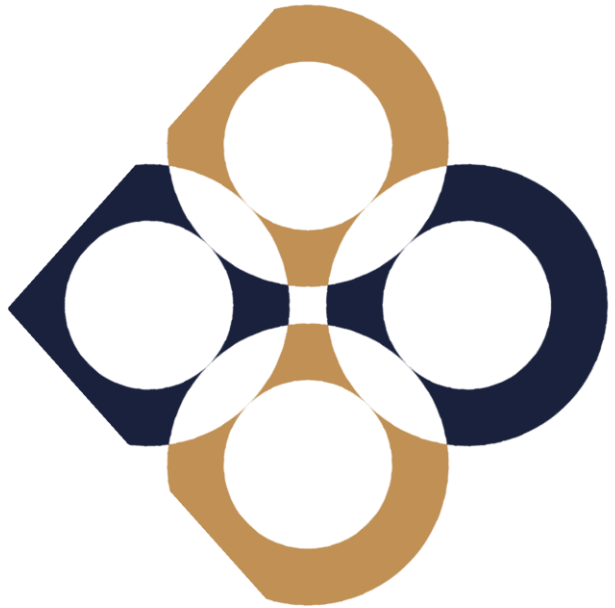
BUDAPESTI CORVINUS EGYETEM

Adatbázisok

Gyakorlat 04 –
A csoportosítás speciális lehetőségei



Részösszegek – GROUP BY ROLLUP

Az oszlopneveket és a NULL értéket kombinálva csoportosít, és megjeleníti a részösszegeket valamint végösszeget. Csoportosításkor a nem NULL oszlopok száma jobbról balra csökken

```
SELECT oszlopkifejezések*,
            aggregálás
FROM …
GROUP BY
ROLLUP(oszlopkifejezések*)
```

* Többnyire oszlopnevek listáját jelenti

Pl:

```
SELECT oszlop1, oszlop2, oszlop3,
            SUM(oszlop4)
FROM …
GROUP BY ROLLUP (oszlop1, oszlop2, oszlop3)
```

esetén a következő csoportok jönnek létre:

- Oszlop1, oszlop2, oszlop3
- Oszlop1, oszlop2, NULL
- Oszlop1, NULL, NULL
- NULL, NULL, NULL

Részösszegek – GROUP BY ROLLUP - Példa

RAKTAR_KOD	KAT_ID	MEGYS	Készlet értéke
5	5	db	294000
5	5	NULL	294000
5	NULL	NULL	294000
6	9	db	1262600
6	9	NULL	1262600
6	NULL	NULL	1262600
9	5	db	73000
9	5	NULL	73000
9	9	db	11800
9	9	NULL	11800
9	NULL	NULL	84800
NULL	NULL	NULL	1641400

Példa
Készítsünk listát a raktáron lévő termékek összértékéről raktárkód, azon belül kategóriakód, majd mennyiségi egység szerinti bontásban! A lista jelenítse meg a részösszegeket és a végösszeget is! A listát szűrjük az 5-ös és 9-es azonosítójú kategóriára! A csoportosításnál a ROLLUP záradékot használjuk!

```
SELECT RAKTAR_KOD, KAT_ID, MEGYS,
            SUM(LISTAAR*KESZLET)
             AS 'Készlet értéke' FROM
Termek
WHERE kat_ID IN (5,9)
GROUP BY ROLLUP (RAKTAR_KOD, KAT_ID, MEGYS)
```

Részösszegek – GROUP BY CUBE

Az oszlopneveket és a NULL értéket kombinálva csoportosít, és megjeleníti a részösszegeket valamint végösszeget. A csoportosításhoz minden lehetséges kombinációt felhasznál.

```
SELECT oszlopkifejezések,
            aggregálás
FROM …
GROUP BY
CUBE(oszlopkifejezések)
```

Pl:
SELECT oszlop1, oszlop2, oszlop3, SUM(oszlop4)
FROM …
GROUP BY CUBE (oszlop1, oszlop2, oszlop3) esetén a következő csoportok jönnek létre:

- Oszlop1, oszlop2, oszlop3
- Oszlop1, oszlop2, NULL
- Oszlop1, NULL, oszlop3
- Oszlop1, NULL, NULL
- NULL, oszlop2, oszlop3
- NULL, oszlop2, NULL
- NULL, NULL, oszlop3
- NULL, NULL, NULL

Részösszegek – GROUP BY CUBE- PÉLDA

RAKTAR_KOD	KAT_ID	MEGYS	Készlet értéke
5	5	db	294000
9	5	db	73000
NULL	5	db	367000
6	9	db	1262600
9	9	db	11800
NULL	9	db	1274400
NULL	NULL	db	1641400
NULL	NULL	NULL	1641400
5	NULL	db	294000
5	NULL	NULL	294000
6	NULL	db	1262600
6	NULL	NULL	1262600
9	NULL	db	84800
9	NULL	NULL	84800
5	5	NULL	294000
9	5	NULL	73000
NULL	5	NULL	367000
6	9	NULL	1262600
9	9	NULL	11800
NULL	9	NULL	1274400

Példa
Készítsünk listát a raktáron lévő termékek összértékéről raktárkód, azon belül kategóriakód, majd mennyiségi egység szerinti bontásban! A lista jelenítse meg a részösszegeket és a végösszeget is! A listát szűrjük az 5-ös és 9-es azonosítójú kategóriára! A csoportosításnál a CUBE záradékot használjuk!

```
SELECT RAKTAR_KOD, KAT_ID, MEGYS,
            SUM(LISTAAR*KESZLET)
             AS 'Készlet értéke'
FROM Termek
WHERE kat_ID IN (5,9)
GROUP BY CUBE (RAKTAR_KOD, KAT_ID, MEGYS)
```

GROUPING SETS

A GROUP BY parancs kiegészítve a GROUPING SETS taggal lehetővé teszi, hogy többféle csoportosítást is megadjunk.

A csoportosításokat leíró oszlopkifejezéseket egymás után, zárójelek között , vesszővel elválasztva kell megadni*

Pl:

```
SELECT oszlop1, oszlop2, SUM(oszlop3)
FROM table
GROUP BY
GROUPING SETS(( oszlop1, oszlop2), (oszlop1)
)
```

* A GROUPING SETS-ek az egyes csoportokra kiadott SELECT-ek UNION ALL-jainak alternatívái

GROUPING SETS - PÉLDA

szulev	nem	Átlagos életkor
NULL	F	35
NULL	N	49
1967	NULL	53
1975	NULL	45
1976	NULL	44
1980	NULL	40
1985	NULL	35
1997	NULL	23

Példa
Készítsünk listát az egyes ügyfelek átlagos életkoráról az ügyfél neme, illetve az ügyfél születési éve szerint csoportosítva! A listát szűrjük azon ügyfelekre, akik neve D-vel vagy E-vel kezdődik!
(Az életkor legyen a születési évtől a jelenlegi évig eltelt évek száma)

```
SELECT szulev, nem,
            AVG(YEAR(GETDATE())-SZULEV)
             AS 'Átlagos életkor'
FROM Ugyfel
WHERE NEV LIKE 'E%' OR NEV LIKE 'D%' GROUP BY
GROUPING SETS((SZULEV),(NEM))
```

A GROUPING és GROUPING _ID függvények

A GROUPING fv. értéke 1, ha az adott oszlopkifejezés szerint aggregálás (sum, min, max stb.) van, különben 0

A GROUPING_ID fv. értéke a paraméterként megkapott oszlopkifejezések aggregációs szintjének száma*

- Használhatók a SELECT, a HAVING és az ORDER BY részekben, amennyiben a GROUP BY is specifikálva van
- Mindkét fv. legfontosabb alkalmazása, hogy megkülönböztessük a ROLLUP, CUBE és GROUPING SETS-ek által visszaadott NULL értékeket (részösszegek, végösszegek) a normál NULL értékektől
- Ha egy oszlop(kifejezés) szerint csoportosítunk, akkor a két fv. ugyanazt az értéket adja vissza

* A szintszám számítása úgy történik, hogy az oszloplistához egy bináris kódot rendelünk. Ennek i-ik eleme 1, ha az i-ik oszlop szerint van aggregálás, különben 0. A szintszám a bináris szám értéke lesz tízes számrendszerben. Pl: ha a csoport (oszlop1, oszlop2), és mindkettő szerint van aggregálás, akkor a szintszám binárisan 11, decimálisan 3.

GROUPING & Részösszeg, végösszeg

Pl: listázzuk, hogy melyik évben hány db terméket rendeltek meg! A lista megfelelően jelölve jelenítse meg a rendelések teljes összegét is!

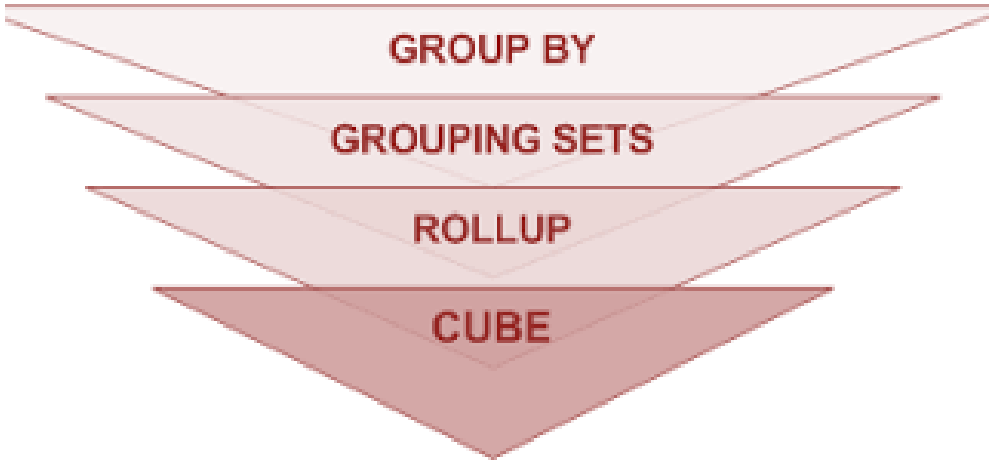
```
SELECT
  (
    CASE GROUPING(YEAR(REND_DATUM))
    WHEN 0 THEN CAST(YEAR(REND_DATUM)
                    AS nvarchar(4))
    WHEN 1 THEN 'Összesen' END
    )
    AS ÉV,
    COUNT(*) AS 'DB'
FROM Rendeles
GROUP BY ROLLUP(YEAR(REND_DATUM))
```

Pl: listázzuk, hogy naponta, azon belül fizetési mód szerint hány rendelés történt!
A lista megfelelően jelölve jelenítse meg a részösszegeket és a végösszeget is!

```
SELECT      IIF(GROUPING(REND_DATUM)=1,'Összesen',
            CAST(REND_DATUM AS nvarchar(10)))
                    AS 'Rendelés dátuma',
  (
    CASE GROUPING_ID(REND_DATUM, FIZ_MOD)
      WHE   0  THEN  FIZ_MOD
        N
      WHE   1  THEN  '**Fizetési módok összesen**'
        N
    END)      as 'FIZ_MOD',
      WHE   3  THEN  'Összesen'
      COUNT(*) as 'DB'
        N
FROM Rendeles
GROUP BY ROLLUP(REND_DATUM, FIZ_MOD)
```

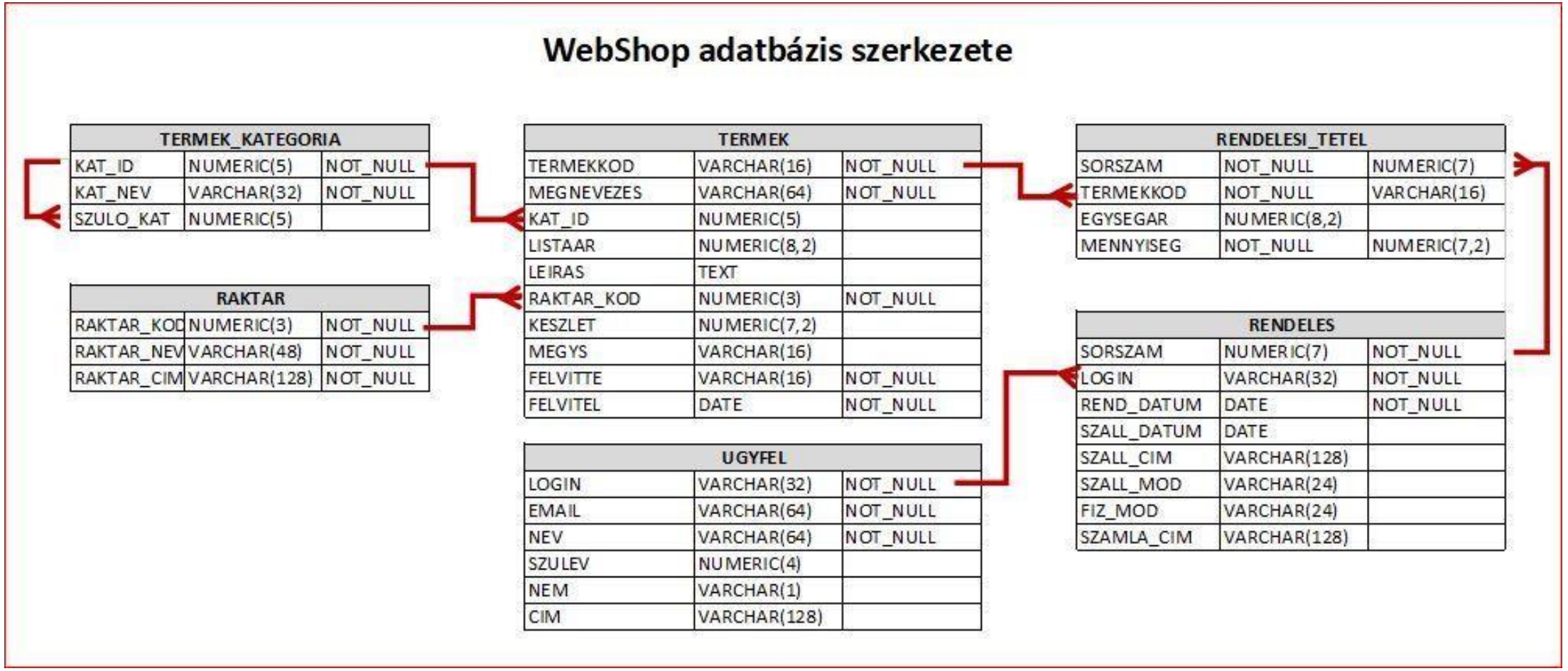
MEGJEGYZÉS

A GROUPING SETS, GROUPING / GROUPING_ID fv.
és a ROLLUP / CUBE együtt is használhatók




```
SELECT CASE
            WHEN GROUPING(szulev)=0
            THEN CAST(SZULEV AS nvarchar(4))
            ELSE 'Minden év'
          END AS 'Születési év', CASE
            WHEN GROUPING(nem)=1
            THEN 'Minden nem'
            ELSE NEM
          END AS 'Nem',
          AVG(YEAR(GETDATE())-SZULEV)
    AS 'Átlagos életkor'
FROM Ugyfel
WHERE NEV LIKE 'E%' OR NEV LIKE 'D%' GROUP BY
GROUPING SETS(ROLLUP(SZULEV),
CUBE(nem))
```

A gyakorlaton használt webshop adatbázis

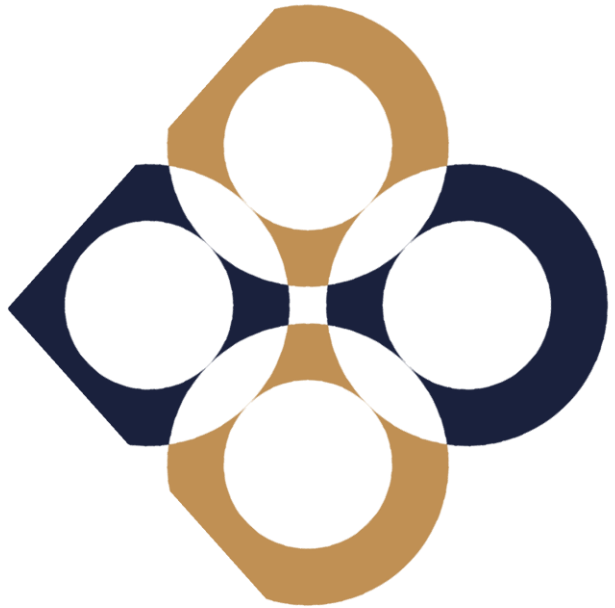


Köszönöm a figyelmet!

BUDAPESTI CORVINUS EGYETEM

Adatbázisok gyakorlat 05

Partíciók. Ablakok. Analitikus függvények



Partíciók

Azon rekordok csoportja, amelyeken az aggregálást el kell végezni

A GROUP BY alternatívái
Formája*:

OVER(
PARTITION BY kifejezés
)

Példa:
Jelenítsük meg a termékek kódja és listaára mellett a termékkategória átlagárát is!

```
SELECT TERMEKKOD, LISTAAR,
        AVG(LISTAAR) OVER(PARTITION BY KAT_ID)
        AS 'Kategória átlagár' FROM
Termek
```

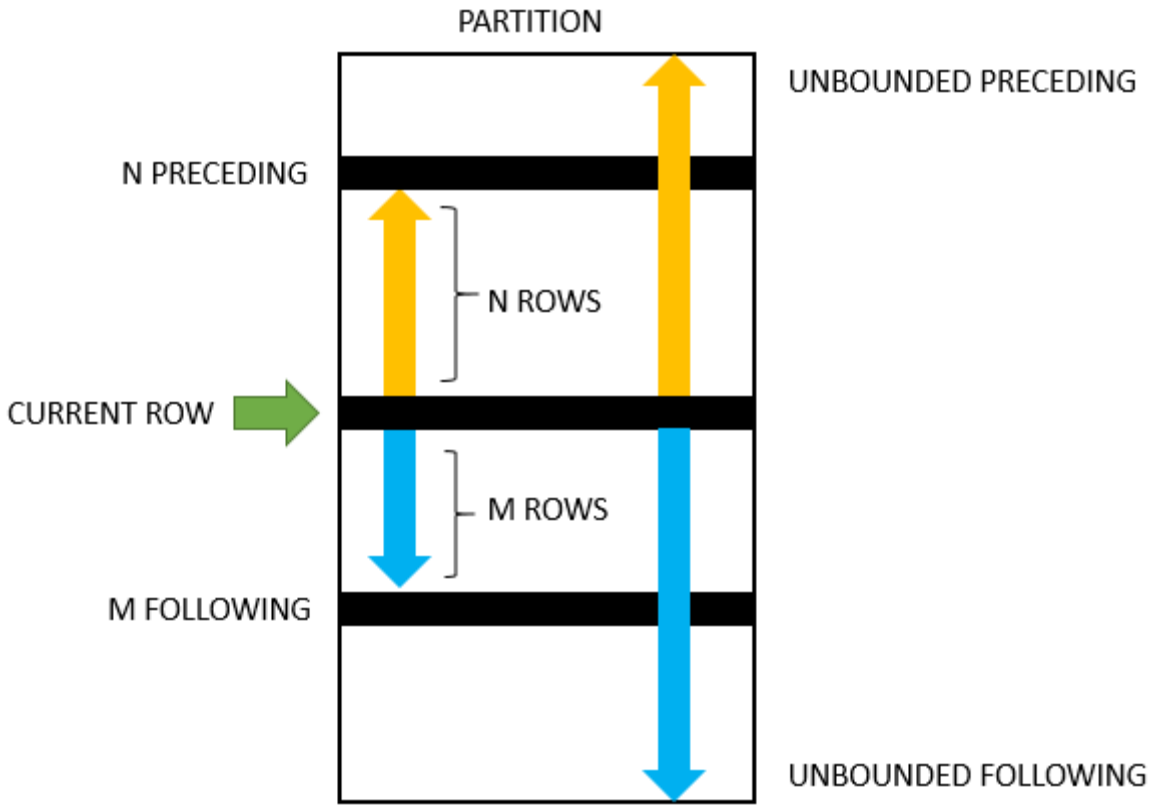
*A partíció kiegészíthető rendezéssel is (lásd következő diák):
OVER (PARTITION BY kifejezés ORDER BY kifejezés)

Ablakok létrehozása A ROWS és RANGE segítségével

ROWS, RANGE: az ablakot (partíció elejét és végét) határozzák meg
Használatukhoz az ORDER BY rész kötelező!

Formája:

OVER(PARTITION BY kifejezés*
ORDER BY kifejezés*
ROWS | RANGE BETWEEN
kezdőpont AND végpont)



* A kifejezés - itt és az összes többi utasítás/függvény leírásban - a gyakorlatban többnyire oszlopnevet vagy oszlopnevek listáját jelenti

ROWS

A ROWS az ablak méretét **fizikailag** adja meg
(legtöbbször az aktuális sort megelőző és/vagy követő sorok számát konkrétan megadja)
Kezdőpont, végpont lehet: CURRENT ROW, n PRECEDING, n FOLLOWING.
Speciálisan: UNBOUNDED PRECEDING (kezdőpont), UNBOUNDED FOLLOWING (végpont)*

Formája:
OVER(
PARTITION BY kifejezés
ORDER BY kifejezés
ROWS BETWEEN kezdőpont
AND végpont
)

* A partíció legelső, illetve legutolsó sorát jelentik meg

Példa:
Listázzuk az egyes megrendelések dátumát, a termék kódját és mennyiségét, valamint a sorszám szerinti előző 5 megrendelés átlagos mennyiségét is!

```
SELECT rt.TERMEKKOD, r.REND_DATUM, rt.MENNYISEG,
        AVG(rt.MENNYISEG) OVER(PARTITION BY
        rt.TERMEKKOD ORDER BY r.SORSZAM ROWS BETWEEN
        5 PRECEDING AND 1 PRECEDING)
         AS 'ELőző 5 rendelés mennyiség átlaga' FROM
Rendeles_tetel rt
    JOIN Rendeles r ON r.SORSZAM = rt.SORSZAM
```

RANGE

A RANGE az ablak méretét **logikailag** adja meg
(nem a sorok számát adja meg, hanem a legelső, legutolsó vagy az aktuális sort, mint az intervallum kezdő- vagy végpontját)
Kezdőpont, végpont lehet: CURRENT ROW, UNBOUNDED PRECEEDING (kezdőpont) és UNBOUNDED FOLLOWING (végpont)

Formája:

OVER(
PARTITION BY kifejezés
ORDER BY kifejezés
RANGE BETWEEN kezdőpont
AND végpont
)

Példa:
Jelenítsük meg, hogy az egyes ügyfelek az adott rendelési dátumig bezárólag összesen hányszor rendeltek!
Megjelenítendő a rendelés dátuma, az ügyfél login-ja és a rendelés darabszáma

```
SELECT DISTINCT REND_DATUM,[LOGIN], COUNT(*)
        OVER(PARTITION BY [LOGIN] ORDER BY
        REND_DATUM RANGE BETWEEN UNBOUNDED
        PRECEDING AND CURRENT ROW)
        AS 'Eddigi rendeléseinek száma' FROM
Rendeles
ORDER BY REND_DATUM, [LOGIN]
```

ROW_NUMBER()

A lekérdezés eredménysoraihoz sorszámokat rendel.

Formája:

ROW_NUMBER()
OVER (
PARTITION BY kifejezés
ORDER BY kifejezés)

Példa:
Készítsünk sorszámozott listát nemenként az ügyfelekről! A sorszámozás szempontja az ügyfél email-címe legyen!

```
SELECT ROW_NUMBER() OVER(PARTITION BY nem
            ORDER BY email)
            AS 'Nemenkénti sorszám', * FROM
Ugyfel
```

A ROW_NUMBER() mindig **szigorúan monoton növekvő** számokat ad vissza!
Több partíció esetén a sorszámozás minden partíciónál újra kezdődik.

RANK()

Megadja, hogy az adott rekord hányadik a partícióban az adott rendezettség szerint.*

Formája:

RANK()
OVER (
PARTITION BY kifejezés
ORDER BY kifejezés)

Példa:
Listázzuk a termékek kódját, megnevezését, kategória kódját, készlet mennyiségét és azt, hogy a termék a készlet alapján hányadik a kategóriájában

```
SELECT TERMEKKOD, MEGNEVEZES, KAT_ID, KESZLET, RANK()
OVER (PARTITION BY KAT_ID
ORDER BY KESZLET DESC)
AS 'Készlet szerinti helyezés kategóriájában' FROM Termek
```

* A RANK() mindig **monoton növekvő számokat** ad vissza!
- Az azonos értékű sorok ugyanazt a sorszámot kapják.
- A következő sorszám az aktuálisnál annyival lesz nagyobb, ahány azonos értékű sor van.

DENSE_RANK()

Megadja, hogy az adott rekord hányadik a partícióban az adott rendezettség szerint.*

Formája:

ROW_NUMBER()
OVER (
PARTITION BY kifejezés
ORDER BY kifejezés)

Példa
Az előző példa DENSE_RANK() függvénnyel

```
SELECT TERMEKKOD, MEGNEVEZES, KAT_ID, KESZLET,
        DENSE_RANK() OVER (PARTITION BY KAT_ID
        ORDER BY KESZLET DESC)
    AS 'Készlet szerinti helyezés kategóriájában' FROM Termek
```

* A DENSE_RANK() **mindig monoton növekvő** számokat ad vissza!

- Az azonos értékű sorok ugyanazt a sorszámot kapják.
- A következő sorszám az aktuálisnál eggyel nagyobb lesz

LAG()

Megadja egy adott sorhoz képest x-sorral korábbi oszlop értékét partíciónként egy adott rendezési szempont szerint

Formája:

LAG(kifejezés, x, default érték)
OVER (PARTITION BY kifejezés
ORDER BY kifejezés)

Példa:
Listázzuk minden rendelési tétel sorszámát, a termék kódját és mennyiségét, valamint az adott termék előző rendelésének mennyiségét!

```
SELECT SORSZAM, TERMEKKOD, MENNYISEG,
        LAG(MENNYISEG,1,0) OVER(PARTITION BY
            TERMEKKOD ORDER BY SORSZAM)
            AS 'Előző rendelési mennyiség'
  FROM Rendeles_tetel
```

A default érték akkor jelenik meg, ha nincs x sorral korábbi elem
Ha x és default érték elmarad, akkor 1 sorral ugrik vissza

LEAD()

Megadja egy adott sorhoz képest x-sorral későbbi oszlop értékét partíciónként egy adott rendezési szempont szerint

Formája:

LEAD(kifejezés, x, default)
OVER (PARTITION BY kifejezés
ORDER BY kifejezés)

Példa:
Listázzuk minden rendelési tétel sorszámát, a termék kódját és mennyiségét, valamint az adott termék kettővel későbbi rendelésének mennyiségét!

```
SELECT SORSZAM, TERMEKKOD, MENNYISEG,
        LEAD(MENNYISEG,2,0) OVER(PARTITION BY
        TERMEKKOD ORDER BY SORSZAM)
    AS 'Két rendeléssel későbbi rendelési mennyiség' FROM
Rendeles_tetel
```

Ha x és default érték elmarad, akkor 1 sort lép előre

FIRST_VALUE()

Megadja egy adott sorrendben lévő csoport (partíció) legelső elemét.

Formája:

FIRST_VALUE(kifejezés)
OVER (ORDER BY kifejezés
PARTITION BY kifejezés)

Példa:
Listázzuk az egyes ügyfelek adatait és első rendelésük dátumát! A lista ne tartalmazzon duplikált sorokat!

```
SELECT DISTINCT u.*,
        FIRST_VALUE(r.REND_DATUM) OVER (Partition BY
        u.LOGIN ORDER BY r.REND_DATUM)
        AS 'Első rendelés' FROM
Ugyfel u
    JOIN Rendeles r ON u.LOGIN = r.LOGIN
```

LAST_VALUE()

Megadja egy adott sorrendben lévő csoport(partíció) legutolsó elemét.*

Formája:

LAST_VALUE(kifejezés)
OVER (ORDER BY kifejezés
PARTITION BY kifejezés)

* A LAST_VALUE esetén vigyázni kell, mivel futtatáskor a partíció legutolsó eleme alapértelmezés szerint az aktuális sor! Megoldás lehet a RANGE vagy helyette fordított sorrend és FIRST_VALUE()

Példa:
Listázzuk az ügyfelek adatai és azt, hogy melyik ügyfél utoljára milyen módon legelőször, illetve legutoljára! A lista ne tartalmazzon duplikált sorokat!

```
SELECT DISTINCT u.*, FIRST_VALUE(r.FIZ_MOD)
            OVER (Partition BY u.LOGIN ORDER BY r.SORSZAM)
             AS 'Fizetési mód legelső rendeléskor', LAST_VALUE(r.FIZ_MOD)
             OVER (Partition BY u.LOGIN ORDER BY r.SORSZAM RANGE
             BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
             FOLLOWING)
             AS 'Fizetési mód legutolsó rendeléskor' FROM
Ugyfel u
JOIN Rendeles r ON u.LOGIN = r.LOGIN
```

NTILE()

A partíció elemeit adott számú osztályba sorolja a megadott sorrend alapján

Formája:

NTILE(osztályok száma)
OVER (ORDER BY kifejezés
PARTITION BY kifejezés)

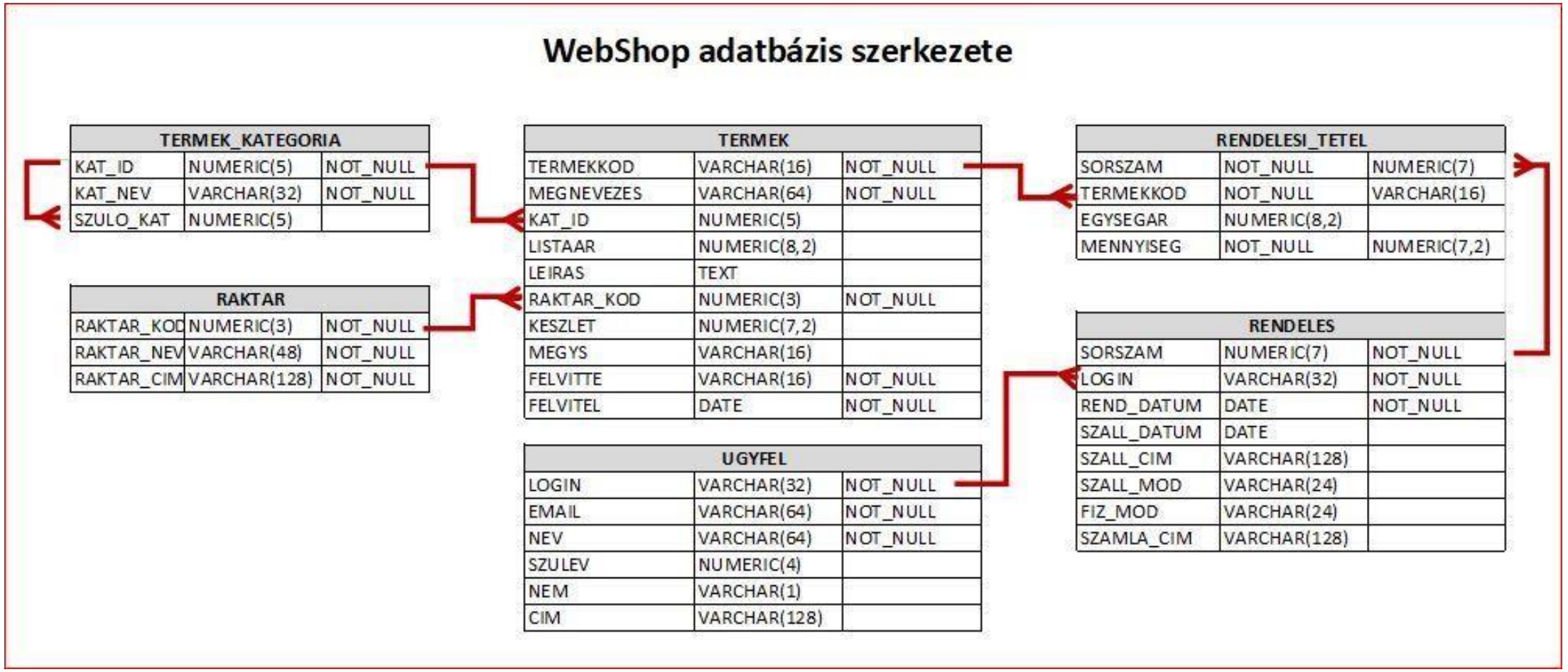
Példa:
Soroljuk be a termékeket kategóriájukban a listaáruk alapján 5 osztályba!

```
SELECT *,
        NTILE(5) OVER(PARTITION BY KAT_ID
        ORDER BY LISTAAR)
        AS 'Osztály'
FROM Termek
```

Megjegyzések

- Az analitikus függvények segítségével sok feladat egyszerűbben megoldható, mint „hagyományos” módon, viszont ilyenkor a lekérdezés többnyire lassúbb lesz
- Bizonyos feladatok a RANGE és a ROWS segítségével is megoldhatók, viszont duplikált sorok esetén a RANGE és a ROWS különböző eredményt adhat
- Egy lekérdezésben több ablak-függvény is szerepelhet
- A ROWS/RANGE esetén a végpont elhagyható, ez esetben alapértelmezés szerint a CURRENT ROW lesz
- Ha a PARTITION BY kimarad, akkor csak egy csoport lesz, amely minden rekordot tartalmaz

A gyakorlaton használt webshop adatbázis

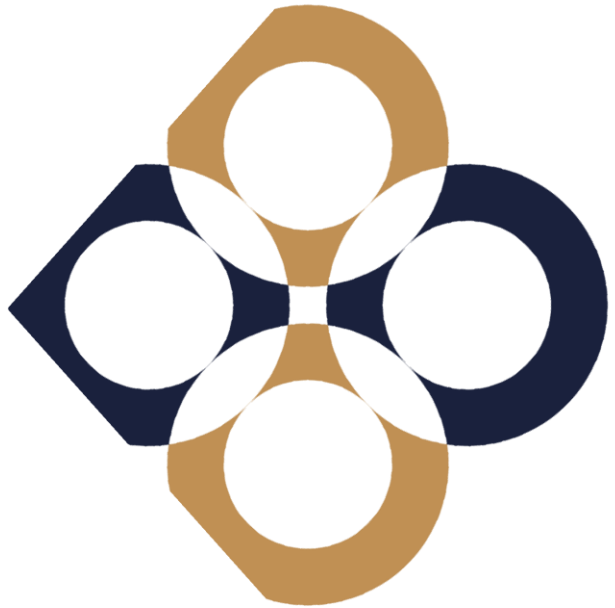


Köszönöm a figyelmet!

BUDAPESTI CORVINUS EGYETEM

Adatbázisok

Gyakorlat 06 –
Beágyazott lekérdezések



Beágyazott lekérdezés (Subquery, al-lekérdezés, alkérdés)

Lekérdezés a lekérdezésen belül

Rendszerint egy (külső) SELECT utasításon belüli (belső) SELECT utasítást jelent*

Először a belső SELECT fut le, majd annak eredményét megkapva a külső SELECT hajtódik végre**

Egy külső SELECT-be több belső SELECT is beágyazható

*Lekérdezést beágyazhatunk az INSERT, DELETE és UPDATE utasításokba is, de ezekkel most nem foglalkozunk

** A korrelált alkérdéseknél ez soronként történik

Beágyazott lekérdezés tipikus formája + példa

```
SELECT    select_list
FROM      table
WHERE     expr operator
                (SELECT    select_list
                 FROM      table);
```

```
SELECT ProductID,
       Name,
       ListPrice
FROM   production.Product
WHERE  ListPrice > (SELECT AVG(ListPrice)
                    FROM   Production.Product)
```

subquery

Beágyazott lekérdezések csoportosítása

Milyen eredményt ad vissza a beágyazott lekérdezés?

- Egy értéket (scalar)
- Több értéket (multi-valued)
- Táblát (table-valued)*

Hivatkozik-e a belső SELECT a külső SELECT valamely oszlopára?

- Ha igen, akkor korrelált alkérdésről beszélünk (correlated subquery)
- Ha nem, akkor önálló alkérdezésről beszélünk (self-contained subquery)

* Ezzel nem foglalkozunk

Hová kerülhet a beágyazott lekérdezés?

- A SELECT részbe – SELECT oszlop1, oszlop2, (subquery), oszlop4
- A FROM részbe – SELECT s.oszloplista FROM (subuqery) s
- **A WHERE részbe*** - lásd 3. dia
- A **HAVING részbe** - SELECT … HAVING kifejezés operátor (subquery)

* Ez a leggyakoribb eset, ezért főleg ezzel foglalkozunk. Kisebb súllyal, de a HAVING-es alkérdés is szerepelni fog a példák és feladatok között

Milyen tipikus esetekben használhatunk beágyazott lekérdezést?

- Ha szeretnénk összehasonlítani egy kifejezés értékét a beágyazott lekérdezés eredményével (legtöbbször <, >, =)
- Ha szeretnénk eldönteni, hogy egy kifejezés eredménye benne van-e a beágyazott lekérdezés eredményhalmazában (IN)
- Ha szeretnénk eldönteni, hogy a beágyazott lekérdezés eredményhalmaza üres-e (EXISTS)

Önálló alkérdés - összehasonlítás

Melyek azok a rendelési tételek, amelyek rendelési mennyisége az átlagos rendelési mennyiségnél nagyobb?

```
SELECT *
FROM rendeles_tetel
WHERE mennyiseg >
(
 SELECT AVG(mennyiseg)
 FROM rendeles_tetel
)
```

Önálló alkérdés – összehasonlítás + ANY, ALL*

Az ANY operátor igaz értéket ad vissza, ha az összehasonlítás eredménye az alkérdés legalább egy eredménysorára teljesül
Az ALL operátor igaz értéket ad vissza, ha az összehasonlítás eredménye az alkérdés minden eredménysorára teljesül

Példa:
Melyek azok a termékek, amelyek nem a legolcsóbbak (listaáruk nem a legkisebb)

```
SELECT megnevezes
FROM Termek
WHERE listaar > ANY
(
  SELECT listaar
  FROM Termek
)
```

*A SOME operátorral nem foglalkozunk

Önálló alkérdés - IN

Melyek azok az ügyfelek, akik már adtak le rendelést?

```
SELECT Nev
FROM Ugyfel
WHERE [login] IN
(
 SELECT DISTINCT [login]
 FROM rendeles
)
```

Korrelált alkérdés - Összehasonlítás

Melyek azok a termékek, amelyek listaára kategóriájukban a legmagasabb?

```
SELECT t.termekkod, t.MEGNEVEZES FROM
Termek t
WHERE t.LISTAAR = (
  SELECT max(t2.LISTAAR)
  FROM Termek t2
  WHERE t.KAT_ID = t2.KAT_ID
)
```

Korrelált alkérdés – Összehasonlítás + ANY, ALL

Melyek azok az a termékek, amelyek saját raktárukban a legolcsóbbak?

```
SELECT t.TERMEKKOD, t.megnevezes
FROM Termek t
WHERE t.listaar <= ALL (
  SELECT t2.listaar
  FROM Termek t2
  WHERE t.RAKTAR_KOD = t2.RAKTAR_KOD
)
```

Korrelált alkérdés - IN

Listázzuk azon ügyfeleket, akik rendeltek már ,’ Esküvői meghívó terméket!

```
SELECT u.NEV
FROM Ugyfel u
WHERE 'Esküvői meghívó' IN
(
    SELECT t.megnevezes
    FROM Rendeles r
            JOIN Rendeles_Tetel rt ON r.SORSZAM = rt.SORSZAM
            JOIN Termek t ON rt.TERMEKKOD = t.TERMEKKOD
    WHERE u.LOGIN = r.LOGIN
)
```

Korrelált alkérdés - EXISTS

Az EXISTS operátor igaz értéket ad vissza, ha a beágyazott SELECT eredményhalmaza nem üres

Példa:
Melyek azok a termékek, amelyekből legalább egyszer rendeltek már 50 darabnál többet?

```
SELECT t.megnevezes
from Termek t
where EXISTS (
    SELECT *
    FROM Rendeles_tetel rt
    WHERE rt.TERMEKKOD = t.TERMEKKOD
            AND rt.MENNYISEG > 50

)
```

Alkérdés - HAVING

Példa:
Melyek azok az ügyfelek, amelyek 2017-ben többször rendeltek, mint 2016-ban?
Elég az ügyfelek azonosítóját (LOGIN) megjeleníteni!

```
SELECT u.LOGIN
FROM Rendeles r JOIN Ugyfel u ON r.LOGIN = u.LOGIN WHERE
YEAR(rend_datum)=2017
GROUP BY u.login
HAVING COUNT(*)> (
    SELECT COUNT(*)
    FROM Rendeles r2 JOIN Ugyfel u2 ON r2.LOGIN = u2.LOGIN
    WHERE YEAR(rend_datum)=2016 AND u2.LOGIN = u.LOGIN
)
```

Beágyazott lekérdezés – fontosabb korlátozások

- Mindig zárójelbe kell tenni
- Összehasonlítás esetén mindig a reláció jobb oldalán áll
- Nem lehet benne ORDER BY*, INTO
- Ha van benne GROUP BY, akkor nem lehet benne DISTINCT
- Ha csak egy értéket ad vissza, akkor nem lehet benne GROUP BY és HAVING sem
- A visszaadott érték(ek)nek (join) kompatibilisnek kell lennie a külső SELECT WHERE feltételével
- Bizonyos adattípusok nem használhatók (ntext, text, image)

* Kivéve, ha TOP, FOR XML vagy OFFSET is szerepel az alkérdésben

Megjegyzések

- A beágyazott lekérdezések helyett többnyire más megoldást is használhatunk (pl: JOIN)
- A beágyazott lekérdezések átláthatóbbá teszik a kódot, viszont performacia szempontjából nem a legjobbak
- Ugyanaz a feladat sokszor többféle operátor használatával is megoldható (pl: IN, EXISTS).
- Nagyobb rekordszám esetén performancia szempontjából legtöbbször az EXISTS a legjobb választás
- Az IN és az EXISTS operátorok tagadhatók is (NOT IN, NOT EXISTS)
- A beágyazott lekérdezések egymásba is ágyazhatók

A gyakorlaton használt webshop adatbázis

Köszönöm a figyelmet!