A Perl és az adatbázisok (1. rész)

Az adatbázisok életútja a szöveges állományoktól az SQL-ig a Perl távcsövén keresztül nézve.

Életünket azóta hálózzák be az adatok kezelésével kapcsolatos teendők, amióta írni tudunk. Az általános iskolában bevett szokássá vált barátság-füzeteink után általában valamilyen katalógus következett, amitől már csak egy lépés volt az üzleti kapcsolatainkat és napi teendőinket tároló határidőnapló. Ezeknek az adatoknak a naprakészen tartása, rendszerezése és nem utolsó sorban a jogosulatlan szemektől való távol tartása sok-sok ismétlődő részfeladat révén okozott unalmas perceket mindannyiunk számára. A számítógép pontosan azokra a feladatokra alkalmazható kiválóan, amik – noha feltétlen pontosságot és mérnöki precizitást igényelnek – a munkavégzés során kevés új ötletet vagy izgalmas megoldást várnak el tőlünk. Röviden szólva a számítógép a mai kor tökéletes rabszolgája. Emiatt jogos igény kényes adataink szervezésével és gondozásával kapcsolatos tennivalóinkat hű szolgánkra bízni. Ahhoz azonban, hogy valóban értékes perceket őrizzünk meg magunknak a szabadidőnkből, gondos tervezés kell. A tervezés első lépése, hogy kiválasztjuk azokat az eszközöket, amelyeket fel szeretnénk használni a munkánk során. Az adatfüggetlenítés elvéből nyilvánvalóan következik, hogy két, egymástól jól elkülönített munkához kell segítőt találnunk. Az egyik kizárólag az adatokra összpontosít, azok tárolásával, felügyeletével és szolgáltatásával törődik. A másik közvetít a felhasználó és az első segítő között, azaz tolmácsolja a felhasználó kéréseit és a megadott módon kivonatolja a temérdek adatból azt, amire kíváncsiak vagyunk. Ebben a cikksorozatban a Perl lesz a közvetítőnk. A továbbiakban feltételezem, hogy nemcsak hallottál erről a hihetetlenül rugalmas parancsnyelvről, de már meg is tetted az első lépéseidet ezen a téren, így nem fogom ecsetelni a nyelv alapjait. A Perl azért jó választás erre a feladatra, mert a legegyszerűbbtől a legösszetettebb adatbáziskezelési feladatokig egyszerű és gyors programozási eszközöket bocsát a rendelkezésünkre. Emiatt nem takarja el a nyelvi megvalósítás módszere az adott megoldás elvi lényegét. Az első segítő személyében pedig cikkről cikkre új adatbáziskezelőt fogunk megismerni. A körutazás során az egyszerűtől az egyre összetettebb rendszerek felé haladunk majd, ami azonban nem jelenti azt, hogy ne lehetne használni akár már a legelső részben bemutatásra kerülő módszereket. Az adatok kezelésénél nagyon könnyű abba a hibába esni, amikor valaki ágyúval lő egy verébre. Mindig fel kell mérni az előre látható igényeket, és azokhoz választani adatbáziskezelő rendszert. A mai adatbázisok legnagyobb hányadának alapjait a táblák képezik. Egy tábla sorok előre nem meghatározott számú, rendezetlen gyűjteménye. Minden sor, más néven rekord, adott számú mezőből épül fel, és minden mezőnek határozott típusa van. Természetesen minden sor ugyanannyi, és ugyanolyan típusú mezőt tartalmaz. A mezők típusának szigorúsága a használt adatbáziskezelő rendszertől függ. Egyes rendszerek különbséget tesznek az egész, és lebegőpontos számok között is, mások csak a szöveget különböztetik meg a számtól, megint mások szövegként kezelnek mindent. Egy adatbáziskezelőnek négy alapvető műveletet kell ismernie. A lekérdezés valamilyen feltételnek eleget tevő rekordok összegyűjtése, illetve ezek mezőinek kiválasztása. A tárolás az adatbázisba történő új rekord felvételét jelenti. A módosítás során egy meglévő rekord egy, vagy több mezője frissül. A törlés pedig egy meglévő rekord eltávolítása az adatbázisból. Az adatbáziskezelő kiválasztása során fontos szempont, hogy az említett műveletek átlagosan milyen gyakorisággal kerülnek végrehajtásra.
Legelső adatbázisunk egy sima, szöveges állomány lesz. Adatbáziskezelőről itt nem beszélhetünk, hiszen még az adatbázis belső szerkezetének épségben tartásáról is magának a programozónak kell gondoskodnia. Egy állomány egy táblát tárol, melyben a rekordokat a sortörés, a rekordok mezőit pedig egy előre meghatározott kitüntetett karakter, többnyire a kettőspont, vagy vessző választja el egymástól. Ennek a módszernek pont az egyszerűségében rejlik a szépsége. Néhány száz sor esetén, ahol a leggyakoribb művelet az új rekord felvétele és a lekérdezés, kevés munkával nagy teljesítmény érhető el. Létjogosultságát jelzi az a tény is, hogy számos komoly rendszer sarokköveit a mai napig ilyen adatbázisok alkotják. Elég csak a UNIX */etc/passwd* állományára gondolni, mely tökéletes iskolapéldája az említett módszernek. A példák során egy olyan adatbázissal fogunk dolgozni, mellyel elsősorban a női szívek meghódítását tehetjük zökkenőmentessé. Az adatbázisban női ismerőseink nevét és telefonszámát fogjuk tárolni, továbbá egy olyan apróságot, amivel levehetjük a leányzót a lábáról. Először is tehát hozz létre egy minta adatbázist, ahol kettőspont választja el a mezőket egymástól. Ehhez tetszőleges szövegszerkesztőt használhatsz.

```
Kata:123-4567:kóla
Évi:765-4321:romantikus vígjáték
Mari:135-2467:állatkert
Mónika:753-6421:virágcsokor
```

Most első lépésként írjunk egy Perl programot az adatok lekérdezéséhez. Az alábbi szkript kiírja az adott állományból az összes olyan lány adatait, akinek a neve illeszkedik a szintén paraméterként átadott mintára.

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

use strict;

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

my $talalat = 0;
open LANYOK, $ARGV[0] or die "Nem tudom megnyitni: "
➥. $ARGV[0] . "\n";

while ( <LANYOK> ) {
    chop;
    my ( $nev, $telszam, $szereti ) = split
    ➥( /:/, $_ );
    if ( $nev =~ /$ARGV[1]/ ) {
       print "Találat: " . $nev . " a(z) " . $. .
       ➥". sorban.\n\n";
       print " Adatlap: " . $nev . "\n";
       print "===========" . "=" x length ( $nev )
       ➥. "\n";
       print "  Telefonszám                  : " .
       ➥$telszam . "\n";
       print "  Ezzel veheted le a lábáról : " .
       ➥$szereti . "\n\n";
       $talalat++;
    }
}

close LANYOK;
print "Összesen " . $talalat . " találat.\n";
```

Az, hogy a parancsértelmezőt `-w` kapcsolóval hívjuk meg és a `strict` modult használjuk egy lépés a kultúrált, átlátható, továbbá lehetőség szerint hibamentes program írása felé. A `-w` fontos figyelmeztető üzenetekkel lát el, ha kifogásolható nyelvi szerkezettel éltünk, ezért mindenképpen javasolt a használata. A `strict` modul pedig szigorú ellenőrzési módot ír elő az értelmezőnek. A második sorban a programnak átadott paraméterek számát ellenőrizzük. Ha egy tömböt skalárként értelmezünk, akkor a tömb elemeinek a számát kapjuk. Így a megadott, összefűzött karakterlánc jelenik meg a képernyőn, és a hibára jellemző visszatérési értékkel kilép a program, ha az átadott paraméterek száma nem egyenlő pontosan 2-vel. A `$talalat` nevű változóban fogjuk tárolni a találatok számát. Ez csupán ahhoz szükséges, hogy a keresés végeztével megjelenítsünk egy összegző sort. A következő sorban megnyitjuk az első paraméterben szereplő állományt olvasásra, így a továbbiakban `LANYOK` fájlleíróval hivatkozhatunk rá. Amennyiben a megnyitás nem volt sikeres, hibaüzenettel kilépünk.

A főciklusban soronként dolgozzuk fel a bemeneti állományt. Minden körben a `$_` alapértelmezett változó fogja tartalmazni a beolvasott sort. Minden sorról levágjuk a sorvége jelet, majd beszédes változónevekbe kigyűjtjük a rekord mezőit. Ha a név mezőre illeszkedik a paraméterként megadott minta, akkor kinyomtatjuk a képernyőre az aktuális rekord adatlapját, továbbá megnöveljük a `$talalat` értékét. A találatot bemutató jelentést a lehető legízlésesebben jelenítjük meg. Először a `$.` különleges változót felhasználva kiíratjuk, hogy a találat melyik sorban történt. Ezután egy sor kihagyással következik az adatlap. Ennek fejlécét hosszának megfelelően aláhúzzuk, majd kinyomtatjuk az egyes mezők tartalmát. A program végén lezárjuk az adatbázist, utolsó lépésként pedig kiíratjuk a képernyőre a találatok számát. A megvalósítás hibája, hogy nem szerepelhet az elválasztó karakter egyik mezőben sem. Elképzelhető, hogy repülő ékezeteket szeretnél használni az adatbázisban, ekkor azonban a rövid ö betűvel gondjaid támadhatnak. Erre a problémára bevett megoldás a rögzített hosszúságú mezők használata. Ez esetben elválasztó karakterre nincs is szükség, így átvágtad a gordiuszi csomót. A második programban megvalósítjuk a tárolás műveletet. Programozási oldalról ez a lehető legkönnyebb, és az erőforrásokat is ebben az esetben vesszük a legkevésbé igénybe. Önmagában a tárolás művelethez nincs szükség az állomány beolvasására. A program egyszerűen hozzáírásra nyitja meg az adatbázist, és egy új sorral gazdagítja a meglévő állományt.

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

use strict;

die "Használat: " . $0 .
    " <adat fájl> <lány neve> <telefonszám>
    ➥<ezt szereti>\n"
    unless 4 == @ARGV;

open LANYOK, ">>" . $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";

print LANYOK join ( ":", $ARGV[1], $ARGV[2],
➥$ARGV[3] ) . "\n";

close LANYOK;
print "Az új lány (" . $ARGV[1] . ") felvéve.\n";
```

Egy kifinomultabb alkalmazásban viszont lehetséges, hogy nem megengedhető két azonos kulcs létezése. Ennek az ellenőrzéséhez előbb az első példában bemutatott módszerrel végig kell futni az állományon, és mikor meggyőződtünk arról, hogy nincs azonos kulcs, utána hozzáfűzni az új rekordot. A rekordok módosítása sima szöveges állományban tárolt adatbázis esetén már nehézkes. A fő gond az, hogy miután beolvastuk az adatokat és bemutattuk a megfelelő bűvészmutatványokat a kiszemelt rekordokon, a módosított eredményhalmazt vissza kell írni eredeti helyére. Mindezt természetesen úgy kell megtennünk, hogy nem veszítünk közben adatot. Kétféle megközelítésben foghatunk neki a probléma megoldásának. Az első, hogy beolvassuk a teljes állományt a memóriába, frissítjük a szükséges rekordokat és

visszaírjuk az eredményt. Ennek azonban egy nagyobb adatbázis esetén feltétele a hatalmas memória és egyenes következménye a költséges műveletek okozta teljesítményvisszaesés. Járhatóbb út, ha rekordról rekordra írunk egy átmeneti állományba, végül ezt kicseréljük az eredetivel.

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

use strict;

die "Használat: " . $0 .
    " <adat fájl> <lány neve> <uj telszám>\n"
    ➥unless 3 == @ARGV;

open REGILANYOK, $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";
open UJLANYOK, ">tmp." . $$ or die "Nem tudok új
➥fájlt létrehozni.\n";

while ( <REGILANYOK> ) {
    my ( $nev, $telszam, $szereti ) = split
    ➥( /:/, $_ );
    if ( $nev eq $ARGV[1] ) {
       $telszam = $ARGV[2];
       print "Módosítás történt a(z) " . $. . ".
       ➥sorban.\n";
    }
    print UJLANYOK join ( ":", $nev, $telszam,
    ➥$szereti );
}

close UJLANYOK;
close REGILANYOK;
unlink $ARGV[0] or die "Nem tudom törölni: " .
➥$ARGV[0] . "\n";
rename "tmp." . $$, $ARGV[0] or die "Nem tudok
➥átnevezni.\n";
print "Kész.\n";
```

Tehát két állományt nyitunk meg a főciklus előtt. Az egyik, melyet a `REGILANYOK` állományleíróval érünk el később, olvasásra nyitjuk meg. Ez az eredeti adatbázisunk, melyet a program végén felváltunk az újjal. A másik, az `UJLANYOK` azonosítójú, egy átmeneti állomány, mely az adott könyvtárban ***tmp.$$*** névvel jön létre, ahol `$$` a programunk folyamat-azonosítója. Ebbe írjuk ki egyenként a rekordokat a `REGILANYOK`-ból, a szükséges módosításokat elvégezve.

A főciklusban sorról sorra dolgozzuk fel az eredeti adatbázist. Minden sort felbontunk és ellenőrizzük, hogy szükséges-e a módosítás. Ha igen, felülírjuk az adott mezőt és erről azonnal tájékoztatjuk a felhasználót is. Végül egyesítjük a szétbontott sort és kiírjuk az átmeneti adatbázisba. Javítható az algoritmus hatékonysága, ha még a felbontás előtt végzünk egy előzetes mintaillesztést a kulcsra. A ciklus végén mindkét állományleírót bezárjuk. Ezután eltávolítjuk az eredeti adatbázist, és átnevezzük az átmenetit. Végül jelezzük a felhasználónak, hogy a folyamat elkészült. Ez a megoldás meglehetősen veszélyes, hiszen ha a törlés sikeres volt, de az átnevezés nem, akkor csak kézi beavatkozással lehet megmenteni az adatbázist.

Már csak a törlés művelet maradt hátra. Ennél egy az egyben ugyanazt az algoritmust követjük, mint az előző esetben, leszámítva azt, hogy a feldolgozás során nem módosítunk a rekordokon, hanem csak a szükségeseket írjuk ki.

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

use strict;

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

open REGILANYOK, $ARGV[0] or die "Nem tudom
➥megnyitni: " . $ARGV[0] . "\n";
open UJLANYOK, ">tmp." . $$ or die "Nem tudok új
➥fájlt létrehozni.\n";

while ( <REGILANYOK> ) {
    if ( /^$ARGV[1]:/ ) {
       print "Törlés történt a(z) " . $. . ".
       ➥sorban.\n";
       next;
    }
    print UJLANYOK $_;
}

close UJLANYOK;
close REGILANYOK;
unlink $ARGV[0] or die "Nem tudom törölni: " .
➥$ARGV[0] . "\n";
rename "tmp." . $$, $ARGV[0] or die "Nem tudok
➥átnevezni.\n";
print "Kész.\n";
```

Mint látható, a költséges `split() / join()` páros helyett egy egyszerű mintaillesztés történt a rekord legelső mezőjére. Ha a főciklusban az átmeneti változóval kapcsolatban hiányérzeted van, képzeld oda a `$_` változót, mely az alapértelmezett minden helyzetben. Máris látni fogod, hogy miért vonatkozik a ciklus elején, az `if` feltétel részében a mintaillesztés a teljes beolvasott sor első mezőjére.

A bemutatott megvalósítások egy további fogyatékossággal is rendelkeznek. Ha például két felhasználó egy-egy folyamattal párhuzamosan akar törölni két különböző rekordot, versenyhelyzet alakul ki. Aki gyorsabb volt, annak a törlés művelete varázslatos módon visszavonásra kerül. Erre megoldás a Perl `flock()` függvénye. Ezzel egy átmeneti zárat helyezhetünk az adatbázisra. A `flock()` két típusú zárat ismer. Az egyik a kizárólagos (***exclusive***), a másik a megosztott (***shared***). Ha egy folyamat feltette a kizárólagos zárat egy állományra, egyetlen másik folyamat sem kaphatja meg, amíg a zár fel nem oldódik. A megosztott zárat párhuzamosan több folyamat is megszerezheti, ám ez alatt kizárólagos zárat nem lehet rátenni az állományra. Többnyire csak olvasó folyamatok használják a megosztott, míg író folyamatok a kizárólagos zárat. Sok minden szerepelt, és nagyon sok más kimaradt ebből az írásból. Viszont, mint mondani szokás, a puding próbája az evés. Kísérletezz, játssz, és egyre többre fogsz rájönni. Sok sikert!

Fülöp Balázs