A Perl és az adatbázisok (2. rész)

Szótár alapú adattárolás állomány-zárolással – az adatbáziskezelők fejlődésének egy elfelejtett lépcsőfoka.

A múlt hónapban egy közönséges szövegfájlban tároltunk egy egyszerű adatbázist. Bár az adattárolás ilyen megvalósítása ma sem számít túlhaladottnak, számos, a komoly rendszerek esetében egyáltalán nem elhanyagolható hiányossága felett átsiklottunk. Mentségünkre csak az szolgálhat, hogy a rendelkezésre álló eszközökkel nehezen hidalhattuk volna át a problémákat.
Ha adatainkat szövegként tároljuk, nincs külön adatbáziskezelő, amely azok épségéért, közvetlen módosításáért, illetve kezeléséért felelne. Mivel nem egy jól elhatárolt réteg végzi ezeket a feladatokat, hanem magának a programozónak kell minderről gondoskodnia, az ilyen rendszer teljesen rugalmatlan és nagyon nehezen bővíthető. A most bemutatásra kerülő módszernél adatbáziskezelő helyett, továbbra is egy függvénykönyvtárra fogunk támaszkodni, amivel azonban már elkerüljük az adatállomány közvetlen kezelését.
Ebben a hónapban a ***Berkeley*** adatbáziskezelőt fogjuk kipróbálni. Ez hash-táblával, vagy kiegyensúlyozott fával dolgozó, szöveges kulcs-érték párokon alapuló adattároló alrendszer. A Berkeley saját állomány-formátummal, a ***dbm***-mel dolgozik. Ezt a formátumot már nem módosíthatjuk közönséges szövegszerkesztővel, sőt a tartalmát sem jeleníthetjük meg ilyen módon.
A szöveges kulcs-érték párokból felépített szótár ötlete meglehetősen régi, és számos megvalósítása létezik, amelyek között szabad felhasználású és zárt forrású, egyaránt akad. Ezek rendszerint szerkezetileg is különböznek. Közülük a *Perl* az `ndbm`, `db`, `gdbm`, `sdbm` és `odbm` szabványokat támogatja. A modulként hozzáférhető megvalósítások abban valamennyien megegyeznek, hogy egy közönséges asszociatív tömbön végzett memória-műveleteket fordítanak le a háttérben alacsony szintű állománykezelési függvényekre. Ez nagyon kényelmes megoldás, hiszen egyetlen tömböt kell csupán kezelni, minden egyéb megy magától. Nem kell a karakterláncok hosszúságával törődni, és nincs határolójel, amely szerepelhetne bárhol. A tároló algoritmus hatékony, így egy rekord módosítása gyorsabb, mint a szöveges alapú adatbázisnál, nagy adatmennyiség esetén pedig a hash-tábla miatt a lekérdezés is sebesebb. Az egyetlen gyenge pont az új rekord felvétele lehet.
Perlben van egy függvény, amely egy változónevet köt össze egy osztállyal, melynek eredményeként a változón végzett bármilyen művelet a kérdéses osztály tagfüggvényeivel valósul meg. Így az adatbázison végzett munka három egyszerű lépésből áll. Hozzá kell kötni egy tömböt az adatállományt felügyelő osztályhoz, a tömbön el kell végezni a szükséges műveleteket, majd meg kell szüntetni a kötést, hogy minden adat a lemezre íródjon. Lássunk egy példát.

```
#!/usr/bin/perl -w

use strict;
use DB_File;

my %adatbazis;

# 1. lépés: adatfeltöltés
tie %adatbazis, 'DB_File', "elso.db", O_CREAT |
    ➥O_WRONLY or
    die "Nem tudom megnyitni az adatbázist.\n";

$adatbazis {'doggie'} = "kutyus";

untie %adatbazis;

# 2. lépés: lekérdezés
tie %adatbazis, 'DB_File', "elso.db", O_RDONLY
    ➥or
    die "Nem tudom megnyitni az adatbázist.\n";

print $adatbazis {'doggie'} . "\n";

untie %adatbazis;
```

Az első két sor – ahogy azt a sorozat korábbi tagjaiban is hangsúlyoztam – elengedhetetlen az átlátható és hibátlan ***Perl*** program írásához. A parancsértelmezőt a -w kapcsolóval hívjuk meg a figyelmeztető üzenetek megjelenítéséhez, majd használatba vesszük a `strict` modult. Az első újdonság a `DB_File` hívása. Ez egy olyan modul, amely tartalmaz egy osztályt a ***Berkeley*** típusú adatbázisok kezeléséhez.
A negyedik sorban a szigorú módnak megfelelően meghatározzuk a használni kívánt asszociatív tömb érvényességi

```
#!/usr/bin/perl -w

use strict;
use DB_File;
use Fcntl ':flock';

my %adatbazis;

# 1. lépés: adatfeltöltés
my $db = tie %adatbazis, 'DB_File',
"masodik.db", O_CREAT | O_RDWR or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "+<&=" . $db -> fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_EX or
    die "Nem tudom megszerezni az egyedi
        ➥zárat.\n";

$adatbazis {'doggie'} = "kutya";

undef $db;
untie %adatbazis;
close ADATALLOMANY;

# 2. lépés: lekérdezés
tie %adatbazis, 'DB_File', "masodik.db",
O_RDONLY or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "<&=" . (tied %adatbazis) ->
    ➥fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_SH or
    die "Nem tudom megszerezni a megosztott
        ➥zárat.\n";

print $adatbazis {'doggie'} . "\n";

untie %adatbazis;
close ADATALLOMANY;
```

körét – jelen esetben olyan, mintha globális változó volna. A program következő része két szakaszra bontható. Az elsőben megnyitja az ***elso.db*** adatállományt, és létrehoz benne egy értéket. A módszer „szótár jellegét" hangsúlyozandó egy angol szót használunk kulcsként, amelynek magyar megfelelője az érték. Az állomány lezárása után a második szakaszban ismét megnyitjuk azt, de már csak olvasásra, majd lekérdezzük az előbb felvett adatot, eleve feltételezve, hogy az létezik. Végül lezárjuk az adatbázist.

Az adatbázis megnyitása mindkét lépésben a `tie` függvénnyel történt, amely az első paraméterként átadott változót köti össze a másodikban meghatározott osztállyal. Mivel közben történik egy példányosítás, a függvény további paraméterei a konstruktornak adódnak át. Jelen esetben csak a megnyitási módot meghatározó állandókat adtuk meg (a továbbiakról a `DB_File` súgójában olvashatunk). A létrejött objektumot egyébként a `tie` vissza is adja, ám ezt itt nem használtuk fel. Végül mindkét esetben az `untie` segítségével szakítottuk meg a kapcsolatot a változó és az adatbázis között.

A módszer egyszerű, de ha két folyamat egyszerre fordul ugyanahhoz az állományhoz, versenyhelyzet alakulhat ki. Ilyenkor kiszámíthatatlan, hogy melyik utasításai jutnak érvényre. Ennek megakadályozására két megoldás létezik, az egyik az állományzárolás, a másik az `O_EXLOCK` jelző használata. Ez utóbbi sajnos nem minden rendszeren támogatott. Ilyenkor segít a UNIX `flock()` rendszerfüggvénye. Ez sajnos egy folyam-azonosítót vár paraméterként, amit nem kapunk meg a `tie` meghívásakor. Kapunk viszont egy objektumot, melynek egy tagfüggvénye visszaadja a megnyitott adatállomány leíróját. Ezt felhasználva az `open()` függvénnyel már készíthetünk egy folyam-azonosítót. Lássuk, hogyan néz ki az előző példa zárolással (*2. lista*).

Az `Fcntl` modul használata a `flock` függvény `LOCK_EX` és `LOCK_SH` állandói miatt szükséges. Az `open` második paramétere egy három részből álló karakterlánc. A „+<", vagy a „<" jelentése megnyitás olvasásra és írásra, illetve a csak olvasásra. Az „&=" azt jelzi, hogy nem a megnyitásra váró állomány neve következik, hanem egy állomány-leíró. Az első lépésben a `tie` által visszaadott objektum `fd()` elemfüggvénye szolgáltatja ezt az adatot. Mivel a második lépésben nem tároltuk külön változóban az objektumot, ezért ennek hivatkozását a tied függvény adja meg.

Az `open` függvényt közvetlenül a `flock()` követi. Ez két paramétert vár. Az első az említett folyam-azonosító, a második a zár típusát határozza meg. Egyedi zárat egyetlen folyamat szerezhet meg, így ezt íráskor szokás használni. A megosztott zárat párhuzamosan több folyamat is megkaphatja, ám ezalatt egyetlen másik sem szerezhet egyedi jogot a zárolásra. Mivel az olvasás egyszerre több folyamat számára is engedélyezhető, az írást viszont tiltani kell, ezt a módszert csak olvasó folyamatok szokták használni.

Van egy további csavar is a fenti példában. Bár előbb történik a kötés, csak utána a megnyitás, mégis előbb a kötést oldjuk fel, és utána zárjuk le a folyamot. Ha egy függvényt és az ellentettjét fordított sorrendben hívjuk meg, rendszerint hiba keletkezik. Esetünkben az `open`-nél nem történt valódi megnyitás, csupán egy folyam-azonosító készült egy meglévő állomány-leíróból, a `close`-nál viszont a lezárás valódi. Ezért ha hamarabb zárjuk le az állományt, mint ahogy a kötést megszüntetnénk, nem az asszociatív tömbben található adatok kerülnek az adatállományba.

A fenti megoldás majdnem tökéletes a versenyhelyzetek elkerülésére. Azért csak majdnem, mert a `tie` által meghívott konstruktor miután megnyitotta az adatállományt, még azelőtt végez egy kis lemezműveletet, hogy a vezérlés az `open` függvényre kerülne. Ezalatt a zárolás sajnos még nem él, de ezt ezzel a módszerrel nem is lehet elkerülni. Az egyedüli gyógyír a beépített zárolás, az `O_EXLOCK` lenne, viszont ez nem mindenhol elérhető.

Egy tábla rekordjai tetszőleges számú mezőből állhatnak, de egy adott táblában minden rekord ugyanannyi mezőt tartalmaz. Vajon lehet-e egy egész rekordot egyetlen kulcs-érték párba sűríteni? A kulcs kerülhet a tábla azonosító me-

```
#!/usr/bin/perl -w

use strict;
use DB_File;
use Fcntl ':flock';

die "Használat: " . $0 . " <adat fájl>\n" unless
    ➥1 == @ARGV;

my %adatbazis;
tie %adatbazis, 'DB_File', $ARGV[0], O_CREAT |
    ➥O_RDWR or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "+<&=" . (tied %adatbazis) ->
fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_EX or
    die "Nem tudom megszerezni az egyedi
        ➥zárat.\n";

print "Neve                          : ";
my $nev = <STDIN>; chop $nev;
print "Telefonszám                   : ";
my $telszam = <STDIN>; chop $telszam;
print "Ezzel veheted le a lábáról : ";
my $szereti = <STDIN>; chop $szereti;

$adatbazis {$nev} = join (":", $telszam,
          ➥$szereti);

untie %adatbazis;
close ADATALLOMANY;
print "Az új lány (" . $nev . ") felvéve.\n";
```

3

```
#!/usr/bin/perl -w

use strict;
use DB_File;
use Fcntl ':flock';

die "Használat: " . $0 . " <adat fájl> <~lány
    ➥neve>\n" unless 2 == @ARGV;

my %adatbazis;
tie %adatbazis, 'DB_File', $ARGV[0], O_RDONLY or
    die "Nem tudom előkészíteni az
        ➥adatbázist.\n";
open ADATALLOMANY, "<&=" . (tied %adatbazis) ->
fd () or
    die "Nem tudom biztonságosan megnyitni az
        ➥állományt.\n";
flock ADATALLOMANY, LOCK_SH or
    die "Nem tudom megszerezni a megosztott
        ➥zárat.\n";

my ($kulcs, $ertek);
while (($kulcs, $ertek) = each %adatbazis) {
    if ($kulcs =~ /$ARGV[1]/) {
       print " Adatlap: " . $kulcs . "\n";
       print "==========" . "=" x length
             ➥($kulcs) . "\n";
       my ($telszam, $szereti) = split (/:/,
          ➥$ertek);
       print "  Telefonszám                  : "
             ➥. $telszam . "\n";
       print "  Ezzel veheted le a lábáról : "
             ➥. $szereti . "\n\n";
    }
}

untie %adatbazis;
close ADATALLOMANY;
```

4

zőjébe, és mivel a ***dbm*** nem szab határt az érték hosszára, az összes többi mezőt is be lehet ide rakni. Ezt pedig megoldható az egyszerű szöveges állománynál már látott elválasztó karakteres, vagy rögzített hosszúságú mezőkből álló láncokkal. Egy ilyen „öszvér" megoldás nemcsak gyorsabb, hanem rugalmasabb is mint egy sima szöveges adatbázis. Végezetül erre is lássunk egy példát. Két szkriptet fogunk elkészíteni. Az elsőnek (*3. lista*) az lesz a feladata, hogy az előző cikkben már látott adathalmazzal feltöltse adatbázisunkat, míg a második (*4. lista*) az adatok lekérdezésére szolgál. Az érték kettősponttal elválasztott mezőket fog tartalmazni, melyeket a `split()` / `join ()` párossal kezelünk. A fent látható programban nincsenek nagy meglepetések. Ez annak köszönhető, hogy a `tie` - `untie` páros segítségével az új elem felvétele mindössze egy változónak való értékadást jelent. Nem kell sokat morfondíroznunk ahhoz sem, hogy kitaláljuk, mi a megoldása a kettős kulcs problémának. Ez a gond ugyanis itt fel sem merül, hiszen ha az asszociatív tömb egy már létező elemére hivatkozunk, akkor az önműködően felülíródik. Az, hogy adott esetben valóban ez az elvárt működés, vagy mégis külön figyelmet kell fordítani a meglévő rekordokra, a feladattól függ.

A kiíratáskor a már megszokott kinézetet próbáltam követni. Természetesen itt nem adható meg, hogy a találat hányadik sorban történt, hiszen egy szótár mindig rendezetlen. Ennek megfelelően elképzelhető, hogy egy adatbázis megváltoztatása után a fent látható `each` szerkezet eltérő sorrendben adja vissza az adatokat. Ez azonban jelen esetben minket nem különösebben érint, legfeljebb két futtatás után más sorrendben látjuk viszont az adatokat.

Akinek van kedve, a fenti példákból kiindulva elkészítheti azt a szkriptet, amely képes egy Berkeley adatbázis egyik rekordját törölni. Mielőtt azonban nekilátnánk, érdemes egy pillantást vetni a `delete()` függvény leírására a ***perlfunc(1)*** kézikönyvlapon. Sok sikert a kísérletezéshez! A jövő hónapban már az SQL lesz terítéken.

Fülöp Balázs