Statisztikai szoftverek – Molnár Gábor József

SAS-ban megírt programok megvalósítása SQL-lel

Bevezetés

A SAS, statisztikai szoftver, egy önálló adatkezelési nyelvvel rendelkezik; ez a SAS BASE amely segítségével alap statisztikákat számolhatunk adatállományainkból. A SAS BASE azonban nem minden esetben a legalkalmasabb. Amikor gyorsabb futást szeretnénk elérni, sokszor célszerűbb az adatbáziskezelőknél gyakran alkalmazott SQL nyelvet használni. Ezt a SAS lehetővé teszi számunkra a proc sql segítségével. Természetesen ez a nyelv is rendelkezik hátrányokkal. Ezekre is rámutatok a következő példákban.

A feladatok lekódolása során célunk nem az azok futásából származó eredmények elemzése, hanem a különbözőképpen megírt programok összehasonlítása, futási idejük, valamint memóra felhasználásuk megfigyelése.

Utóbbi két tulajdonság mérésére lesz segítségül a SAS BASE <*options fullstimer*> parancsa, amely futás után logolja a fent említett mérési eredményeket. Ennek kódbaágyazására csak az első program lefutásakor van szükség; a következő futtatásokkor minden esetben megjelenik a tárgyalt tulajdonságok a log ablakban.

Megjegyzés: a feladatok bemutatása során először a SAS majd az SQL-es változat kerül bemutatásra.

Feladatunk, az órán közösen kidolgozott minta1.sas és minta2.sas SAS BASE-ben megírt programokat megírjuk SQL nyelven, és összehasonlítsuk a futási-, ill. memóriaigényüket.

A programokat részenként mutatjuk be.

Feladatok

1. feladat: minta1.sas, minta1sql.sas

A program célja megmutatni az egyes ügyfelek nevét, lakóhelyét, egyenlegét forintban és hogy mekkora összegű szerződést kötöttek.

5 különböző táblánk van: egy az árfolyamokat (**Arfolyam**), egy a fedezeteket (**Fedezet**), egy a hiteleket (**Hitel**), egy a számlákat (**Szamla**) és egy az ügyfeleket (**Ugyfel**) tartalmazó.

A feladat megoldásához két fő lépést kell végrehajtanunk.
Első lépésben összekapcsoljuk a táblákat olyan módon, hogy egy sorban láthassuk az ügyfelek adatait és az általuk felvett hitelhez kapcsolódó adatokat. Ezt a műveletet merge-elésnek nevezzük. Ahhoz, hogy a SAS megfelelően csatlakoztassa a két táblát az kell, hogy az összefűzendő (merge-elendő) táblák rendezve legyenek aszerint, ami szerint az összekapcsolást végezni akarjuk.
Tehát két különböző utasításra van szükségünk: a proc sort – a két táblára külön-külön végrehajtva, majd a rendezést követően a merge.

```
1.  proc sort data=Kotprogi.ugyfel;
2.  by ugyfel_kod;
3.  run;
4.  proc sort data=Kotprogi.hitel;
5.  by ugyfel_kod;
6.  run;
7.  data Kotprogi.hitel2;
8.    merge
9.      Kotprogi.hitel
10.     Kotprogi.ugyfel
11. ;
12.   by ugyfel_kod;
13.   run;
```

SQL-ben azonban ezt az egészet egyszerre is elvégezhetjük. Az SQL 4 különböző csatlakozást[1] kínál fel. A merge parancs megegyezik a full joinnal. Nekünk arra van szükségünk, hogy azok az ügyfelek is láthatók legyenek a táblában, akik nem vettek fel hitelt. És azok is, akik nem szerepelnek az **Ugyfel** táblában. Ettől teljes.
Az SQL-es megoldás azonban így nem egész, hiszen ha olyan ügyfelünk lenne, akinek a kódja nem szerepel a **Hitel** táblában (vagyis, aki nem vett fel hitelt), akkor a keletkező új táblában az ügyfelünk kódja helyén egy üres cella állna. Ennek kiküszöbölésére használjuk a coalesce parancsot, ami több paramétert kaphat. Ha az egyik paramétere üres cellát

[1] További join típusokat lásd [1]-ben.

eredményezne az új táblánkban, akkor helyette a másik paraméterének értékét helyezzük el.

```
proc sql;
        create table Kotprogi.Hitel4 as
                select
                        coalesce(Hitel.UGYFEL_KOD, Ugyfel.UGYFEL_KOD) as
                        UGYFEL_KOD,
                        *
                from Kotprogi.Hitel as hitel
                         full join
                         Kotprogi.Ugyfel as ugyfel
                on hitel.UGYFEL_KOD = ugyfel.UGYFEL_KOD
                order by DEVIZANEM_KOD, UGYFEL_KOD
        ;
quit;
```

Most tekintsük a memóriafelhasználást és az időigényt. A bevezetőben említett parancsot (*options fullstimer*) használva kapjuk meg az eredményeket. SAS Base esetén a futás 21.28 s alatt ment végbe, 555 kB-ot felhasználva, addig az SQL-es megoldás 26.17 s-ig futott 754 kB-ot használva. Tehát hiába egyszerűbb/rövidebb a megoldás SQL-ben, a SAS-hoz kevesebb memória kellett, és egy kicsit kevesebb idő is.

A feladat azonban ezzel még nincs elvégezve. Az elkészült új táblát össze kell merge-elnünk az **Arfolyam** táblával, magyarul el szeretnénk érni, hogy minden deviza mellett megjelenjen annak árfolyama is, hogy kiszámíthassuk forintbeli értéküket.

Az előzőhöz hasonlóan a két összekapcsolandó táblát rendeznünk kell, ezt követően történhet meg az összekötés. Ezenkívül új változókat is bevezetünk az egyenlegek és a szerződés nyilvántartásához forintban. Az utóbbiak közül a huf_egyenleg2 érdekes számunkra. Ez egy olyan értéket tárol, ami euró esetén 30%-kal megnöveli az eredeti egyenleget.

```
proc sort data=Kotprogi.hitel2;
   by devizanem_kod;
        run;
proc sort data=Kotprogi.arfolyam;
   by devizanem_kod;
        run;
data Kotprogi.hitel3;
  merge
    Kotprogi.hitel2
    Kotprogi.arfolyam;
 by devizanem_kod;
        if tipus='H';
        huf_egyenleg=egyenleg_osszeg*arfolyam;
        huf_szerzodes=szerzodes_osszeg*arfolyam;
        if devizanem_kod='EUR'
     then huf_egyenleg2=egyenleg_osszeg*1.3;
```

```
17.          else huf_egyenleg2=huf_egyenleg;
18.  run;
```

SQL-ben nincs változás a join műveletben. Itt is használunk *coalesce* parancsot a *null* értékek elkerülése érdekében. A különbség az új változók létrehozásánál van. Ezeket a *select* utasításban adhatjuk meg. Alias nevet pedig az *alias* paranccsal adhatunk.

A huf_egyenleg2-őt szintén a *select* utasításban vehetjük fel, de mivel értéke feltételhez kötött, ezért egy újabb utasítást, a *case*-t használjuk.

Ennek szintaktikája:

case <*változó*>
    when <*a változó értéke*>
    then <*utasítás*>
    else <*utasítás*>
    …
end

Mindkét nyelv esetén azokat a sorokat kívántuk kiszűrni, ahol a hitel típusa „H” (azaz hitel – ez minden sornál teljesül), ezt SAS esetén a merge-elést követően egy *if*-fel tehetjük meg, SQL-ben pedig a *where* záradékkal.

```
1.  proc sql;
2.          create table Kotprogi.Hitel5 as
3.                  select
4.                          coalesce(Arfolyam.DEVIZANEM_KOD,    Hitel4.DEVIZANEM_KOD)
                            as DEVIZANEM_KOD,
5.                          *,
6.                          EGYENLEG_OSSZEG*arfolyam as huf_egyenleg,
7.                          SZERZODES_OSSZEG*arfolyam as huf_szerzodes,
8.                          case Hitel4.DEVIZANEM_KOD
9.                                  when 'EUR'
10.                                 then EGYENLEG_OSSZEG*1.3
11.                                 else EGYENLEG_OSSZEG*arfolyam
12.                         end as huf_egyenleg2
13.                 from Kotprogi.Hitel4 as hit
14.                          full join
15.                          Kotprogi.Arfolyam as arf
16.                 on hit.DEVIZANEM_KOD = arf.DEVIZANEM_KOD
17.                 where TIPUS like 'H'
18.                 order by DEVIZANEM_KOD, UGYFEL_KOD
19.         ;
20. quit;
```

A futási időt és memórahasználatot illetőleg az előbbihez hasonló eredmények születtek. SAS-nál 25.51 s és 772 kB kellett a futáshoz, SQL-nél viszont 4.34 s és 1107 kB.

Lényeges különbség az előző futás és az újabb között, hogy az SQL-es futás csupán néhány másodpercig tartott. Ezt magyarázhatja az, a korábbi lépésben létrejövő táblához kapcsolt **Arfolyam** tábla kis mérete.

Végül a kiiratás maradt hátra. Ez SAS-ban a proc printtel, SQL-ben pedig egy választó lekérdezéssel történik. Különbséget fedezhetünk fel a kapott eredmény táblázatok között. SAS esetén egy általunk ki nem választott oszlop is áll. Ez az Obs, ami arra utal, hogy az adott sor a kiírásra kerülő tábla hányadik sora eredetileg.

A lekérdezés futása 0.28 s-ig tartott, 205 kB-t felhasználva, a proc print pedig 0.48 s-ig, 131 kB-t felhasználva. Az értékek nem mutatnak nagy eltérést. Ennek kimutatásához viszont nagyobb adatbázisra lenne szükségünk.

2. feladat: minta2.sas, minta2sql.sas

2.a. feladat: a program célja lakóhely és azonbelül devizanem szerint csoportosítva megadni az egyenleget devizában és forintban. Kíváncsiak vagyunk a devizában csoportosított összes egyenlegre is.

A feladat megoldására a SAS Base proc tabulate-je szolgál, amelyen belül egy külön class utasítása segítségével adható meg, hogy mi alapján csoportosítunk (sorok); var utasítással adható meg, hogy mit akarunk csoportosítani (oszlopok); és a table utasítással adható meg, hogy mi is szerepeljen a sorok és oszlopok találkozásánál.

```
1.  proc tabulate data=Kotprogi.hitel3 format=ert.;
2.  class ugyfel_lakhely devizanem_kod;
3.  var egyenleg_osszeg huf_egyenleg huf_egyenleg2;
4.  table ugyfel_lakhely='*devizanem_kod=' all='Összesen'
5.        *devizanem_kod=',
6.        egyenleg_osszeg='Egyenleg devizában'*sum='
7.        huf_egyenleg='Egyenleg HUF250'*sum='
8.        huf_egyenleg2='Egyenleg HUF300'*sum='
9.        /box='Lakóhely / Devizanem';
10. run;
```

SQL esetében egy select utasításban kell megadnunk a lakóhelyet, a devizanemet és az egyenleg összegeket. Tehát a SAS-tól eltérően itt azonos tulajdonságú oszlopként kezelve az egyébként oszlop- és sorfejléceket. A SAS megoldástól különbözik az is, hogy itt minden sorban szerepel a lakóhely (a proc table egyesítve a cellákat, ügyesen csak egyszer írja ki). Valamint, hogy az Összesen sort egy újabb select-tel kell összefűzni az előbbi select kimenetéhez egy union paranccsal.

A kapott kimutatásban az SQL-es megoldásnál nem volt lehetőség a sorfejléc cellák háttérszínének megváltoztatására.

```
1.  proc sql;
2.          select
3.                  UGYFEL_LAKHELY as lakohely label='Lakóhely',
4.                  DEVIZANEM_KOD as devizanem label='Devizanem',
5.                  sum(egyenleg_osszeg) label='Egyenleg devizában' format=comma17.1,
6.                  sum(huf_egyenleg) label='Egyenleg HUF250' format=comma17.1,
7.                  sum(huf_egyenleg2) label='Egyenleg HUF300' format=comma17.1
8.          from Kotprogi.Hitel3
9.          group by lakohely, devizanem
10.
11.         union
12.
13.         select
14.                 'Összesen' as UGYFEL_LAKHELY,
15.                 DEVIZANEM_KOD as devizanem,
16.                 sum(egyenleg_osszeg),
17.                 sum(huf_egyenleg),
18.                 sum(huf_egyenleg2)
19.         from Kotprogi.Hitel3
20.         group by devizanem
21.         ;
```

```
22. quit;
```

A futási eredményekben egyértelműen a proc tabulate viszi a pálmát. Az SQL-es 712 kB helyett 184 kB memóriát fogyasztott. Időbeli eltérés pedig nem volt.

2.b feladat: kérjük le lakóhely szerinti csoportosításban az egyenleget minden devizanem esetén, valamint az egyenlegeket forintban is.

A SAS megoldás esetén nincs más teendőnk, mint az előbb alkalmazott proc tabulate-et hasonlóan alkalmazni.

```
1. proc tabulate data=Kotprogi.hitel3 format=ert.;
2. class ugyfel_lakhely devizanem_kod;
3. var egyenleg_osszeg huf_egyenleg huf_egyenleg2;
4. table ugyfel_lakhely='' all='Összesen',
5.       devizanem_kod=''*egyenleg_osszeg='Egyenleg devizában'*sum=''
6.       huf_egyenleg='Egyenleg HUF250'*sum=''
7.       huf_egyenleg2='Egyenleg HUF300'*sum=''
8.       /box='Lakóhely';
9. run;
```

SQL esetén viszont egy bonyolultabb a dolgunk. Ugyanis ahhoz, hogy minden devizanem esetére kiszámoljuk az egyenleget külön lekérdezéseket kell létrehoznunk. Ezeket kell a selectbe beszúrnunk, így kapjuk a táblázat oszlopait. Majd ugyanezt meg kell ismételnünk mégegyszer, ha az Összesen értékekre, azaz lakóhelyet tekintve kollektív egyenlegre vagyunk kíváncsiak. Természetesen egy újabb union parancsot kell használnunk a lakóhely szerinti és a végső összesítő sor összefűzéséhez.

```
1. proc sql;
2.         select
3.                 UGYFEL_LAKHELY as lakohely label='Lakóhely',
4.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                   DEVIZANEM_KOD like 'EUR' and UGYFEL_LAKHELY like lakohely)
5.                         format=comma17.1 label='EUR Egyenleg devizában' as euro,
6.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                   DEVIZANEM_KOD like 'HUF' and UGYFEL_LAKHELY like lakohely)
7.                         format=comma17.1 label='HUF Egyenleg devizában' as forint,
8.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                   DEVIZANEM_KOD like 'USD' and UGYFEL_LAKHELY like lakohely)
9.                         format=comma17.1 label='USD Egyenleg devizában' as dollar,
10.                sum(huf_egyenleg) label='Egyenleg HUF250' format=comma17.1,
11.                sum(huf_egyenleg2) label='Egyenleg HUF300' format=comma17.1
12.        from Kotprogi.Hitel3
13.        group by lakohely
14.
15.        union
16.
17.        select
18.                'Összesen' as UGYFEL_LAKHELY as lakhely,
```

```
19.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                    DEVIZANEM_KOD like 'EUR'),
20.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                    DEVIZANEM_KOD like 'HUF'),
21.                 (select sum(egyenleg_osszeg) from Kotprogi.Hitel3 where
                    DEVIZANEM_KOD like 'USD'),
22.                 sum(huf_egyenleg),
23.                 sum(huf_egyenleg2)
24.         from Kotprogi.Hitel3
25.         group by lakhely
26.         ;
27. quit;
```

A megjelenő táblázatban újra színezési és cellaegyesítési hiányosságokat fedezhetünk fel az SQL-es megoldásnál.

A futási értékek önmagukért beszélnek: míg az SQL kódnak 1.78 s és 1 MB-ra volt szüksége, addig a SAS-nak csak 0.38 s-ra és 184 kB-ra.

Konklúzó

Elmondhatjuk, hogy nem használhatjuk egyöntetűen minden feladatra ugyanazt a nyelvet, hiszen míg a tabulate eljárásoknál egyértelműen a SAS győzedelmeskedik memóriafelhasználásban, addig a merge-elésnél az SQL-es megoldások eredményezhetnek gyorsabb megoldásokat.

Hivatkozások

[1] Katona Endre: http://www.inf.u-szeged.hu/~katona/db-ea1.pdf
[2] Judy Loren: An advenced tutorial for business users