Az SQL lekérdezőnyelv

- A legtöbb relációs ABKR az adatbázist az SQL-nek (Structured Query Language) nevezett lekérdezőnyelv segítségével kérdezi le és módosítja.
- Az SQL központi magja ekvivalens a relációs algebrával, de sok kiterjesztést dolgoztak ki hozzá, mint például az összesítések.
- Az SQL-nek számos verziója ismeretes, szabványokat is dolgoztak ki, ezek közül a legismertebb az SQL-92 vagy SQL2.
- Az SQL egy új szabványa az SQL3, mely rekurzióval, objektumokkal, triggerekkel stb. terjeszti ki az SQL2-őt. Számos kereskedelmi ABKR már meg is valósította az SQL3 néhány javaslatát.

Egyszerű lekérdezések SQL-ben

A relációs algebra vízszintes kiválasztás műveletét:

$$\sigma_f(R)$$

az SQL a SELECT, FROM és WHERE kulcsszavak segítségével valósítja meg a következőképpen:

```
SELECT *
FROM R
WHERE f;
```

példa: Legyen a NagyKer nevű adatbázis a következő relációsémákkal:

```
Részlegek (RészlegID, Név, Helység, ManSzemSzám);
Alkalmazottak (SzemSzám, Név, Fizetés, Cím,
                    RészlegID);
Managerek (SzemSzám);
ÁruCsoportok (CsopID, Név, RészlegID);
Áruk (ÁruID, Név, MértEgys, MennyRakt, CsopID);
Szállítók (SzállID, Név, Helység, UtcaSzám);
Vevők (VevőID, Név, Helység, UtcaSzám, Mérleg,
           Hihetőség);
Szállít (SzállID, ÁruID, Ár);
Szerződések (SzerződID, Dátum, Részletek, VevőID);
Tételek (TételID, Dátum, SzerződID);
Szerepel (TételID, ÁruID, RendMenny, SzállMenny).
```

„Keressük azon alkalmazottakat, akik a 9-es részlegnél dolgoznak és a fizetésük nagyobb, mint 500 euró”.

```
SELECT *
FROM Alkalmazottak
WHERE RészlegID = 9 AND Fizetés > 500;
```

- A FROM kulcsszó után adhatjuk meg azokat a relációkat, melyre a lekérdezés vonatkozik.
- A kiválasztás feltételét a WHERE kulcsszó után tudjuk megadni.
- A SELECT kulcsszó utáni * azt jelenti, hogy az eredmény reláció fogja tartalmazni a FROM után megadott reláció összes attribútumát.

- Az SQL nyelv nem különbözteti meg a kis és nagy betűket.
- Nem szükséges új sorba írni a FROM és WHERE kulcsszavakat, általában a fenti módon szokták megadni, de lehet egy sorban kis betűkkel is.

```
select * from alkalmazottak where részlegID =
9 and fizetés > 500;
```

A relációs algebra vetítés művelete

$$\pi_{A_{i_1}, A_{i_2}, \ldots, A_{i_k}}(R)$$

SELECT-SQL paranccsal:

```
SELECT A_i1, A_i2, ..., A_ik
FROM R;
```

A SELECT kulcsszó után megadhatjuk az *R* reláció bármely attribútumát és az eredmény sorok ezen attribútumokat fogják csak tartalmazni, ugyanazzal a névvel, amivel az *R* relációban szerepelnek.

példa: $\pi_{\text{Név, Fizetés}}(\text{Alkalmazottak})$

```
SELECT Név, Fizetés
FROM Alkalmazottak;
```

Lekérdezés feldogozás:

- a FROM kulcsszó után megadott relációt a feldolgozó végigjárja
- minden sor esetén ellenőrzi a WHERE kulcsszó után megadott feltételt azon sorokat, melyek esetén a feltétel teljesül, az eredmény relációba helyezzük.
- a feldogozás hatékonyságát növeli, ha a feltételben szereplő attribútumok szerint létezik indexállomány.

- a vetítés során kapott eredmény reláció esetén *megváltoztathatjuk az attribútumok neveit* az AS kulcsszó segítségével, ha a FROM után szereplő reláció attribútum nevei nem felelnek meg.
- az AS nem kötelező.
- a SELECT kulcsszó után kifejezést is használhatunk.

példa: Ha például a fizetést nem euró-ban, hanem $-ban szeretnénk és az euró/dollár arány mondjuk 1.3, akkor a nagy fizetésű alkalmazottakat a 9-es részlegből a következő paranccsal kapjuk meg:

```
SELECT Név AS Név9, Fizetés * 1.3 AS Fizetes$
   FROM Alkalmazottak
   WHERE RészlegID = 9 AND Fizetés > 500;
```

- A WHERE kulcsszó utáni feltétel lehet *összetett*, használhatjuk az AND, OR és NOT logikai műveleteket.
- A műveletek sorrendjének a meghatározására használhatunk zárójeleket, ha ezek megelőzési sorrendje nem felel meg.
- Az SQL nyelvben is, mint a legtöbb programozási nyelvben a NOT megelőzi az AND és OR műveletet, az AND pedig az OR-t.

példa: „Keressük a 3-as és 6-os részleg alkalmazottait akiknek kicsi a fizetése, 200 eurónal kisebb.”

```
SELECT Név, Fizetés
   FROM Alkalmazottak
   WHERE (RészlegID = 3 OR RészlegID = 6)
       AND Fizetés < 200;
```

Karakterláncok összehasonlítása esetén használhatjuk a LIKE kulcsszót, egy mintával való összehasonlításhoz:

```
k LIKE m
```

ahol k egy karakterlánc és m egy minta.

- A mintában % jelnek a k-ban megfelel bármilyen karakter 0 vagy nagyobb hosszúságú sorozata.
- A mintában _ jelnek megfelel egy akármilyen karakter a k-ból.
- A LIKE kulcsszó segítségével képezett feltétel igaz, ha a k karakterlánc megfelel az m mintának.

példa:

```
SELECT *
FROM Alkalmazottak
  WHERE Név LIKE 'Kovács%';
```

Használhatjuk a `k NOT LIKE m`

Más szűrőfeltételek:

- BETWEEN kulcsszó segítségével megadunk egy intervallumot, és azt vizsgáljuk, hogy adott oszlop, mely értéke esik a megadott intervallumba.

```
WHERE <oszlop> BETWEEN <kifejezés_1>
    AND <kifejezés_2 >
```

példa:

```
SELECT Név
FROM Alkalmazottak
WHERE Fizetés BETWEEN 300 AND 500;
```

Ugyanazt az eredményt adja, mint a:

```
SELECT Név
FROM Alkalmazottak
WHERE Fizetés >= 300 AND Fizetés <=500;
```

- Az IN operátor után megadunk egy értéklistát, és azt vizsgáljuk, hogy az adott oszlop mely mezőinek értéke egyezik az adott lista valamelyik elemével.

```
WHERE <oszlop> IN (<kifejezés_1>, <kifejezés_2>
            [,...])
```

„Keressük az ’531’-es, ’532’-s és ’631’-es csoportok diákjait:”

```
SELECT Név
FROM Diákok
WHERE CsopKod IN ('531', '532', '631');
```

példa: Legyen az Egyetem nevű adatbázis a következő relációsémákkal:

```
Szakok (SzakKod, SzakNév, Nyelv);
Csoportok (CsopKod, Evfolyam, SzakKod);
Diákok (BeiktatásiSzám, Név, SzemSzám, Cím,
     SzületésiDatum, CsopKod,Átlag);
TanszékCsoportok (TanszékCsopKod, Név);
Tanszékek (TanszékKod, Név, TanszékCsopKod);
Beosztások (BeosztásKod, Név);
Tanárok (TanárKod, Név, SzemSzám, Cím, PhD,
            TanszékKod, BeosztásKod,Fizetés);
Tantárgyak (TantKod, Név);
Tanít (TanárKod, TantKod);
Jegyek (BeiktatásiSzám, TantKod, Datum, Jegy)
```

- A SELECT SQL parancs lehetőséget ad az eredmény reláció rendezésére az ORDER BY kulcsszavak segítségével.
- Alapértelmezés szerint növekvő sorrendben történik a rendezés (ASC), csökkenő sorrend DESC kulcsszóval.

példa:
```
SELECT Név
FROM Diákok
WHERE CsopKod IN ('531', '532', '631')
ORDER BY CsopKod, Név;
```

példa: A diákokat átlag szerint csökkenő sorrendben adja meg:

```
SELECT Név
FROM Diákok
ORDER BY Átlag DESC;
```

Több relációra vonatkozó lekérdezések

- a relációs algebra egyik fontos tulajdonsága, hogy a műveletek eredménye szintén reláció, és az eredmény operandus lehet a következő műveletben.
- a SELECT-SQL is kihasználja ezt, a relációkat összekapcsolhatjuk, egyesíthetjük, metszetet vagy különbséget is számíthatunk.

A Descartes szorzat

$$R_1 \times R_2$$

műveletét a következő SQL parancs valósítja meg:

```
SELECT *
FROM R1, R2;
```

Théta-összekapcsolás: $R_1 \bowtie_\theta R_2$

```
SELECT *
FROM R1, R2
WHERE θ;
```

a természetes összekapcsolás

$$R_1 \bowtie R_2 = \pi_{B \cup C}(R_1 \bowtie_{(R_1.A_1=R_2.A_1)\wedge(R_1.A_2=R_2.A_2)\wedge\ldots\wedge(R_1.A_p=R_2.A_p)} R_2,$$

```
SELECT *
FROM R1, R2
WHERE R1.A1 = R2.A1 AND R1.A2 = R2.A2 AND … AND R1.Ap = R2.Ap;
```

példa: Legyenek a következő relációk:

```
Csoportok (CsopKod, Evfolyam, SzakKod);
Diákok (BeiktatásiSzám, Név, Cím,
          SzületésiDatum, CsopKod, Átlag);
```

a diákok esetén kiírjuk az évfolyamot és szakkódot is

```
SELECT Név, CsopKod, Evfolyam, SzakKod
FROM Diákok, Csoportok
WHERE Diákok.CsopKod = Csoportok.CsopKod;
```

Ha elfelejtjük a join feltételt az eredmény Descartes szorzat lesz, melynek méretei nagyon nagyok lehetnek.

- Vannak olyan ABKR-ek, melyek az előbbi feladat megoldására a JOIN kulcsszót is elfogadják (pl. MS SQL Server):

```
SELECT Név, CsopKod, Evfolyam, SzakKod
FROM Diákok INNER JOIN Csoportok
ON Diákok.CsopKod = Csoportok.CsopKod;
```

- Van OUTER JOIN is.

példa: „keressük az összes harmadéves diák nevét”

```
SELECT Név
FROM Diákok, Csoportok
WHERE Diákok.CsopKod = Csoportok.CsopKod
   AND Evfolyam = 3;
```

- Több mint két relációt is összekapcsolhatunk természetes összekapcsolással, fontos, hogy az összes join feltételt megadjuk.
- Ha az összekapcsolandó relációk száma k, és minden két-két relációnak egy-egy közös attribútuma van, akkor a join feltételek száma k–1.
- Ha 4 relációt kapcsolunk össze, a join feltételek száma minimum 3.

példa: „Adjuk meg azon szállítók nevét és címét, kik szállítanak édességet” (ÁruCsoportok.Név = ‘édesség’)

```
SELECT Szállítók.Név, Szállítók.Helység,
    Szállítók.UtcaSzám
FROM ÁruCsoportok, Áruk, Szállít, Szállítók
WHERE ÁruCsoportok.CsopID = Áruk.CsopID
   AND Áruk.ÁruID = Szállít.ÁruID
   AND Szállít.SzállID = Szállítók.SzállID
   AND ÁruCsoportok.Név = 'édesség';
```

- a FROM záradékban szereplő R relációhoz hozzárendelhetünk egy másodnevet, melyet *sorváltozó*nak nevezünk.
- sorváltozót akkor használunk, ha rövidebb vagy más nevet akarunk adni a relációnak, illetve ha a FROM után kétszer is ugyanaz a reláció szerepel.
- ha használtunk másodnevet, akkor kötelező használni.

példa: Keressük azon alkalmazottakat, akik ugyanazon a címen laknak, például férj és feleség, vagy szülő és gyerek.

```
SELECT Alk1.Név AS Név1, Alk2.Név AS Név2
FROM Alkalmazottak AS Alk1, Alkalmazottak AS Alk2
WHERE Alk1.Cím = Alk2.Cím
       AND Alk1.Név < Alk2.Név;
```

- A lekérdező feldolgozó ugyanazt a relációt kell kétszer bejárja, hogy a kért párokat megtalálja.
- Ha az Alk1.Név < Alk2.Név feltételt nem tettük volna, akkor minden alkalmazott bekerülne az eredménybe önmagával is párosítva.
- Ezt esetleg a <> feltétellel is megoldhattuk volna, de akkor egy férj–feleség páros kétszer is bekerült volna, csak más sorrendben. Például: (‘Kovács István’, ‘Kovács Sára’) és (‘Kovács Sára’, ‘Kovács István’) is.
- Mivel gyereknek lehet ugyanaz a neve, mint a szülőnek, ezért jobb megoldás a: Alk1.Név < Alk2.Név feltételt kicserélni:

```
Alk1.SzemSzám < Alk2. SzemSzám;
```

Algoritmus egy egyszerű SELECT−SQL lekérdezés kiértékelésére:
Input: R_1, R_2,…, R_n relációk a FROM záradék után
Begin
  Minden t_1 sorra az R_1-ből
    Minden t_2 sorra az R_2-ből
      …
      Minden t_n sorra az R_n-ből
        Ha a WHERE záradék igaz a t_1, t_2, …, t_n attribútumainak az értékeire
        Akkor
          A SELECT záradék attribútumainak értékeiből alkotott sort az eredményhez adjuk
End

- A relációs algebra *halmazműveleteit* (egyesítés, metszet és különbség) használhatjuk az SQL nyelvben, azzal a feltétellel, hogy az operandus relációknak ugyanaz legyen az attribútumhalmaza.
- Az SQL2 standard adja ezeket a lehetőségeket, de a kereskedelmi rendszerek esetén nem használhatjuk mindegyiket.

A megfelelő kulcsszavak:

- UNION az egyesítésnek,
- INTERSECT a metszetnek
- EXCEPT a különbségnek. (Oracle: MINUS)

példa: Legyenek a Szállítók és Vevők relációk a NagyKer adatbázisból és a következő lekérdezés: „Keressük a kolozsvári cégeket, akikkel kapcsolatban áll a cégünk.” A megoldást a következő lekérdezés adja:

```
(SELECT Név, UtcaSzám
  FROM Szállítók
  WHERE Helység = 'Kolozsvár')
   UNION
(SELECT Név, UtcaSzám
  FROM Vevők
  WHERE Helység = 'Kolozsvár');
```

példa: Legyenek az Alkalmazottak és Managerek relációk a NagyKer adatbázisból és a „Keressük azon alkalmazottakat, akik nem managerek” lekérdezés:

```
(SELECT SzemSzám, Név FROM Alkalmazottak)
   EXCEPT
(SELECT SzemSzám, Név FROM Managerek,
                           Alkalmazottak
  WHERE Managerek.SzemSzám =
              Alkalmazottak.SzemSzám);
```

Ha az alkalmazott névre nem vagyunk kíváncsiak:

```
(SELECT SzemSzám FROM Alkalmazottak)
   EXCEPT
(SELECT SzemSzám FROM Managerek);
```

A feladatot oly módon is megoldhatjuk, ha a kereskedelmi rendszer nem támogatja az EXCEPT műveletet, hogy alkalmazzuk a NOT EXISTS vagy NOT IN záradékot.

példa: Legyen az Egyetem adatbázisa, és tegyük fel, hogy van olyan eset, hogy egy fiatal tanársegéd a matematika szakról, tehát elvégezte a matematika szakot, de még diák az informatika szakon. Legyen a következő lekérdezés: „keressük azon tanárokat, akik még diákok”. A megoldás:

```
(SELECT Név FROM Tanárok)
    INTERSECT
(SELECT Név FROM Diákok);
```

A feladatot a következőképpen is megoldhatjuk, ha a kereskedelmi rendszer nem támogatja az INTERSECT műveletet:

```
SELECT Név FROM Tanárok
WHERE EXISTS
 (SELECT Név FROM Diákok
    WHERE Diákok.SzemSzám = Tanárok.SzemSzám);
```

Ismétlődő sorok

- Az SQL nyelv relációi az absztrakt módon definiált relációktól abban különböznek, hogy az SQL nem tekinti őket halmaznak, azaz a relációk multihalmazok.
- A SELECT parancs eredményében szerepelhet két vagy több teljesen azonos sor, viszont van lehetőség ezen ismétlődések megszüntetésére.
- A SELECT kulcsszó után a DISTINCT szó segítségével kérhetjük az azonos sorok megszüntetését.

példa: Egyetem adatbázisa esetén keressük azon csoportokat, amelyekben vannak olyan diákok, akiknek átlaga kisebb, mint 7.

```
SELECT DISTINCT CsopKod
FROM Diákok
WHERE Átlag < 7;
```

- A SELECT paranccsal ellentétben, a UNION, EXCEPT és INTERSECT halmazelméleti műveletek megszüntetik az ismétlődéseket.
- Ha nem szeretnénk, hogy az ismétlődő sorok eltűnjenek, a műveletet kifejező kulcsszó után az ALL kulcsszót kell használjuk.

példa: Az Egyetem adatbázisból keressük a személyeket, akik lehetnek tanárok vagy diákok. A következő parancs nem szünteti meg az ismétlődéseket:

```
(SELECT Név FROM Tanárok)
 UNION ALL
(SELECT Név FROM Diákok);
```

Összesítő függvények és csoportosítás

- az SQL nyelv lehetőséget ad egy oszlopban szereplő értékek összegezésére, meghatározzuk a legkisebb, legnagyobb vagy átlag értéket egy adott oszlopból.
- az *összesítés* művelete egy oszlop értékeiből *egy* új értéket hoz létre.
- a reláció egyes sorait bizonyos feltétel szerint csoportosíthatjuk, például egy oszlop értéke szerint, és a csoporton belül végezhetünk összesítéseket.

Összesítő függvények a következők:

- SUM, megadja az oszlop értékeinek az összegét;
- AVG, megadja az oszlop értékeinek a átlagértékét;
- MIN, megadja az oszlop értékeinek a minimumát;
- MAX, megadja az oszlop értékeinek a maximumát;
- COUNT, megadja az oszlopban szereplő értékek számát, beleértve az ismétlődéseket is, ha azok nincsenek megszüntetve a DISTINCT kulcsszóval.

Ezeket a függvényeket egy skalár értékre alkalmazhatjuk, általában egy SELECT záradékbeli oszlopra.

példa: az alkalmazottak átlag fizetése:

```
SELECT AVG(Fizetés)
FROM Alkalmazottak;
```

példa: az alkalmazottak száma:

```
SELECT COUNT(*)
FROM Alkalmazottak;
```

Mindkét példa esetén biztosak vagyunk abban, hogy egy alkalmazott csak egyszer szerepel a relációban, mivel a személyi szám elsődleges kulcs.

példa: Az Egyetem adatbázis esetén keressük azon csoportoknak a számát (hány darab van), amelyekben vannak olyan diákok, akik átlaga kisebb, mint 7:

```
SELECT COUNT(DISTINCT CsopKod)
FROM Diákok
WHERE Átlag < 7;
```

- Az eddigi összesítések az egész relációra vonatkoztak.
- Sok esetben a reláció sorait csoportosítani szeretnénk egy vagy több oszlop értékei szerint.
- A csoportosítást a GROUP BY kulcsszó segítségével érjük el. A parancs általános formája:

```
SELECT < csoportosító oszlopok listája >,
    <összesítő-függvény>(<oszlop>)
FROM   <reláció>
[WHERE <feltétel>]
[GROUP BY <csoportosító oszlopok listája>]
[HAVING <csoportosítási-feltétel>]
[ORDER BY <oszlop>];
```

- A GROUP BY után megadjuk a csoportosító attribútumok (oszlopok) listáját, melyek azonos értéke szerint történik a csoportosítás.
- Csak ezeket az oszlopokat válogathatjuk ki a SELECT kulcsszó után és azokat, melyekre valamilyen összesítő függvényt alkalmazunk.
- Azon oszlopoknak, melyekre összesítő függvényt alkalmaztunk, érdemes más nevet adni, hogy könnyebben tudjunk hivatkozni rá.

példa: Legyenek az Alkalmazottak reláció sorai:

SzemSzám	***Név***	***RészlegID***	***Fizetés***
111111	Nagy Éva	2	300
222222	Kiss Csaba	9	400
456777	Szabó János	9	900
234555	Szilágyi Pál	2	700
123444	Vincze Ildikó	1	800
333333	Kovács István	2	500

- részlegeken belüli átlagfizetés, minimális, maximális fizetés, illetve összfizetés:

```
SELECT RészlegID, AVG(Fizetés) AtlFiz,
   MIN(Fizetés) MinFiz, MAX(Fizetés) MaxFiz,
   SUM(Fizetés) OsszFiz
FROM Alkalmazottak
GROUP BY RészlegID;
```

A kapott eredmény:

RészlegID	***AtlagFiz***	***MinFiz***	***MaxFiz***	***OsszFiz***
1	800	800	800	800
2	500	300	700	1500
9	650	400	900	1300

- a lekérdezés processzor először rendezi a reláció sorait a csoportosítandó oszlop értékei szerint
- utána azokat a sorokat, ahol ezen oszlopoknak ugyanaz az értéke, az eredmény relációban csak egy sor fogja képviselni, ahol megadhatjuk az oszlop értékét, amely a lekérdezett relációban minden sorban ugyanaz.
- a többi oszlopra csakis összesítéseket végezhetünk
- a SELECT parancs megengedi, hogy a csoportosító attribútum hiányozzon a vetített attribútumok listájából.

példa: A következő lekérdezés helyes:

```
SELECT AVG(Fizetés) AS ÁtlagFizetés
FROM Alkalmazottak
GROUP BY RészlegID;
```

eredménye pedig:

ÁtlagFizetés
800
500
650

példa: Legyen a Szállít (SzállID, ÁruID, Ár) reláció. A következő lekérdezés minden áru esetén meghatározza az átlagárat, amiben a különböző szállítók ajánlják.

```
SELECT ÁruID, AVG(Ár) AS ÁtlagÁr
FROM Szállít
GROUP BY ÁruID;
```

- a GROUP BY záradékot használhatjuk többrelációs lekérdezésben is.
- a lekérdezés processzor először az operandus relációkkal a WHERE feltételét figyelembe véve elvégzi a join, esetleg a Descartes szorzat műveletet és ennek az eredmény relációjára alkalmazza a csoportosítást.

példa: Ha a fenti példa esetén kíváncsiak vagyunk az árunak a nevére:

```
SELECT Áruk.Név, AVG(Ár)
FROM Szállít. ÁruID = Áruk.ÁruID
WHERE Szállít, Áruk
GROUP BY Áruk.Név;
```

- Remélhetőleg az áru neve is egyedi kulcs.
- Megoldhatjuk úgy is, hogy először ÁruID szerint, majd áru név szerint csoportosítunk

- lehetséges több csoportosítási attribútum is.

példa: Legyenek a következő relációk az Egyetem adatbázisból:

```
Tanszékek (TanszékKod, Név, TanszékCsopKod);
Beosztások (BeosztásKod, Név);
Tanárok (TanárKod, Név, SzemSzám, Cím, PhD,
         BeosztásKod, TanszékKod,     Fizetés);
```

és a következő lekérdezés: „Számítsuk ki a tanárok átlagfizetését tanszékeken belül, beosztásokra leosztva.”

```
SELECT TanszékKod, BeosztásKod, AVG(Fizetés)
FROM Tanárok
GROUP BY TanszékKod, BeosztásKod
```

Tanár Kod	***Név***	***Cím***	***PhD***	***Beosztás Kod***	***Tanszék Kod***	***Fizetés***
KB12	Kiss Béla	Petőfi u. 12	Y	ADJ	ALG	150
NL03	Nagy László	Kossuth u. 3	Y	ADJ	REN	160
KG05	Kovács Géza	Ady tér 5	N	ADJ	ALG	160
PI14	Péter István	Dóm tér 14	N	TNS	REN	120
NE55	Németh Eva	Dózsa u. 55	Y	PRO	ALG	300
VS77	Vígh Sándor	Rózsa u. 77	Y	PRO	REN	310
LL63	Lukács Lóránt	Viola u. 63	Y	ADJ	REN	170
LS07	László Samu	Rákóczi u. 7	N	TNS	REN	110
KP52	Kerekes Péter	Váczi u. 52	Y	PRO	ALG	280

a lekérdezés eredménye:

Tanszék Kod	***Beosztás Kod***	***AVG (Fizetés)***
ALG	ADJ	155
ALG	PRO	290
REN	ADJ	165
REN	PRO	310
REN	TNS	115

példa: ismét: minden áru esetén átlagár!

```
SELECT Áruk. ÁruID, Áruk.Név, AVG(Ár)
FROM Szállít. ÁruID = Áruk.ÁruID
WHERE Szállít, Áruk
GROUP BY Áruk.ÁruID, Áruk.Név;
```

A vetítés attribútumai között nem kell feltétlenül szerepeljen az ÁruID, de ha egy név többször is előfordul, akkor az eredmény furcsa lesz. □

- A csoportosítás után kapott eredmény reláció soraira a HAVING kulcsszót használva egy feltételt alkalmazhatunk.
- Ha csoportosítás előtt szeretnénk kiszűrni sorokat, azokra a WHERE feltételt lehet alkalmazni.
- A HAVING kulcsszó utáni feltételben azon oszlopok szerepelhetnek, melyekre a SELECT parancsban összesítő függvényt alkalmaztunk.

példa: Keressük azon részlegeket, ahol az alkalmazottak átlagfizetése nagyobb, mint 500 euró, átlagfizetés szerint növekvő sorrendben.

```
SELECT RészlegID, AVG(Fizetés)
   FROM Alkalmazottak
   GROUP BY RészlegID
   HAVING AVG(Fizetés) > 500
   ORDER BY AVG(Fizetés);
```

A fenti adatokat figyelembe véve az eredmény reláció a következő lesz:

RészlegID	*AVG(Fizetés)*
9	650
1	800

Ha nem adjuk meg az ORDER BY záradékot, akkor a GROUP BY záradékban megadott oszlopok szerint rendezi az eredményt.

példa: *Helytelen* a következő parancs:

```
SELECT RészlegID, AVG(Fizetés)
   FROM Alkalmazottak
   WHERE AVG(Fizetés) > 500
   GROUP BY RészlegID;
```

példa: Keressük azon tanszékeket, ahol a tanársegédeket kivéve a tanárok átlagfizetése nagyobb, mint 240 euró.

```
SELECT TanszékKod, AVG(Fizetés)
   FROM Tanárok
   WHERE BeosztásKod <> 'TNS'
   GROUP BY TanszékKod
    HAVING AVG(Fizetés) > 240;
```

Az eredmény a következő lesz:

Tanszék Kod	***AVG (Fizetés)***
ALG	125

- Alkalmazhatunk összesítő függvényt a csoportosítás után.

példa: Keressük a tanárok tanszékenkénti átlagfizetéseiből a legnagyobb értéket:

```
 SELECT MAX(AVG(Fizetés))
FROM Tanárok
GROUP BY TanszékKod;
```

- A lekérdezés processzor először elvégzi a csoportosítást TanszékKod szerint, majd az átlagfizetésekből kiválasztja a legnagyobbat.

Alkérdések

- A WHERE záradékban eddig a feltételben skaláris értékeket tudtunk összehasonlítani.
- Az alkérdések segítségével sorokat vagy relációkat tudunk összehasonlítani.
- Egy alkérdés egy olyan kifejezés, mely egy relációt eredményez, például egy select-from-where kifejezés.

Alkérdést tartalmazó SELECT SQL parancs általános formája:

```
SELECT <attribútum_lista>
FROM   <tábla>
WHERE <kifejezés> <operátor>
    (SELECT <attribútum_lista>
                FROM <tábla>);
```

- A rendszer először az alkérdést hajtja végre és annak eredményét használja a „fő” lekérdezés, kivéve a korrelált alkérdéseket.

Alkérdéseket annak megfelelően csoportosíthatjuk, hogy az eredménye hány sort és hány oszlopot tartalmaz:

– egy oszlopot, egy sort, vagyis egy *skalár* értéket ad vissza (single-row);
– egy oszlopot, több sort, ún. *több soros alkérdés* (multiple-row subquery);
– több oszlopot, több sort, ún. *több oszlopos alkérdés* (multiple-column);

Alkérdés mely skalár értéket ad vissza:

- select-from-where kifejezésben egy attribútum egyetlen értéke
- ez konstansként használható
- egy attribútummal vagy egy másik konstanssal összehasonlíthatjuk.
- nagyon fontos, hogy az alkérdés select-from-where kifejezése csak egy attribútumnak egyetlen értékét adja eredményül, különben hibajelzést kapunk.

példa: „Keressük a 'Tervezés' nevű részleg managerének a nevét."

```
1)  SELECT Név
2)  FROM Alkalmazottak
3)  WHERE SzemSzám =
4)      (SELECT ManSzemSzám
5)       FROM Részlegek
6)       WHERE Név = 'Tervezés');
```

- biztosak kell legyünk, hogy csak egy ’Tervezés’ nevű részleg legyen az adatbázisban.
- ezt elérhetjük ha egyedi kulcs megszorítást kérünk a Részlegek relációra a CREATE TABLE parancsban a UNIQUE kulcsszó segítségével.
- abban az esetben, ha az alkérdés nulla vagy egynél több sort eredményez, a lekérdezés futás közbeni hibát fog jelezni.
- az alkérdés eredményül az 123444 személyi számot adja, és a lekérdezés a következőképpen hajtódik végre:

```
SELECT Név
FROM Alkalmazottak
WHERE SzemSzám = 123444
```

A lekérdezés eredménye: ‘Vincze Ildikó’ lesz.

A *skalár* értéket adó alkérdéssel használható *operátorok* az:

=, <, <=, >, >=, <>.

példa: „Keressük azon alkalmazottakat, kiknek fizetése nagyobb, mint annak az alkalmazottnak, kinek a személyi száma 333333.”

```
SELECT Név
FROM Alkalmazottak
WHERE Fizetés >
 (SELECT Fizetés
  FROM Alkalmazottak
  WHERE SzemSzám = 333333);
```

példa: „Keressük azon alkalmazottakat, kiknek a fizetése az összes alkalmazott minimális fizetésével egyenlő.”

```
SELECT Név
FROM Alkalmazottak
WHERE Fizetés =
    (SELECT MIN(Fizetés)
     FROM Alkalmazottak);
```

példa: „Keressük azon részlegeket és az alkalmazottak minimális fizetését a részlegből, ahol a minimális fizetés nagyobb, mint a minimális fizetés a 2-es ID-jű részlegből.”

```
SELECT RészlegID, MIN(Fizetés)
FROM   Alkalmazottak
 GROUP BY RészlegID
 HAVING MIN(Fizetés) >
    (SELECT MIN(Fizetés)
     FROM Alkalmazottak
     WHERE RészlegID = 2);
```

A lekérdezés processzor először az alkérdést értékeli ki, ennek eredményeként egy skalár értéket (300) kapunk és a fő lekérdezés ezzel a skalár értékkel fog dolgozni.

Csínján kell bánjunk a csoportosítással.

Példa: Egy *helytelen* SELECT parancs:

```
SELECT SzemSzám, Név
FROM Alkalmazottak
WHERE Fizetés =
 (SELECT MIN(Fizetés)
  FROM Alkalmazottak
     GROUP BY RészlegID);
```

A *több-soros alkérdések* esetén a WHERE záradék feltétele olyan *operátorok*at tartalmazhat, amelyeket egy *R* relációra alkalmazhatunk, ebben az esetben az eredmény logikai érték lesz. Bizonyos operátoroknak egy skaláris *s* értékre is szükségük van. Ilyen operátorok:

- `EXISTS` *R* – feltétel, mely akkor és csak akkor igaz, ha *R* nem üres.

példa:

```
SELECT Név
FROM Alkalmazottak, Managerek
WHERE Alkalmazottak. SzemSzám =
                    Managerek.SzemSzám
 AND EXISTS
   (SELECT *
       FROM Alkalmazottak
       WHERE Fizetés > 500);
```

- *s* `IN` *R*, mely akkor igaz, ha *s* egyenlő valamelyik *R*-beli értékkel. Az *s* NOT IN *R* akkor igaz, ha *s* egyetlen *R*-beli értékkel sem egyenlő.

példa: „Adjuk meg azon szállítók nevét és címét, akik valamilyen csokit szállítanak” (Áruk.Név LIKE ‘%csoki%’)

```
1)  SELECT Név, Helység, UtcaSzám
2)  FROM Szállítók
3)  WHERE SzállID IN
4)          (SELECT SzállID
5)           FROM Szállít
6)           WHERE ÁruID IN
7)            (SELECT ÁruID
8)             FROM Áruk
9)             WHERE Név LIKE '%csoki%')
              );
```

A kereskedelmi rendszerek különböző mélységig tudják az alkérdéseket kezelni. Van olyan, amelyik csak 1 alkérdést engedélyez.

- *s* `> ALL` *R*, mely akkor igaz, ha *s* nagyobb, mint az *R* reláció minden értéke, ahol az *R* relációnak csak egy oszlopa van.
  - A > operátor helyett bármelyik összehasonlítási operátort használhatjuk.
  - Az *s* `<> ALL` *R* eredménye ugyanaz, mint az *s* `NOT IN` *R* feltételé.

példa: Legyen a következő lekérdezés:

```
SELECT SzemSzám, Név
FROM Alkalmazottak
WHERE Fizetés > ALL
    (SELECT MIN(Fizetés)
       FROM Alkalmazottak
       GROUP BY RészlegID);
```

Mivel az alkérdés több sort is visszaad, a „> ALL” operátort alkalmazva, a Fizetés oszlop értékét összehasonlítja az összes minimális fizetés értékkel az alkérdésből.

Tehát a lekérdezés megadja azon alkalmazottakat, kiknek fizetése nagyobb, mint a minimális fizetés minden részlegből.

- *s* `>` `ANY` *R*, mely akkor igaz, ha *s* nagyobb az *R* egyoszlopos reláció legalább egy értékénél.
  - A > operátor helyett akármelyik összehasonlítási operátort használhatjuk.

példa: „Keressük azokat a tanárokat, akik beosztása nem professzor, és van olyan professzor, akinek a fizetésénél az illető tanárnak nagyobb a fizetése.”

```
SELECT Név, BeosztásKod, Fizetés
FROM Tanárok
 WHERE Fizetés > ANY
    (SELECT Fizetés
      FROM Tanárok
      WHERE BeosztásKod = 'PRO')
      AND BeosztásKod <> 'PRO';
```

A *több oszlopos alkérdés:*

- a SELECT kulcsszó után megadhatunk több mint egy oszlopot,
- szükségszerűen a fő lekérdezésben is ugyanannyi oszlopot kell megadjunk az összehasonlító operátor bal oldalán is.
- az összehasonlítás párokra vonatkozik.

példa: „Keressük azokat a tanárokat, akiknek a fizetése egyenlő az algebra tanszék beosztásnak megfelelő átlag fizetésével.”

```
SELECT Név, BeosztásKod, Fizetés
FROM Tanárok
 WHERE BeosztásKod, Fizetés IN
    (SELECT BeosztásKod, AVG(Fizetés)
     FROM Tanárok
     WHERE TanszékKod = 'ALG'
     GROUP BY BeosztásKod); □
```

- alkérdés: az algebra tanszéken belül a beosztásoknak megfelelő átlagfizetések
- fő lekérdezés kiválaszt egy tanárt, ha az alkérdésben megtalálja a tanár beosztás kodja mellett a fizetést is

Korrelált alkérdések

- eddig bemutatott alkérdések esetén az alkérdés csak egyszer kerül kiértékelésre és a kapott eredményt a magasabb rendű lekérdezés hasznosítja.
- a beágyazott alkérdéseket úgy is lehet használni, hogy az alkérdés többször is kiértékelésre kerül.
- az alkérdés többszöri kiértékelését egy, az alkérdésen kívüli sorváltozóval érjük el.
- az ilyen típusú alkérdést *korrelált* alkérdésnek nevezzük.

példa: Az Egyetem adatbázis esetén keressük azon diákokat, akik egyedül vannak a csoportjukban 10-es átlaggal.

```
SELECT Név, CsopKod
FROM Diákok D1
WHERE Átlag = 10 AND NOT EXISTS
    (SELECT D2.BeiktatásiSzám
      FROM Diákok D2
      WHERE D1.CsopKod = D2.CsopKod
      AND D1.BeiktatásiSzám <> D2.BeiktatásiSzám
          AND D2.Átlag = 10);
```

lekérdezés kiértékelés:

- a D1 sorváltozó végigjárja a Diákok relációt;
- minden sorra a D1-ből a D2 sorváltozó végigjárja a Diákok relációt;
- legyen *d*1 egy sor a Diákok relációból, amelyet a fő lekérdezés az eredménybe helyez, ha megfelel a WHERE utáni feltételnek.
    - a *d*1.Átlag értéke 10 kell legyen
    - az alkérdés eredménye pedig üres halmaz.

– az alkérdés tartalmaz sorokat, ha létezik a Diákok relációban egy *d*2 sor: *d*2.csopkod=*d*1.csopkod, *d*2.átlag=10 és *d*2.BeiktatásiSzám <> *d*1.BeiktatásiSzám attribútum értékétől. (Vagyis:találtunk egy másik diákot, ugyanabból a csoportból, akinek az átlaga 10-es).

– ha az alkérdés üres halmaz, akkor kiválasztja a *d*1-et, és ekkor találtunk olyan diákot, aki egyedül van a csoportjában 10-es átlaggal.

Más típusú összekapcsolási műveletek

- relációs algebra természetes összekapcsolás művelete (SELECT - INNER JOIN) esetén, csak azok a sorok kerülnek be az eredmény relációba, melyek esetében a közös attribútum ugyanaz az értéke mindkét relációban megtalálható.
- a lógó sorok nem kerülnek be az eredménybe.
- bizonyos esetekben szükségünk van a lógó sorokra is.
- az OUTER JOIN kulcsszó segítségével azon sorok is megjelennek az eredményben, melyek értéke a közös attribútumra nem található meg a másik táblában, vagyis a lógó sorok, melyekben a másik tábla attribútumai NULL értékeket kapnak.
- a *külső összekapcsolás* (outer join) eredménye tartalmazza a belső összekapcsolás (*inner join*) eredménye mellett a lógó sorokat is.

A *külső összekapcsolás* 3–féle lehet:

1. külső baloldali összekapcsolás:

```
R LEFT OUTER JOIN S ON R.X = S.X
```

– eredménye tartalmazza a bal oldali *R* reláció összes sorát, azokat is, amelyek esetében az *X* attribútumhalmaz értéke nem létezik az *S* reláció *X* értékei között.
– az eredmény az *S* attribútumait is tartalmazza NULL értékekkel.

2. külső jobboldali összekapcsolás:

```
R RIGHT OUTER JOIN S ON R.X = S.X
```

– eredménye a jobb oldali *S* reláció összes sorát tartalmazza, azokat is amelyek esetében az *X* attribútumhalmaz értéke nem létezik az *R* reláció *X* értékei között.
– az eredmény az *R* attribútumait is tartalmazza NULL értékekkel.

3.

```
R FULL OUTER JOIN S ON R.X = S.X
```

eredménye azon sorokat tartalmazza, melyek esetében a közös attribútum értéke megegyezik mindkét relációban és mind a bal oldali *R* reláció lógó sorait, mind az *S* reláció lógó sorait magában foglalja.

SzemSzám	*Név*	*RészlegID*	*Fizetés*
111111	Nagy Éva	2	300
222222	Kiss Csaba	9	400
456777	Szabó János	9	900
234555	Szilágyi Pál	2	700
123444	Vincze Ildikó	1	800
567765	Katona József	NULL	600
556789	Lukács Anna	NULL	700
333333	Kovács István	2	500

RészlegID	*RNév*	*ManagerSzemSzám*
1	Tervezés	123444
2	Könyvelés	234555
3	Eladás	NULL
9	Beszerzés	456777

```
SELECT * FROM Alkalmazottak
INNER JOIN Részlegek
ON Alkalmazottak.RészlegID =
          Részlegek. RészlegID;
```

SzemSzám	***Név***	***RészlegID***	***Fizetés***	***RNév***	***ManagerSzemSzám***
111111	Nagy Éva	2	300	Könyvelés	234555
222222	Kiss Csaba	9	400	Beszerzés	456777
456777	Szabó János	9	900	Beszerzés	456777
234555	Szilágyi Pál	2	700	Könyvelés	234555
123444	Vincze Ildikó	1	800	Tervezés	123444
333333	Kovács István	2	500	Könyvelés	234555

```
SELECT * FROM Alkalmazottak
 LEFT OUTER JOIN Részlegek
 ON Alkalmazottak.RészlegID =
    Részlegek. RészlegID;
```

Szem Szám	***Név***	***Részleg ID***	***Fizetés***	***RNév***	***Manager SzemSzám***
111111	Nagy Éva	2	300	Könyvelés	234555
222222	Kiss Csaba	9	400	Beszerzés	456777
456777	Szabó János	9	900	Beszerzés	456777
234555	Szilágyi Pál	2	700	Könyvelés	234555
123444	Vincze Ildikó	1	800	Tervezés	123444
567765	Katona József	NULL	600	NULL	NULL
556789	Lukács Anna	NULL	700	NULL	NULL
333333	Kovács István	2	500	Könyvelés	234555

```
SELECT * FROM Alkalmazottak
RIGHT OUTER JOIN Részlegek
ON Alkalmazottak.RészlegID =
     Részlegek. RészlegID;
```

Szem Szám	***Név***	***RészlegID***	***Fizetés***	***RNév***	***Manager SzemSzám***
111111	Nagy Éva	2	300	Könyvelés	234555
222222	Kiss Csaba	9	400	Beszerzés	456777
456777	Szabó János	9	900	Beszerzés	456777
234555	Szilágyi Pál	2	700	Könyvelés	234555
123444	Vincze Ildikó	1	800	Tervezés	123444
333333	Kovács István	2	500	Könyvelés	234555
NULL	NULL	3	NULL	Eladás	NULL

```
SELECT * FROM Alkalmazottak
FULL OUTER JOIN Részlegek
ON Alkalmazottak.RészlegID =
 Részlegek. RészlegID;
```

Szem Szám	***Név***	***Részleg ID***	***Fizetés***	***RNév***	***Manager SzemSzám***
111111	Nagy Éva	2	300	Könyvelés	234555
222222	Kiss Csaba	9	400	Beszerzés	456777
456777	Szabó János	9	900	Beszerzés	456777
234555	Szilágyi Pál	2	700	Könyvelés	234555
123444	Vincze Ildikó	1	800	Tervezés	123444
333333	Kovács István	2	500	Könyvelés	234555
567765	Katona József	NULL	600	NULL	NULL
556789	Lukács Anna	NULL	700	NULL	NULL
NULL	NULL	3	NULL	Eladás	NULL

Változtatások az adatbázisban

Beszúrás

```
INSERT INTO R(A1, A2, …, An)
       VALUES (v1, v2, …, vn);
```

- A művelet során az R relációba egy új sor kerül, ahol az A_i attribútum értéke v_i, minden $i = 1,2, \ldots, n$.
- Ha az attribútumlista nem tartalmazza R összes attribútumát, akkor a hiányzó attribútumok az alapértelmezés szerinti értéket kapják
- A leggyakrabban használt alapértelmezés szerinti érték a nullérték, tehát ha nem adjuk meg valamelyik attribútum értékét, az NULL lesz.

példa: Legyen a Részlegek reláció és a következő parancs:

```
INSERT INTO Részlegek
          (RészlegID, RNév, ManagerSzemSzám)
      VALUES (4, 'Termelés', 567765);
```

- Minden beszúrási művelet esetén az ABKR ellenőrzi a megszorításokat, ha a beszúrásra kerülő sor nem teljesít egyet is a megszorítások közül, hibát jelez és nem hajtja végre a beszúrási műveletet.
- Ha megadjuk az összes attribútum értéket, akkor nem szükséges felsorolni az attribútum neveket, csak arra kell ügyelnünk, hogy pontosan abban a sorrendben adjuk meg őket, ahogy a relációban szerepelnek.

példa: Az előbbi parancsot megadhatjuk attribútumlista nélkül is:

```
INSERT INTO Részlegek
  VALUES (4, 'Termelés', 567765);
```

példa:

```
INSERT INTO Alkalmazottak
 (SzemSzám, Név, RészlegID)
  VALUES (414141, 'Tamás Erika', 4);
```

– a Fizetés attribútumot nem adtuk meg, ennek értéke NULL értékkel fog feltöltödni, mivel nincs alapértelmezett értéke, viszont lehet null.

példa: a következő parancs hibát fog jelezni, mivel 111111 személyi számmal már létezik alkalmazott és a SzemSzám elsődleges kulcs:

```
INSERT INTO Alkalmazottak
 (SzemSzám, Név, RészlegID)
   VALUES (111111, 'Tamás Zoltán', 4);
```

- eddig egy sort szúrtunk be egy relációba.
- az attribútumlista helyett megadhatunk egy alkérdést is, melynek segítségével több sort is beszúrhatunk.

```
Részlegek(RészlegID, Név, Helység,ManSzemSzám);
Alkalmazottak(SzemSzám, Név, Fizetés, Cím,
                          RészlegID);
```

- a két reláció kölcsönösen hivatkozik egymásra külső kulcsos megszorítással,
- az Alkalmazottak táblába először begyűjtjük az összes managert,
- átvesszük az összes létező részleget a Részlegek relációba a manager személyi számával együtt.

```
INSERT INTO Részlegek (RészlegID, ManSzemSzám)
   SELECT DISTINCT RészlegID, SzemSzám
       FROM Alkalmazottak;
```

Törlés

```
DELETE FROM R WHERE <feltétel>;
```

- A FROM kulcsszó után meg kell adjuk azon relációt, melyből sorokat szeretnénk törölni.
- Az utasítás azt eredményezi, hogy a megadott relációból kitörlődik minden olyan sor, mely a WHERE kulcsszó után megadott feltételt teljesíti.

példa:
```
DELETE FROM Alkalmazottak
WHERE SzemSzám = 111111;
```

SzemSzám	***Név***	***RészlegID***	***Fizetés***
111111	Nagy Éva	2	300

- Egy adott sor törlése esetén a legjobb, ha a reláció elsődleges kulcsának az értékét, vagy egy egyedi kulcsnak az értékét adjuk meg.
- Abban az esetben, ha a feltétel nem egyedi kulcs értékére vonatkozik, valószínű, hogy több sor is törlődik.

```
DELETE FROM Alkalmazottak
 WHERE Fizetés <= 400;
```

Módosítás

Módosíthatunk egy vagy több sort egy relációban:

```
UPDATE R SET <értékadások> WHERE <feltétel>;
```

A parancs az *R* reláció sorait módosítja,

az értékadásokat vesszővel választjuk el.

minden értékadás az *R* egy attribútumát, egyenlőségjelet és egy kifejezést tartalmaz.

példa:
```
UPDATE Alkalmazottak
   SET Név = 'Nagy Éva Mária', Fizetés = 450
      WHERE SzemSzám = 111111;
```

Több sort is módosíthatunk egy UPDATE parancs segítségével.

példa:
```
UPDATE Alkalmazottak
   SET Fizetés = Fizetés * 1.2
      WHERE Fizetés < 600;
```

példa: A WHERE feltétel tartalmazhat alkérdéseket is:

```
UPDATE Alkalmazottak
SET Fizetés = Fizetés * 1.5
 WHERE SzemSzám IN
     (SELECT SzemSzám FROM Managerek);
```

Problémák a törlés, beszúrás esetén akkor jelenhetnek meg, ha egy relációnak nincs egyedi kulcsa,

- ugyanazt a sort többször is be tudjuk szúrni,
- törlés esetén pedig egy parancs az összes azonos sort kitörli, nem tudjuk csak az egyiket azonosítani.

példa: Legyen a Szállít reláció és tegyük fel, hogy nem adtuk meg a (SzállID, ÁruID) párost elsődleges kulcsnak. Az aktuális értékek a relációban:

SzállID	*ÁruID*	*Ár*
111	45	2.5
222	45	2.6
111	67	1.7
111	56	2.2
222	67	1.8
222	56	2.2

```
INSERT INTO Szállít VALUES (111, 45, 2.5);
```

```
DELETE FROM Szállít
WHERE SzállID = 111 AND ÁruID = 45;
```