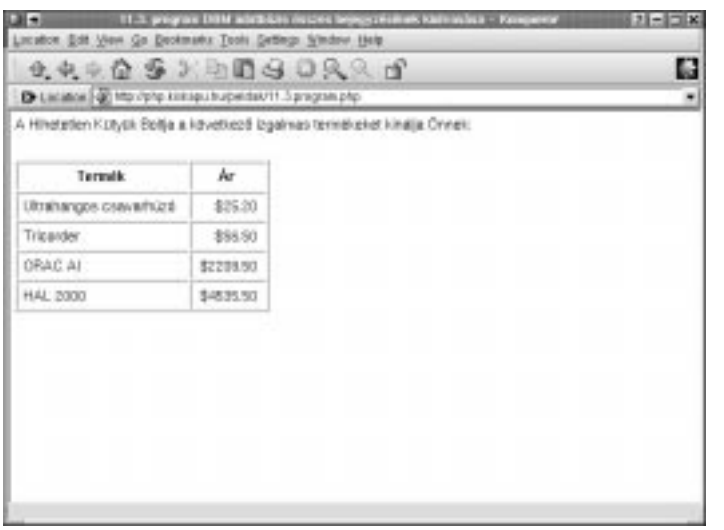
11.3. program (folytatás)

```
10: <p>
11: <table border="1" cellpadding="5">
12: <tr>
13: <td align="center"> <b>Termék</b> </td>
14: <td align="center"> <b>Ár</b> </td>
15: </tr>
16: <?php
17: $dbm = dbmopen( "./adat/termekek", "c" )
18:        or die( "Nem lehet megnyitni a DBM adatbázist."
                   );
19: $kulcs = dbmfirstkey( $dbm );
20: while ( $kulcs != "" )
21:     {
22:     $ertek = dbmfetch( $dbm, $kulcs );
23:     print "<tr><td align = \"left\"> $kulcs </td>";
24:     print "<td align = \"right\"> \$$ertek
              </td></tr>";
25:     $kulcs = dbmnextkey( $dbm, $kulcs );
26:     }
27: dbmclose( $dbm );
28: ?>
29: </table>
30: </body>
31: </html>
```

A 11.1. ábrán a 11.3. program eredménye látható.

11.1. ábra

A DBM adatbázis összes bejegyzésének lekérdezése.



Elemek meglétének lekérdezése

Mielőtt kiolvasnánk vagy módosítanánk egy elemet, hasznos lehet tudni, hogy létezik-e egyáltalán ilyen kulcsú elem az adatbázisban vagy sem. Erre a célra a `dbmexists()` függvény szolgál, amely paraméterként egy érvényes DBM azonosítót vár, illetve az ellenőrizendő elem nevét. A visszatérési érték `true`, ha az elem létezik.

```
if ( dbmexists( $dbm, "Tricorder" ) )
   print dbmfetch( $dbm, "Tricorder" );
```

Elem törlése az adatbázisból

Az adatbázisból elemeket a `dbmdelete()` függvénnyel törölhetünk. A függvény bemenő paramétere egy érvényes DBM azonosító és a törlendő elem neve. Sikeres törlés esetén a visszatérési érték `true`, egyéb esetben (például ha az elem nem létezik) `false`.

```
dbmdelete( $dbm, "Tricorder" );
```

Összetett adatszerkezetek tárolása DBM adatbázisban

A DBM adatbázisban minden adat karaktersorozat formájában tárolódik, ezért az egész és lebegőpontos számokon, illetve karakterláncokon kívül minden egyéb adattípus elvész. Próbáljunk meg például egy tömböt tárolni:

```
$tomb = array( 1, 2, 3, 4 );
$dbm = dbmopen( "./adat/proba", "c" ) or
 ➥ die("Nem lehet megnyitni a DBM adatbázist.");
dbminsert( $dbm, "tombproba", $tomb );
print gettype( dbmfetch( $dbm, "tombproba" ) );
// A kimenet: "string"
```

Itt létrehozunk egy tömböt és a `$tomb` változóba helyezzük. Ezután megnyitjuk az adatbázist és beszúrjuk a tömböt `tombproba` néven, majd megvizsgáljuk a `dbmfetch()` függvény visszatérési típusát, amikor megpróbáljuk visszaolvasni a `tombproba` elemet – láthatjuk, hogy karakterláncot kaptunk vissza. Ha kiírattuk volna a `tombproba` elem értékét, az `"Array"` karakterláncot kaptuk volna. Úgy látszik, ezzel el is úszott minden reményünk arra, hogy tömböket vagy más összetett adatszerkezetet tároljunk a DBM adatbázisban.

Szerencsére a PHP rendelkezik egy olyan lehetőséggel, amely segítségével a bonyolultabb szerkezeteket is egyszerű karaktersorozattá alakíthatjuk. Az így „kódolt” szerkezetet már tárolhatjuk későbbi használatra, DBM adatbázisban vagy akár fájlban is.

Az átalakítást a `serialize()` függvénnyel végezhetjük el, amelynek bemenő paramétere egy tetszőleges típusú változó, a visszaadott érték pedig egy karakterlánc:

```
$tomb = array( 1, 2, 3, 4 );
print serialize( $tomb );
// A kimenet: "a:4:{i:0;i:1;i:1;i:2;i:2;i:3;i:3;i:4;}"
```

A DBM adatbázisban ezt a karakterláncot tárolhatjuk. Ha vissza akarjuk állítani eredeti formájára, az `unserialize()` függvényt kell használnunk.

Ezzel a módszerrel lehetőségünk nyílik összetett adatszerkezetek tárolására a DBM adatbázisok által kínált egyszerű eszközökkel is. A 11.4. listában egy asszociatív tömb tartalmazza a termékekről rendelkezésre álló információkat – ezt alakítjuk át karakterlánccá és helyezzük egy DBM adatbázisba.

11.4. program Összetett adatszerkezetek tárolása DBM adatbázisban

```
 1: <html>
 2: <head>
 3: <title>11.4. program Összetett adatszerkezetek tárolása
          DBM adatbázisban</title>
 4: </head>
 5: <body>
 6: Összetett adatok tárolása
 7: <?php
 8: $termekek = array(
 9:       "Ultrahangos csavarhúzó" => array( "ar"=>"22.50",
10:                                   "szallitas"=>"12.50",
11:                                   "szin"=>"zöld" ),
12:       "Tricorder"                 => array( "ar"=>"55.50",
13:                                   "szallitas"=>"7.50",
14:                                   "szin"=>"vörös" ),
15:       "ORAC AI"                   => array( "ar"=>"2200.50",
16:                                   "szallitas"=>"34.50",
17:                                   "szin"=>"kék" ),
18:       "HAL 2000"                  => array( "ar"=>"4500.50",
19:                                   "szallitas"=>"18.50",
20:                                   "szin"=>"rózsaszín" )
21:       );
```

11.4. program (folytatás)

```
22: $dbm = dbmopen( "./adat/ujtermekek", "c" )
23:         or die("Nem lehet megnyitni a DBM adatbázist.");
24: foreach( $termekek as $kulcs => $ertek )
25:       dbmreplace( $dbm, $kulcs, serialize( $ertek ) );
26: dbmclose( $dbm );
27: ?>
28: </body>
29: </html>
```

A listában egy többdimenziós tömböt építünk fel, amely kulcsként tartalmazza a termék nevét és három tömbelemben tárolja a színt, az árat és a szállítás költségét. Ezután egy ciklus segítségével feldolgozzuk az összes elemet: a termék nevét és a karakterlánccá alakított tömböt átadjuk a `dbmreplace()` függvénynek. Ezután lezárjuk az adatbázist.

A 11.5. program az adatok visszatöltésére ad megoldást.

11.5. program Összetett adatszerkezetek visszaolvasása DBM adatbázisból

```
 1: <html>
 2: <head>
 3: <title>11.5. program Összetett adatszerkezetek
           visszaolvasása
 4:        DBM adatbázisból</title>
 5: </head>
 6: <body>
 7: A Hihetetlen Kütyük Boltja
 8:    a következő izgalmas termékeket kínálja
 9:    Önnek:
10: <p>
11: <table border="1" cellpadding ="5">
12: <tr>
13: <td align="center"> <b>Termék</b> </td>
14: <td align="center"> <b>Szín</b> </td>
15: <td align="center"> <b>Szállítás</b> </td>
16: <td align="center"> <b>Ár</b> </td>
17: </tr>
18: <?php
```

11.5. program (folytatás)

```
$dbm = dbmopen( "./adat/ujtermekek", "c" )
           or die("Nem lehet megnyitni
                    a DBM adatbázist.");
$kulcs = dbmfirstkey( $dbm );
while ( $kulcs != "" )
     {
     $termektomb = unserialize( dbmfetch( $dbm,
      $kulcs ) );
     print "<tr><td align=\"left\"> $kulcs </td>";
     print '<td align="left" $termektomb["szin"]
            "</td>";
     print '<td align="right"$
            $termektomb["szallitas"] "</td>";
     print '<td align="right"$ $termektomb["ar"]
            "</td></tr>\n";
     $kulcs = dbmnextkey( $dbm, $kulcs );
     }
dbmclose( $dbm );
?>
</table>
</body>
</html>
```

Ez a megoldás hasonló a 11.3. példában látottakhoz, de ebben az esetben több mezőt nyomtatunk ki. Megnyitjuk az adatbázist és a `dbmfirstkey()`, illetve a `dbmnextkey()` függvénnyel beolvassuk az összes adatot az adatbázisból. A ciklusban az `unserialize()` függvénnyel létrehozzuk a termékeket tartalmazó tömböt. Így már egyszerű feladat kiíratni az összes elemet a böngészőablakba. A 11.2. ábrán a 11.5. program kimenete látható.

11.2. ábra

Összetett adatszerkezetek visszaolvasása DBM adatbázisból

Egy példa

Már eleget tudunk ahhoz, hogy elkészítsünk egy működőképes programot az ebben az órában tanult módszerekkel. A feladat a következő: készítsünk egy karbantartó oldalt, ahol a webhely szerkesztője megváltoztathatja a 11.2. példaprogram által létrehozott adatbázis termékeinek árát. Tegyük lehetővé a rendszergazda számára, hogy új elemekkel bővítse az adatbázist, illetve hogy a régi elemeket törölje. Az oldalt nem tesszük nyilvános kiszolgálón elérhetővé, ezért a biztonsági kérdésekkel most nem foglalkozunk.

Először is fel kell építenünk az űrlapot, amely az adatbázis elemeit tartalmazza. A termékek árának módosításához szükségünk lesz egy szövegmezőre és minden elemhez tartozni fog egy jelölőnégyzet is, melynek segítségével az adott elemet törlésre jelölhetjük ki. A lapon el kell helyeznünk két további szövegmezőt is, az új elemek felvételéhez. A 11.6. példában az oldalt elkészítő program olvasható.

11.6. program HTML űrlap készítése DBM adatbázis alapján

```
 1: <?
 2: $dbm = dbmopen( "./adat/termekek", "c" )
 3:         or die("Nem lehet megnyitni
                   a DBM adatbázist.");
 4: ?>
 5: <html>
 6: <head>
 7: <title>11.6. program HTML űrlap készítése
 8: DBM adatbázis alapján </title>
 9: </head>
10: <body>
11: <form method="POST">
12: <table border="1">
13: <tr>
14: <td>Törlés</td>
15: <td>Termék</td>
16: <td>Ár</td>
17: </tr>
18: <?php
19: $kulcs = dbmfirstkey( $dbm );
20: while ( $kulcs != "" )
21:     {
22:     $ar = dbmfetch( $dbm, $kulcs );
23:     print "<tr><td> <input type=\"checkbox"\
              name=\"torles[]\" ";
```

11.6. program (folytatás)

```
24:      print "value=\"$kulcs\"> </td>";
25:      print "<td> $kulcs </td>";
26:      print "<td> <input type=\"text\"
               name=\"arak[$kulcs]\" ";
27:      print "value=\"$ar\"> </td></tr>";
28:      $kulcs = dbmnextkey( $dbm, $kulcs );
29:      }
30: dbmclose( $dbm );
31: ?>
32: <tr>
33: <td> </td>
34: <td><input type="text" name="uj_nev"></td>
35: <td><input type="text" name="uj_ar"></td>
36: </tr>
37: <tr>
38: <td colspan="3" align="right">
39: <input type="submit" value="Változtat">
40: </td>
41: </tr>
42: </table>
43: </form>
44: </body>
45: </html>
```

Szokás szerint az első lépés az adatbázis megnyitása. Ezután megkezdünk egy űrlapot és mivel nem adunk meg céloldalt, feldolgozó programként magát az oldalt jelöljük ki.

Elkészítjük a táblázat fejlécét, majd végigolvassuk az adatbázist a `dbmfirstkey()` és `dbmnextkey()` függvények segítségével, az értékeket a `dbmfetch()` függvénnyel olvasva ki.

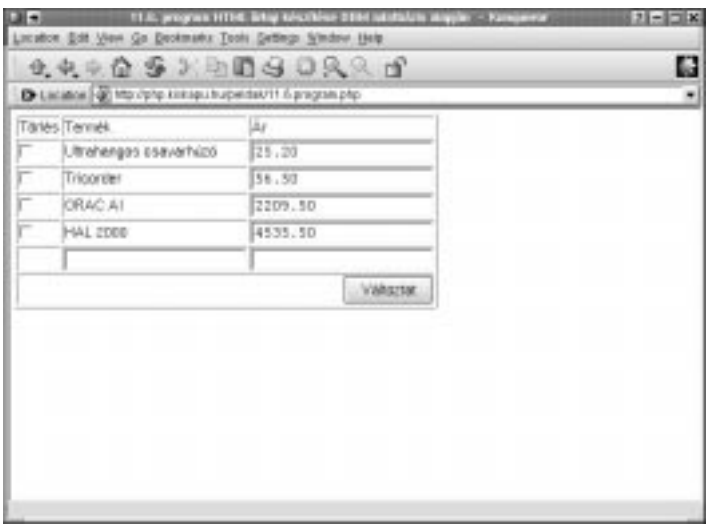
A táblázat első mezője minden sorban egy jelölőnégyzetet tartalmaz. Vegyük észre, hogy mindegyik jelölőnégyzet neve `"torles[]"`. Ennek hatására a PHP létrehozza a `$torles` tömböt, amely az összes bejelölt elemet tartalmazza. A jelölőnégyzetekhez tartozó értékek, így a `$torles` tömb elemei is azok az azonosítók lesznek, amelyekkel a DBM adatbázis az adott terméket nyilvántartja (ezt az értéket a `$kulcs` változóban tároljuk). Így, miután a rendszergazda kitöltötte és elküldte az űrlapot, a program `$torles` tömbje azon adatbázis-elemek kulcsait fogja tárolni, amelyeket törölnünk kell.

Ezután kiírjuk a böngészőablakba az elem nevét és létrehozunk egy szöveges mezőt a termék árának. A mezőt ugyanolyan módon nevezzük el, mint az előzőt, ezúttal azonban a szögletes zárójelek közé beírjuk az azonosítót, amely alapján a DBM adatbázis az elemet tárolja. A PHP ennek hatására létrehozza az `$arak` tömböt, amelyben a kulcsok a termékek azonosítói.

Lezárjuk az adatbázist és visszatérünk HTML módba az új bejegyzés létrehozására szolgáló `uj_nev` és `uj_ar` mezők megjelenítéséhez. A 11.3. ábrán a 11.6. program kimenetét láthatjuk.

11.3. ábra

HTML űrlap készítése DBM adatbázis alapján



Miután elkészítettük az űrlapot, meg kell írnunk a kódot, amely a felhasználó által megadott adatokat kezeli. A feladat nem olyan nehéz, mint amilyennek látszik. Először töröljük a kijelölt elemeket az adatbázisból, majd módosítjuk az árakat, végül felvesszük az új elemet az adatbázisba.

Miután a karbantartó kitöltötte és elküldte az űrlapot, a törlendő elemek listája a `$torles` tömbben áll rendelkezésünkre. Mindössze annyi a dolgunk, hogy végigolvassuk a tömböt, és az összes elemét töröljük az adatbázisból.

```
if ( isset ( $torles ) )
     {
     foreach ( $torles as $kulcs => $ertek )
          {
          unset( $arak[$ertek]);
          dbmdelete( $dbm, $ertek );
          }
     }
```

Először is megvizsgáljuk, hogy létezik-e a `$torles` változó. Ha a felhasználó csak most érkezett az oldalra vagy nem jelölt ki törlése egyetlen terméket sem, a változó nem létezik. Ellenkező esetben egy ciklus segítségével végigolvassuk és minden elemére meghívjuk a `dbmdelete()` függvényt, amely eltávolítja a DBM adatbázisból a paraméterként megadott elemet. Hasonlóan járunk el az `$arak` tömb esetében is, itt azonban a PHP `unset()` függvényét kell használnunk a tömbelem törlésére. Az `$arak` tömb a felhasználótól érkezett, feltehetően részben módosított árakat tartalmazza. Ha nem törölnénk az `$arak` tömbből a megfelelő elemet, a következő kód újra beillesztené.

Az adatbázis elemeinek frissítése során két választási lehetőségünk van. Az első változatot akkor alkalmazzuk, ha az adatbázis karbantartását nem egyetlen szerkesztő fogja végezni, tehát feltételezhető, hogy a programot egyidőben több felhasználó is futtatni fogja – eszerint csak azokat az elemeket változtatjuk meg, amelyeket a felhasználó kijelölt. A másik változat, amely az összes elemet megváltoztatja, akkor alkalmazható, ha a program csak egyetlen példányban futhat:

```
if ( isset ( $arak ) )
     {
     foreach ( $arak as $kulcs => $ertek )
          dbmreplace( $dbm, $kulcs, $ertek );
     }
```

Először is ellenőriznünk kell az `$arak` tömb meglétét. Ebben a tömbben az adatbázis egy teljesen új változata lehet. Egy ciklus segítségével tehát végigolvassuk az összes elemet és egyesével frissítjük az adatbázisban.

Végül ellenőriznünk kell, hogy a felhasználó kezdeményezte-e új elem felvételét az adatbázisba:

```
if ( ! empty( $uj_nev ) && ! empty( $uj_ar ) )
     dbminsert( $dbm, "$uj_nev", "$uj_ar" );
```

Ahelyett, hogy az `$uj_nev` és `$uj_ar` meglétét ellenőriznénk, azt kell ellenőriznünk, hogy értékük nem „üres”-e. Ez apró, de lényeges különbség. Amikor a felhasználó elküldi az űrlapot, a változók mindenképpen létrejönnek, de ha a szövegmezők nem kaptak értéket, a megfelelő változók üres karakterláncot tartalmaznak. Mivel nem akarunk üres elemeket tárolni az adatbázisban, fontos ellenőrizni, hogy nem üresek-e a változók. Azért használjuk a `dbminsert()` függvényt a `dbmreplace()` helyett, hogy elkerüljük a már bevitt termékek véletlen felülírását.

A teljes kódot a 11.7. példa tartalmazza.

11.7. program A teljes adatbázis-karbantartó program

```
 1: <?php
 2: $dbm = dbmopen( "./adat/termekek", "c" )
 3:         or die("Nem lehet megnyitni
                   a DBM adatbázist.”);
 4:
 5: if ( isset ( $torles ) )
 6:      {
 7:      foreach ( $torles as $kulcs => $ertek )
 8:           {
 9:           unset( $arak[$ertek]);
10:           dbmdelete( $dbm, $ertek );
11:           }
12:      }
13:
14: if ( isset ( $arak ) )
15:      {
16:      foreach ( $arak as $kulcs => $ertek )
17:           dbmreplace( $dbm, $kulcs, $ertek );
18:      }
19:
20: if ( ! empty( $uj_nev ) && ! empty( $uj_ar ) )
21:      dbminsert( $dbm, "$uj_nev", "$uj_ar" );
22: ?>
23:
24: <html>
25: <head>
26: <title>11.7. program A teljes adatbázis-karbantartó
           program </title>
27: </head>
28: <body>
29:
30: <form method="POST">
31:
32: <table border="1">
33: <tr>
34: <td>Törlés</td>
35: <td>Termék</td>
36: <td>Ár</td>
37: </tr>
38:
39: <?php
```

11.7. program (folytatás)

```
40: $kulcs = dbmfirstkey( $dbm );
41: while ( $kulcs != "" )
42:      {
43:      $ar = dbmfetch( $dbm, $kulcs );
44:      print "<tr><td> <input type=\"checkbox"\
                name=\"torles[]\" ";
45:      print "value=\"$kulcs\"> </td>";
46:      print "<td> $kulcs </td>";
47:      print "<td> <input type=\"text\"
                name=\"arak[$kulcs]\" ";
48:      print "value=\"$ar\"> </td></tr>";
49:      $kulcs = dbmnextkey( $dbm, $kulcs );
50:      }
51:
52: dbmclose( $dbm );
53: ?>
54:
55: <tr>
56: <td> </td>
57: <td><input type="text" name="uj_nev"></td>
58: <td><input type="text" name="uj_ar"></td>
59: </tr>
60:
61: <tr>
62: <td colspan="3" align="right">
63: <input type="submit" value="Változtat">
64: </td>
65: </tr>
66:
67: </table>
68: </form>
69:
70: </body>
71: </html>
```

Összefoglalás

Ebben az órában megtanultuk, hogyan használjuk a PHP hatékony DBM függvényeit adatok tárolására és visszaolvasására. Megtanultuk a `dbmopen()` használatát egy új DBM azonosító létrehozására. Ezt az azonosítót használtuk az összes többi DBM függvénynél is. Új adatokat adtunk az adatbázishoz a `dbminsert()` függvénnyel, módosítottunk meglevőket a `dbmreplace()`-szel és töröltünk a `dbmdelete()` használatával. Megtanultuk, hogyan használhatjuk a `dbmfetch()` függvényt az adatok visszaolvasására. A `serialize()` és `unserialize()` függvényekkel összetett adatszerkezeteket tároltunk DBM adatbázisban, végül egy gyakorlati példán keresztül megnéztük, hogyan is használhatók fel e módszerek a valós problémák megoldására.

Kérdések és válaszok

Mikor használjak DBM adatbázist SQL adatbázis helyett?
A DBM jó választás, ha kis mennyiségű és viszonylag egyszerű szerkezetű adatot szeretnénk tárolni (név–érték párokat). A DBM adatbázist használó programok rendelkeznek a hordozhatóság nagyon fontos előnyével, de ha nagyobb mennyiségű adatot kell tárolnunk, válasszunk inkább egy SQL adatbáziskezelőt, például a MySQL-t.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Melyik függvényt használhatjuk egy DBM adatbázis megnyitásához?
2. Melyik függvénnyel szúrhatunk be új elemet egy DBM adatbázisba?
3. Melyik függvénnyel módosíthatunk egy elemet?
4. Hogyan érünk el egy elemet az adatbázisban, ha ismerjük a nevét?
5. Hogyan olvasnánk ki egy DBM adatbázisból a legelső elem nevét (nem az értékét)?
6. Hogyan érjük el a további neveket?
7. Hogyan törlünk egy elemet a DBM adatbázisból, ha ismerjük a nevét?

Feladatok

1 Hozzunk létre egy DBM adatbázist a felhasználók azonosítóinak és jelszavainak tárolására. Készítsünk egy programot, amellyel a felhasználók létrehozhatják a rájuk vonatkozó bejegyzést. Ne feledjük, hogy két azonos nevű elem nem kerülhet az adatbázisba.

2 Készítsünk egy bejelentkező programot, amely ellenőrzi a felhasználó azonosítóját és jelszavát. Ha a felhasználói bemenet egyezik valamelyik bejegyzéssel az adatbázisban, akkor üdvözöljük a felhasználót valamilyen különleges üzenettel. Egyébként jelenítsük meg újra a bejelentkező űrlapot.