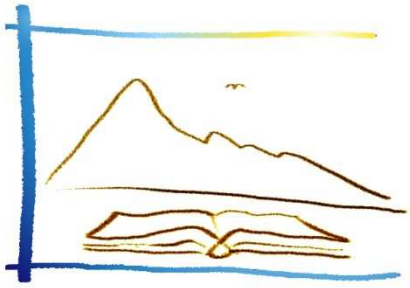


KONSTANTINUSZ

CONSULT TANÁCSADÓ KFT.

Analitikus és egyéb hasznos függvények Oracle 11g alatt

1 TARTALOMJEGYZÉK

2 BEVEZETÉS

Napjaink informatikai környezete lehetővé teszi az információk nagy mennyiségű tárolását, és az ezekben történő gyors keresést. Erre alapozva egyre nagyobb szerepet játszanak a döntés támogató rendszerek az élet minden területén. A vezetők egyre nagyobb mértékben támaszkodnak az évek során felhalmozódott adatok elemzéséből származó információkra. Az adatok elemzése többnyire erőforrás igényes művelet, ezért a relációs adatbázis kezelő rendszerek különböző eszközökkel támogatják a nagy mennyiségű adatok analízisét.
Jelen dokumentum az Oracle 11g által nyújtott néhány eszközt és azok használatát taglalja példákon keresztül. Ezen eszközök nagy részét a szaknyelv analitikus függvényeknek nevezi.

3 AZ ANALITIKUS FÜGGVÉNYEKRŐL ÁLTALÁNOSAN

Az analitikus függvények általános formája a következő:

Függvény_név(arg1,..., argn) OVER ([PARTITION BY <...>] [ORDER BY <....>] [<ablak_definíció>])

Az analitikus függvények kiértékelése az összes *join* és a *where* ágon felsorolt feltételek kiértékelése után történik meg.
A formula az alkalmazni kívánt függvény nevével és annak paramétereivel kezdődik.
A továbbiakban részletezni fogom a formula egyes részeit, majd példákon keresztül megnézzük azok használatát.

3.1 Az ORDER BY rész

Egy partíción belül a rekordok sorrendjét az *order by* rész segítségével tudjuk befolyásolni. Egyes függvények (pl.: Lead, Lag, Rank, stb.) kimenetét befolyásolja a rekordok partíción belüli sorrendje, másokét nem (pl. Sum, Avg, Min, stb.)
Az *order by* rész általános formája a következő:

ORDER BY <sql_kif> [ASC|DESC] NULLS [FIRST|LAST]

Az [ASC | DESC] résszel tudjuk befolyásolni, hogy a halmaz rendezettsége növekvő vagy csökkenő legyen.
A NULLS [FIRST | LAST] résszel pedig azt mondjuk meg, hogy a rendezettség szerint a null értékek a halmaz elejére vagy végére kerüljenek.

3.2 A PARTITION BY rész

A *partition by* használatával az eredmény halmaza csoportosítható, mely csoportokon aggregációk hajthatók végre. Jogosan merül fel a kérdés, hogy akkor mi a különbség a *partition by* záradékkal ellátott analitikus függvény és egy *group by* záradékkal ellátott lekérdezés között.

A legfontosabb különbség talán az, hogy míg a *group by* záradékkal ellátott lekérdezés *select* ágán nem szerepelhet olyan oszlop definíció, amely nem szerepel a *group by* ágon, addig az analitikus függvényeknél nincs ilyen megkötés, viszont az aggregáció ugyan úgy elvégezhető. Azaz az analitikus függvények úgy végzik el a csoportosítást és rajtuk az aggregációt, hogy a megjelenő eredményhalmaz ténylegesen nem lesz csoportosítva.

3.3 Az ablak_definíciós rész

Néhány analitikus függvény támogatja az *ablak_definíció* használatát, melynek segítségével tovább szűkíthetjük a partíción belüli rekordok számát, oly módon, hogy a partíción belül meghatározzuk az ablak kezdetét és a végét. Ezeket a határokat az aktuális sorhoz képest relatívan tudjuk megadni. Kétféle ablaktípus létezik, a ROW és a RANGE.
Az ablak definíció általános szintaxisa a következő:

[ROW | RANGE] BETWEEN <kezdőpont_kif> AND <végpont_kif>

ahol a <kezdőpont_kif> a következőképpen nézhet ki:
(UNBOUNDED PRECEDING | CURRENT ROW | <sql_kif> [PRECEDING | FOLLOWING])

ahol a <végpont_kif> a következőképpen nézhet ki:
(UNBOUNDED FOLLOWING | CURRENT ROW | <sql_kif> [PRECEDING | FOLLOWING])

3.3.1 ROW típus esetén

Az UNBOUNDED PECEDING jelentése az aktuális sort megelőző partíción belüli első sor. Ennek analógiájára az UNBOUNDED FOLLOWING az aktuális sort követő partíción belüli utolsó sort fogja jelenteni.
A CURRENT ROW az aktuális sort jelenti.
Az <sql_kif> PRECEDING az aktuális sort <sql_kif>-el megelőző sort, az <sql_kif> FOLLOWING pedig az aktuális sort követő <sql_kif>-dik sort jelenti, ahol az <sql_kif> értékének pozitív egésznek kell lenni.
A kezdőpontnak mindig kisebbnek kell lenni a végpontnál.

3.3.2 RANGE típus esetén

A RANGE típus esetén a szintaxis ugyan az mint a ROW típus esetén, csupán értelmezésbeli különbség van, valamit további megkötések, melyek a következők.

- Az *order by* záradék csak egy kifejezést tartalmazhat.
- A <kezdőpont_kif> és a <végpont_kif>-ben szereplő <sql_kif> típusának úgymond kompatibilisek kell lenni az *order by* záradékban szereplő kifejezés típusával. Ez azt jelenti, hogy ha a kifejezés típusa *number*, akkor az *order by* kifejezésnek *number* vagy *date* típusúnak kell lenni. Ha a kifejezés egy intervallum típus, akkor az *order by* kifejezésnek *date* típusúnak kell lenni.

A RANGE típus értelmezésében a kezdő és a végpont nem más, mint az *order by* által meghatározott oszlop aktuális értékének a kifejezéssel történő eltolása.

4 ANALITIKUS FÜGGVÉNYEK HASZNÁLATA

4.1 A LAG és a LEAD függvények

A LAG és a LEAD függvények segítségével egy halmaz bármely során állva elkérhetjük egy az általunk definiált rendezési reláció szerinti az adott sort x-el megelőző (Lag) illetve követő (Lead) sor egy oszlopának értékét.

```
LEAD (<sql_kifejezés>, <eltolás>, <alapértelmezett_érték>)
OVER ([PARTITION BY <...>] [ORDER BY <....>])
LAG (<sql_kifejezés>, <eltolás>, <alapértelmezett_érték>)
OVER ([PARTITION BY <...>] [ORDER BY <....>])
```

Mindkét függvénynek 3 paramétere van:

- sql kifejezés: Egy szabványos sql kifejezés, mely ki lesz értékelve a megelőző vagy a következő soron.
- eltolás: Egy egész szám, mely megmondja hány sorral megelőző illetve követő soron értékelődjön ki az első paraméter.
- alapértelmezett érték: Egy alapértelmezett érték mely akkor kerül visszaadásra, ha a kiértékelt kifejezés eredménye null.

4.1.1 Példa a LAG és a LEAD függvények használatára

Tegyük fel, hogy szükségünk van a dolgozók nevének és fizetésének listájára abc sorrendben úgy, hogy minden sorba oda kell írnunk azt is, hogy az előző és a következő sorban mennyi volt a fizetés. Amennyiben nincs megelőző vagy következő sor úgy 0-t írjunk a megfelelő helyre. Ez a probléma a következőképpen oldható meg.

```
SELECT dolg.név "Név",
        dolg.fizetés "Fizetés",
        LAG(dolg.fizetés,1,0) OVER (ORDER BY dolg.név) "Előző sor fizetése",
        LEAD(dolg.fizetés,1,0) OVER (ORDER BY dolg.név) "Következő sor fizetése"
FROM aa dolg;
```

A lekérdezés eredménye a következő:

	Név	Fizetés	Előző sor fizetése	Következő sor fizetése
1	Attila	130000	0	100000
2	Béla	100000	130000	80000
3	Éva	80000	100000	60000
4	Géza	60000	80000	100000
5	Laci	100000	60000	100000
6	Pisti	100000	100000	110000
7	Tünde	110000	100000	0

4.2 A ROW_NUMBER, RANK és DENSE_RANK függvények

A függvények szintaxisa a következő:

```
ROW_NUMBER () OVER ([PARTITION BY <...>] [ORDER BY <....>])
RANK () OVER ([PARTITION BY <...>] [ORDER BY <....>])
DENSE_RANK () OVER ([PARTITION BY <...>] [ORDER BY <....>])
```

Az említett függvények mindegyike egy sorszámot ad az eredmény halmaz minden egyes sorának egy rendezési relációt alapul véve. A rendezési relációt a már fentebb említett *order by* záradék segítségével tudjuk definiálni. Az eltérés a sorszámok kiosztásába van, melyet a következő szabály határoz meg.

- ROW_NUMBER() : A row_number esetén a sorszámok szigorúan monoton növekvő sort alkotnak, ahol az N. elem a halmazban az N. sorszámot kapja.
- RANK() : A rank esetén a sorszámok monoton növekvő sort alkotnak. Abban az esetben különbözik a row_number-től, ha a rendezési reláció szerint a halmaz tartalmaz azonos sorokat. Ilyen esetben, ha az N. és az N+1. elem a rendezési reláció szerint egyenlő, akkor az N. és az N+1. elem is N. sorszámot kapja, azonban a rendezési reláció szerinti következ N+2. eltérő elem az N+2. sorszámot kapja.
- DENSE_RANK() : A dense_rank esetén a sorszámok monoton növekvő sort alkotnak. Ez is abban az esetben különbözik a row_number-től, ha a rendezési reláció szerint a halmaz tartalmaz azonos sorokat. Abban különbözik a rank függvénytől, hogy itt a rendezési reláció szerinti következő N+2. eltérő elem az N+1. sorszámot kapja.

4.2.1 Példa a ROW_NOMBER, RANK és DENSE_RANK függvények használatára

A három függvény segítségével rangsoroljuk a dolgozókat, a fizetésük szerinti rendezettségük alapján.

```
SELECT row_number() OVER(ORDER BY dolg.fizetés) row_number,
        rank() OVER(ORDER BY dolg.fizetés) rank,
        dense_rank() OVER(ORDER BY dolg.fizetés) dense_rank,
        dolg.név,
        dolg.fizetés
FROM aa dolg;
```

A lekérdezés eredménye a következő:

	Row_number	Rank	Dense_rank	Név	Fizetés
1	1	1	1	Géza	60000
2	2	2	2	Éva	80000
3	3	3	3	Pisti	100000
4	4	3	3	Béla	100000
5	5	3	3	Laci	100000
6	6	6	4	Tünde	110000
7	7	7	5	Attila	130000

Amennyiben ugyan ezt a rangsorolást a telephelyen belül szeretnénk megtenni, úgy használnunk kell a *partition by* záradékot.

```
SELECT row_number() OVER(PARTITION BY dolg.telep ORDER BY dolg.fizetés) row_number,
       rank() OVER(PARTITION BY dolg.telep ORDER BY dolg.fizetés) rank,
       dense_rank() OVER(PARTITION BY dolg.telep ORDER BY dolg.fizetés) dense_rank,
       dolg.név,
       dolg.fizetés,
       dolg.telep
FROM aa dolg;
```

Ezesetben az eredmény a következő:

	ROW_NUMBER	RANK	DENSE_RANK	NÉV	FIZETÉS	TELEP
1	1	1	1	Géza	60000	Böszörmény
2	2	2	2	Éva	80000	Böszörmény
3	3	3	3	Tünde	110000	Böszörmény
4	1	1	1	Pisti	100000	Debrecen
5	2	1	1	Béla	100000	Debrecen
6	3	1	1	Laci	100000	Debrecen
7	4	4	2	Attila	130000	Debrecen

4.2.2 Futtatási tervek a DENSE_RANK használata esetén

A dense_rank függvény jól használható a következő probléma megoldásához. Adjuk vissza azokat a szolgáltatásokat, amelyeket azon a napon rögzítettek, amikor az utolsó rögzítés történt. Az általános megoldás a következőképpen néz ki.

```
SELECT ID
FROM service s1
WHERE TRUNC(s1.rec_time) = (SELECT MAX(TRUNC(s2.rec_time))
                           FROM service s2);
```

Ha megnézzük a végrehajtási tervet, jól látszik, ami a lekérdezésből várható, hogy a service tábla kétszer is végig lesz olvasva teljesen. Egyszer, mikor kiválasztjuk a maximumát a rögzítési időknek, majd még egyszer mikor kiválasztjuk a maximum alapján az aznapi rögzítéseket.

Nézzük hogyan oldható meg ez a probléma a DENSE_RANK használatával.

```
SELECT ID
FROM (SELECT ID,
                DENSE_RANK() OVER(ORDER BY TRUNC(s.rec_time) DESC NULLS LAST) rnk
FROM service s)
WHERE rnk = 1;
```

Ha most is megnézzük a végrehajtási tervet látjuk, hogy eltűnt az egyik TABLE ACCESS FULL sor.

Természetesen a *table access full* egy index elhelyezésével elkerülhető, azonban a hangsúly az egyszeri végrehajtáson van, hisz a lekérdezés nem mindig ilyen egyszerű, hiszen a service tábla helyett használhatnánk akár egy nézetet is ami 10 tábla összekapcsolásából áll elő. Ebben az esetben már nem mindegy hogy hányszor olvassuk végig a táblát.

A fentebbi példá kipróbáltam egy 3 470 680 sort tartalmazó táblán. A rec_time oszlopon nem volt index.

A futási idő az első megoldás esetén 10,657 sec.

A futási idő a második megoldás esetén 8,938 sec.

4.3 A FIRST_VALUE és a LAST_VALUE függvények

A first_value és a last_value függvények a képzett csoport meghatározott sorrendjének első illetve utolsó rekordjának megfogására szolgál.

Szintaxisa a következő:

FIRST_VALUE(<sql_kifejezés>) OVER ([PARTITION BY <...>] [ORDER BY <....>[<ablak_definíció>]])
LAST_VALUE(<sql_kifejezés>) OVER ([PARTITION BY <...>] [ORDER BY <....>[<ablak_definíció>]])

4.3.1 Példa a FIRST_VALUE használatára

Adott a következő probléma. Írassuk ki telephelyenként a dolgozók fizetésének a telephely legkisebb fizetésétől való eltérését.

```
SELECT   t.név "Név",
         t.telep "Telephely",
         t.fizetés "Fizetés",
         t.first_v "Legkisebb fiz.",
         t.fizetés - t.first_v "Eltérés"
FROM (
  SELECT dolg.név,
         dolg.telep,
         dolg.fizetés,
          first_value(dolg.fizetés) OVER(PARTITION BY dolg.telep ORDER BY dolg.fizetés) first_v
  FROM aa dolg )t;
```

Az eredményhalmaz a következőképpen néz ki:

	Név	Telephely	Fizetés	Legkisebb fiz.	Eltérés
1	Géza	Böszörmény	60000	60000	0
2	Éva	Böszörmény	80000	60000	20000
3	Tünde	Böszörmény	110000	60000	50000
4	Pisti	Debrecen	100000	100000	0
5	Béla	Debrecen	100000	100000	0
6	Laci	Debrecen	100000	100000	0
7	Attila	Debrecen	130000	100000	30000

4.4 A KEEP FIRST és a KEEP LAST kulcsszavak

Ezen két analitikus függvény elég speciális és valószínűleg használatuk sem lesz túl gyakori, azonban szükség esetén rengeteg fáradtságtól megkímélheti a programozót. A függvények szintaxisa is eltér az általános formától.

A szintaxis a következő:

Függvénynév() KEEP (DENSE_RANK FIRST ORDER BY <sql_kif>) OVER ([PARTITION BY <...>])
Függvénynév() KEEP (DENSE_RANK LAST ORDER BY <sql_kif>) OVER ([PARTITION BY <...>])

Látható, hogy az *order by* záradék kikerült az over() részből és átkerült a FIRST vagy a LAST kulcsszó után. Az over() rész csupán a *partition by* záradékot tartalmazza. A DENSE_RANK pedig jelen esetben egy kulcsszó nem a már előzőekben tárgyalt függvény, még sem véletlen a hasonlóság.

A függvény a következőképpen működik. A *partition by* záradék által meghatározott csoportot rangsorolja a dense_rank-nál leírt szabályok alapján, majd a rangsorolás szerinti első (FIRST esetén) vagy utolsó (LAST esetén) rangsorba eső rekordokon végrehajtja az aggregáló függvényt.

4.4.1 Példa a KEEP FIRST használatára

Adott a következő probléma. Írassuk ki minden dolgozóhoz a telephelyén dolgozó legalacsonyabb szintű dolgozók átlagfizetését.

```
SELECT   dolg.név,
         dolg.telep,
         dolg.szint,
         dolg.fizetés,
         AVG(dolg.fizetés) KEEP (DENSE_RANK FIRST ORDER BY dolg.szint)
                  OVER (PARTITION BY dolg.telep) Avg_fiz_szint
FROM aa dolg
ORDER BY dolg.telep, dolg.szint
```

A lekérdezés eredménye a következőképpen néz ki:

	NÉV	TELEP	SZINT	FIZETÉS	AVG_FIZ_SZINT
1	Géza	Böszörmény	1	60000	70000
2	Éva	Böszörmény	1	80000	70000
3	Tünde	Böszörmény	4	110000	70000
4	Laci	Debrecen	1	100000	100000
5	Béla	Debrecen	2	100000	100000
6	Pisti	Debrecen	3	100000	100000
7	Attila	Debrecen	5	130000	100000

4.5 A ratio_to_report függvény

A függvény szintaxisa a következő:

```
RATIO_TO_REPORT(<sql_kifejezés>) OVER ([PARTITION BY <...>])
```

A ratio_to_report függvény egy halmaz minden elemére megmondja, hogy azok hány százaléka a halmaz elemeinek összegének. A halmaz méretét a *partition by* záradékkal tudjuk szabályozni, a halmaz értékeit pedig paraméterül kapja a függvény. A függvény nem támogatja az *order by* záradékot és az ablak definíciót sem.

4.5.1 Példa a ratio_to_report függvény használatára

```
SELECT d.*,
         ratio_to_report(d.a2) over (partition by d.a1) as ratio_to_report
FROM test d
```

A lekérdezés eredménye a következő:

	A1	A2	RATIO_TO_REPORT
1	1	2	0,2
2	1	3	0,3
3	1	5	0,5
4	2	8	0,888888888888889
5	2	1	0,111111111111111
6	4	1	0,1
7	4	9	0,9

4.6 A NTILE függvény

A függvény szintaxisa a következő:

NTILE(kosarak száma) OVER ([PARTITION BY <...>] ORDER BY<...>)

Az NTILE függvény a rendezettséget alapul véve a partíció elemeit a paramétereként kapott számú kosárba osztja szét. Amennyiben a partíción belül a sorok száma nem többszöröse a kosarak számának, úgy mindig az alacsonyabb sorszámú kosarak kerülnek először feltöltésre.

4.6.1 Példa az NTILE függvény használatára

A példa a dolgozókat osztja szét telephelyenként 3 kosárba, a dolgozók nevének sorrendje alapján.

```
SELECT   dolg.név "Név",
         dolg.telep "Telephely",
         NTILE(3) OVER (PARTITION BY dolg.telep ORDER BY dolg.név) "Kosár"
FROM aa dolg
```

A lekérdezés eredménye a következő:

	Név	Telephely	Kosár
1	Éva	Böszörmény	1
2	Géza	Böszörmény	2
3	Tünde	Böszörmény	3
4	Attila	Debrecen	1
5	Béla	Debrecen	1
6	Laci	Debrecen	2
7	Pisti	Debrecen	3

Az eredményben látható, hogy a Debreceni partícióban az egyes kosárban került az osztást követően kimaradt egy elem.

4.7 A PARTITION BY záradék az aggregációs függvények után

Bármelyik csoportosító függvény után használhatjuk a *partition by* záradékot. A *partition by* segítségével a csoportosító függvényünket végrehajthatjuk a csoportosítás egy részcsoportján.

A *partition by* szintaxisa a következő:

{SUM | AVG | MAX | MIN | COUNT | ... } OVER ([PARTITION BY sql_kif1[,...]])

4.7.1 Példa a PARTITION BY záradék használatára

Adott a következő feladat. Határozzuk meg, hogy a beosztások össz fizetése mennyivel tér el az ugyan azon telephelyen található legmagasabb összfizetéssel rendelkező beosztástól.

```
SELECT t.beosztás "Beosztás",
        t.telep "Telephely",
        t.sfiz "Össz. fizu",
        t.msfiz "Tel. beosz. max fizu",
        t.msfiz - t.sfiz "Eltérés"
FROM (SELECT dolg.beosztás,
                dolg.telep,
                sum(dolg.fizetés) sfiz,
                MAX(SUM(dolg.fizetés)) OVER (PARTITION BY dolg.telep) msfiz
        FROM aa dolg
        GROUP BY dolg.beosztás, dolg.telep
        ORDER BY dolg.telep) t
```

A lekérdezés eredménye a következő:

	Beosztás	Telephely	Össz. fizu	Tel. beosz. max fizu	Eltérés
1	Kuli munkás	Böszörmény	140000	140000	0
2	Nagy főnök	Böszörmény	110000	140000	30000
3	Kis főnök	Debrecen	100000	130000	30000
4	Munkás	Debrecen	100000	130000	30000
5	Király	Debrecen	130000	130000	0
6	Kuli munkás	Debrecen	100000	130000	30000

4.8 Példa a ROW típusú ablakdefiníció használatára

Nézzünk néhány példát arra, hogy a mit kapunk eredményül az egyes esetekben, ha a telephelyet választjuk partíciónak és a fizetés a sorrend. Sajnos a példa nem túl életszerű, de a cél az egyszerűségen és a követhetőségen volt.

```
SELECT   dolg.név "Név",
          dolg.telep "Telephely",
         dolg.fizetés "Fizetés",
         COUNT(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
                  ROWS BETWEEN 3 PRECEDING
                  AND 1 FOLLOWING) "Előz 3 - Köv 1",
          COUNT(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
                  ROWS BETWEEN UNBOUNDED PRECEDING
                  AND CURRENT ROW) "Előz - Aktuális",
         COUNT(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
                  ROWS BETWEEN 3 PRECEDING
                  AND 1 PRECEDING) "Előz 2 - Előz 1",
          COUNT(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
```

```
                    ROWS BETWEEN 1 FOLLOWING
                    AND 3 FOLLOWING) "Előz 1 - Köv 3"
FROM aa dolg
ORDER BY dolg.telep, dolg.fizetés
```

A lekérdezés eredménye a következő:

	Név	Telephely	Fizetés	Előz 3 - Köv 1	Előz - Aktuális	Előz 2 - Előz 1	Előz 1 - Köv 3
1	Géza	Böszörmény	60000	2	1	0	2
2	Éva	Böszörmény	80000	3	2	1	1
3	Tünde	Böszörmény	110000	3	3	2	0
4	Pisti	Debrecen	100000	2	1	0	3
5	Béla	Debrecen	100000	3	2	1	2
6	Laci	Debrecen	100000	4	3	2	1
7	Attila	Debrecen	130000	4	4	3	0

4.9 Példa a RANGE típusú ablakdefiníció használatára

Ez a példa már egy kicsit életszerűbb. Határozzuk meg, hogy telephelyenként az egyes dolgozók esetén, hány olyan dolgozó van, ahol a dolgozó fizetésének ötödével többet vagy kevesebbet keresnek.

```
SELECT dolg.név "Név",
    dolg.telep "Telephely",
    dolg.fizetés "Fizetés",
    Count(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
        RANGE BETWEEN UNBOUNDED PRECEDING
        AND (dolg.fizetés/5) PRECEDING) "Kevesebb az ötödével",
    COUNT(*) OVER (PARTITION BY dolg.telep ORDER BY dolg.fizetés
        RANGE BETWEEN (dolg.fizetés/5) FOLLOWING
        AND UNBOUNDED FOLLOWING) "Több az ötödével"
FROM aa dolg
```

A lekérdezés eredménye a következő:

	Név	Telephely	Fizetés	Kevesebb az negyedével	Több az negyedével
1	Géza	Böszörmény	60000	0	2
2	Éva	Böszörmény	80000	1	1
3	Tünde	Böszörmény	110000	2	0
4	Pisti	Debrecen	100000	0	1
5	Béla	Debrecen	100000	0	1
6	Laci	Debrecen	100000	0	1
7	Attila	Debrecen	130000	3	0

5 EGYÉB HASZNOS FÜGGVÉNYEK

Az analitikus függvényeken túl az Oracle biztosít még több hasznos függvényt, melyek rangsorolnak vagy csoportosítási műveletekhez kapcsolódnak. Nézzünk ezek közül is egy kettőt.

5.1 A LISTAGG függvény

A függvény szintaxisi a következő:

LISTAGG(sql_kif,elválasztó_kar) WITHIN GROUP (ORDER BY<...>) [OVER (PARTITION BY <...>)]

A Listagg függvény ugyan nem egy hagyományos értelemben vett analitikus függvény, azonban használata nagyban hasonlít azokra és emellett sokszor nagyon hasznos.

Mint már fentebb említettem, alap esetben egy *group by* záradékkal rendelkező lekérdezés *select* ágán nem szerepelhet aggregációs függvény nélkül olyan oszlop, amely nem szerepel a *group by* záradékban. A listagg függvény tulajdonképpen ezt oldja fel úgy, hogy a paramétereként kapott oszlop csoportban szereplő értékeit egy elválasztó karaktersorozatot használva összefűzi.

Két paramétere van:

- sql kifejezés: ez egy olyan oszlopdefiníció, amely nem szerepel a *group by* záradékban.
- elválasztó karaktersorozat: ezzel a karaktersorozattal lesz elválasztva a csoport minden eleme az összefűzés során.

A függvényt a WITHIN GROUP kulcsszavak követik, majd zárójelben egy rendezési relációt kell megadni, mely szerint az oszlop értékei rendezve lesznek a felsoroláson belül.

5.1.1 Példa a LISTAGG függvény használatára

```
SELECT dolg.fizetés "Fizetés",
       listagg(dolg.név,', ') WITHIN GROUP (ORDER BY dolg.név) "Nevek"
FROM aa dolg
GROUP BY dolg.fizetés;
```

A lekérdezés eredménye a következő:

	Fizetés	Nevek
1	60000	Géza
2	80000	Éva
3	100000	Béla, Laci, Pisti
4	110000	Tünde
5	130000	Attila

5.2 A WIDTH_BUCKET függvény

A függvény szintaxisa a következő:

WIDTH_BUCKET(sql_kif, alsó_határ, felső_határ, zsákok_száma)

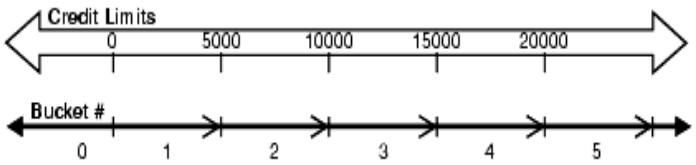
A WIDTH_BUCKET egy hisztogramm függvény, megy egy kiértékelt kifejezés értékeit szétosztja egy egyenlő részekre felosztott intervallumon.

Négy paramétere van:

- kifejezés: Ez a kifejezés adja az értéket melyet az intervallumon el kell helyezni.

- alsó határ: Az intervallum kezdete.
- felső határ: Az intervallum vége.
- zsákok száma: Hány részre osszuk fel az intervallumot.

Az első három paraméter lehet numerikus és dátum típusú. Más típus nem megengedett. Az utolsó paraméternek egy pozitív egész számnak kell lenni.Ezek után az alsó és felső határ közé eső intervallumot felosztjuk a zsákok számával. Ha az intervallumunk 1 - 20000 közé esik és a zsákok száma 4, akkor a felosztás a következőképpen fog kinézni.



Ez alapján lesz az első paraméterként megadott kifejezés elhelyezve valamelyik zsákba. Az intervallum [0,5000) halmazokra van felosztva. Az ábrán látszik, hogy van egy 0-ás és egy 5-ös zsák is. Értelemszerűen a 0-ás zsákba kerülnek azok az értékek, melyek kisebbek az intervallum alsó határánál és az 5-ös zsákba kerülnek azok az értékek, melyek nagyobbak az intervallum felső határánál.

5.2.1 Példa a WIDTH_BUCKET függvény használatára

Tegyük fel, hogy a dolgozóinkat szeretnénk beosztani 4 csoportba a fizetésük szerint és tudjuk, hogy a legalacsonyabb fizetés 60 000 Ft a legmagasabb pedig 130 000 Ft.

```
SELECT dolg.név "Név",
       dolg.fizetés "Fizetés",
       WIDTH_BUCKET(dolg.fizetés,60000,130000,4) "Csoport"
FROM aa dolg;
```

A lekérdezés eredménye a következő:

	Név	Fizetés	Csoport
4	Géza	60000	1
3	Éva	80000	2
6	Tünde	110000	3
7	Béla	100000	3
2	Pisti	100000	3
1	Laci	100000	3
5	Attila	130000	5

A példából látszik, hogy a balról zárt jobbról nyílt halmaz miatt Attila már felülcsordul és az 5-ös zsákba kerül.

6 KONKLÚZIÓ

Bár a példatábla kicsi volt, a könnyebb érthetőség kedvéért, azért érzékelhető, hogy bonyolultabb problémák is egyszerűen lekezelhetők az analitikus függvények segítségével. Ha csak a LAG vagy a LEAD függvényre gondolunk, a lekérdezés

aktuális sorában egy előző sor valamelyik értékére hivatkozni anélkül hogy valamilyen egyszerű szabállyal meg tudnánk határozni az előző sor elsődleges kulcsát, nem egy triviális probléma. Továbbá a végrehajtási tervet vizsgálva látható, hogy az oracle minimalizálja az ehhez szükséges erőforrásokat és egy végigolvasással határozza meg az értékeket.

Az oracle eszköztárában vannak még egyéb analitikus függvények (pl. lineáris regresszión alapuló függvények), melyek esetenként bonyolult problémák megoldását teszik lehetővé hatékony módon.

7 IRODALOMJEGYZÉK

1. Oracle Tuning - The Definitive Reference Second Edition (http://rampant-books.com/book_1002_oracle_tuning_definitive_reference_2nd_ed.htm)
2. http://download.oracle.com/docs/cd/B19306_01/server.102/b14200/functions001.htm#sthref964