

Lekérdezések végrehajtása

Adatbázisok tervezése, megvalósítása
és menedzselése

Lekérdezések lefordítása I.

SQL lekérdezés



Lekérdezés
elemzése



Logikai
lekérdezésterv
kiválasztása



Fizikai terv
kiválasztása



Lekérdezés
kifejezésfája

Logikai lekérdezésterv
kifejezésfája

Fizikai lekérdezésterv
kifejezésfája



Lekérdezések végrehajtása II.

- A **lekérdezésfordítás** három legfontosabb lépése:
 - **elemzés** (**parsing**): ennek során egy elemző fa készül,
 - **lekérdezés átírása** (**query rewriting**): az elemző fából egy kezdeti lekérdezéstervet készítünk, amely a műveleteknek egy algebrai megvalósítását tartalmazza. Ezt a tervet aztán egy olyan ekvivalens lekérdezési tervvé alakítjuk át, amely várhatóan rövidebb idő alatt végrehajtható. Ez a **logikai lekérdezésterv** (**logical query plan**).
 - **fizikai terv előállítás** (**physical plan generation**). A logikai terv valamennyi operátorának megvalósítására kiválasztunk egy algoritmust és meghatározzuk ezek végrehajtási sorrendjét. A fizikai tervet egy lekérdezésfával ábrázoljuk. Itt olyan részletek is szerepelnek, hogy az egyes relációkhoz miként férünk hozzá, illetve, hogy a relációkat kell-e rendezni, és ha igen, mikor.
- Az utóbbi két lépést **lekérdezés optimalizálásnak** (**query optimization**) nevezzük.

Mivel foglalkozunk ma?

- A mai kilencven percben elsősorban a második és harmadik lépéshez szükséges „eszközökkel” foglalkozunk.
- Az SQL lekérdezések élethűbb leírásához új relációs algebrai műveleteket vezetünk be és a régi és új műveleteinket halmazok helyett **multihalmazok** fölött értelmezzük.
- Ezek után algoritmusokat vizsgálunk az egyes műveletek végrehajtására.

Mivel foglalkozunk egy hét múlva, vagy valamikor?

- Megtanuljuk, miként lehet elkészíteni egy-egy SQL lekérdezéshez a megfelelő **elemző fát**.
- Majd az elemző fát átalakítjuk egy **kezdeti lekérdezéstervvé**, amely már csak relációs algebrai műveleteket tartalmaz.
- Szabályokat tanulunk, amelyek segítségével az iménti kezdeti tervből optimálisabb tervet készíthetünk.
- **Stb.**

Relációs algebra kiterjesztése

- Eddig tanult műveletek: projekció (Π), szelekció (σ), Descartes-szorzat (\times), unió (\cup), különbség ($-$), természetes összekapcsolás (\bowtie), Théta-összekapcsolás (\bowtie_c), átnevezés (ρ).
- Ezeket most multihalmazok fölött értelmezzük, azaz egy reláció nem sorok **halmazából**, hanem **multihalmazából** áll, ahol **megengedettek az ismétlődések**.
- Ezeken kívül bevezetjük még:
 - az ismétlődések kiküszöbölésére szolgáló műveletet (δ),
 - a rendezést (τ_L)
 - csoportosítás és összesítést (γ_L)
 - a projekciónak egy kiterjesztett változatát.

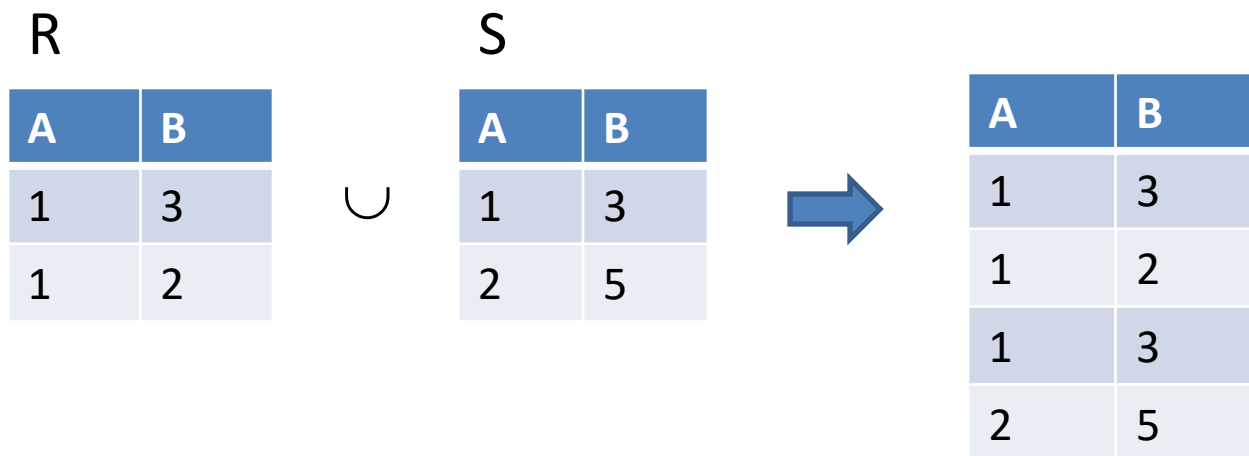
Kiterjesztett projekció

- $\Pi_L(R)$ listája az SQL `SELECT` záradékához hasonlóan tartalmazhatja:
 - R egy attribútumát,
 - $x \rightarrow y$, ahol x, y attribútumnevek, s itt x -et y -ra nevezzük át,
 - $E \rightarrow y$, ahol E R attribútumait, konstansokat, aritmetikai operátorokat és karakterlánc operátorokat tartalmazhat például: $A + 5$,
 $C || 'nevű emberek'$.

R			$\Pi_{A, B+C \rightarrow x}(R)$	
A	B	C	A	x
0	1	3	0	4
4	9	1	4	10
5	5	7	9	12


Multihalmaz műveletek

- **Unió:** $R \cup S$ -ben egy t sor annyiszor fordul elő ahányszor előfordul R -ben, plusz ahányszor előfordul S -ben.
- **Metszet:** $R \cap S$ -ben egy t sor annyiszor fordul elő, amennyi az R -ben és S -ben lévő előfordulások minimuma.
- **Különbség:** $R - S$ -ben egy t sor annyiszor fordul elő, mint az R -beli előfordulások mínusz az S -beli előfordulások száma.



A „többi művelet” multihalmazok fölött

- A projekció, szelekció, Descartes-szorzat, természetes összekapcsolás és Théta-összekapcsolás végrehajtása során nem küszöböljük ki az ismétlődéseket.

R			$\Pi_A(R)$	
A	B		A	
1	2		1	
1	5		1	
2	3		2	

Ismétlődések kiküszöbölése és rendezés

- **Ismétlődések kiküszöbölése:** $\delta(R)$.

A művelet jelentése egyértelmű. A DISTINCT reprezentálására szolgál.

- **Rendezés:** $\tau_{A_1, \dots, A_n}(R)$.

Először A_1 attribútum szerint rendezzük R sorait. Majd azokat a sorokat, amelyek értéke megegyezik az A_1 attribútumon, A_2 szerint é.í.t. Hasonlóan az ORDER BY működéséhez.

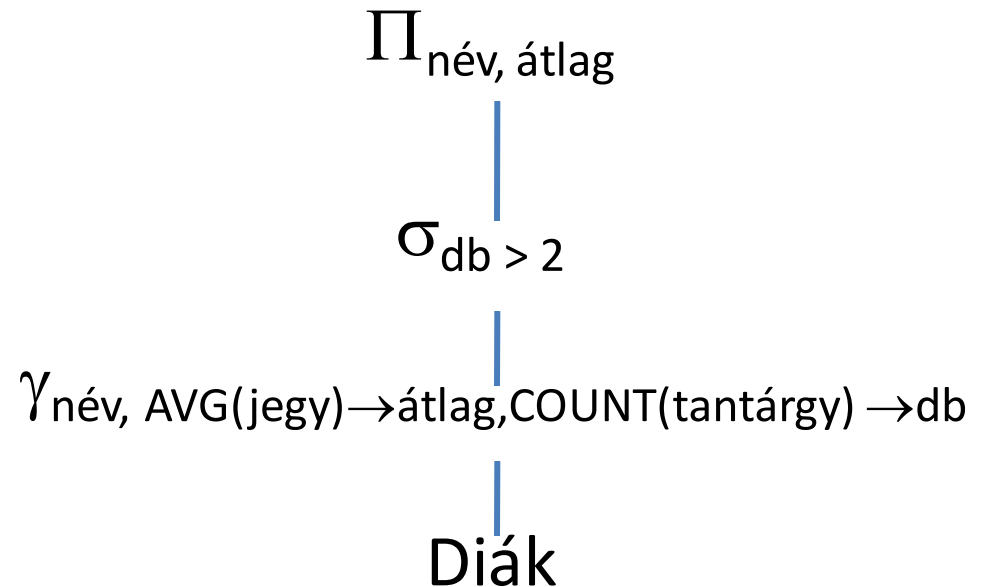
Csoportosítás és összesítés I.

- A művelet a **csoportosítást** (GROUP BY), a csoportokon végezhető **összesítő függvényeket** (AVG, SUM, COUNT, MIN, MAX) reprezentálja.
- Emellett segít a **csoportokon megadható feltételek** (HAVING) leírásában.
- A művelet jele: $\gamma_L(R)$.
- Itt az **L** lista valamennyi eleme a következők egyike:
 - R egy attribútuma: ez az attribútum egyike a csoportosító attribútumoknak (a GROUP BY után jelenik meg).
 - a reláció egyik attribútumára alkalmazott összesítő operátor.

Csoportosítás és összesítés II.

A példa reláció: **Diák (név, tantárgy, jegy)**

```
SELECT név, AVG(jegy) AS átlag  
FROM diák  
GROUP BY név  
HAVING COUNT(tantárgy) > 2;
```



Feladatok

R

A	B
0	1
0	1
2	3
2	5
1	4

- Mi lesz $\gamma_{A, \text{SUM}(B), \text{COUNT}(B)}$ (R) eredménye a példatáblán?
- Adjuk meg a következő lekérdezésnek megfelelő relációs algebrai kifejezést! (Az $S(A, B, C, D)$ relációt használjuk.)

```
SELECT A, COUNT (C)
FROM R
WHERE B > 50
GROUP BY A
HAVING MIN (D) = 123;
```

Fizikai lekérdezés-operátorok

- A fizikai operátorok gyakran a relációs algebrai műveletek megvalósításai.
- Szükségünk van olyan operátorokra is, amelyek nem kapcsolódnak egyik algebrai művelethez sem.
- Ilyen például a **tábla átvizsgálása (scan)**. Két alapvető megközelítés:
 - az R reláció a háttértárolón (**disc**) blokkokba szervezett, a blokkok elhelyezkedése ismert a rendszer számára, ezért lehetséges azok egymás utáni beolvasása. Ez a tulajdonképpeni **tábla átvizsgálás**.
 - **Index alapú átvizsgálás**: ekkor R blokkjait egy indexen keresztül érjük el.

A költségek kiszámításának modellje

- A költségek méréséhez a **lemez I/O műveletek** számát fogjuk használni.
- Ezen kívül figyelembe szokták még venni a **CPU költséget** is (Oracle), illetve osztott adatbázisok esetén a **hálózati kommunikáció költségét**.
- Emellett feltételezzük, hogy egy tetszőleges operátor argumentumai a lemezen találhatóak, viszont a **végeredmény a memóriában marad**.
- Ez utóbbi okai:
 - az **eredmény kiírásának költsége** csak a válasz méretétől függ, a kiszámítás módjától nem.
 - sok esetben az eredmény **a memóriában marad** (pl. átadjuk egy programnak vagy egy másik fizikai operátornak).

A költségbecslés paramétereit I.

- Tegyük fel, hogy a **memória puffereinek mérete megegyezik a lemezblokkok méretével**. A rendelkezésünkre álló pufferek számát **M**-mel jelöljük.
- Sok esetben **ez csupán egy becslés**, ráadásul a művelet végrehajtása során is változhat.
- Mivel az operátor költségeinek kiszámításakor nem számoljuk a kimenet előállításának költségét, ezért **M csak a bemenet és a közbeeső eredmények tárolására szolgál**.

A költségbecslés paramétereii II.

- Az egyszerűség kedvéért feltételezzük, hogy a lemezen lévő adatokhoz blokkonként férhetünk hozzá, és egyszerre csak egy blokk kerül beolvasásra.
- $B(R)$ adja meg, hogy R hány blokkba fér el összesen. (**Nem biztos, hogy R ennyi blokkban is tárolódik!**)
- Az R reláció **nyalábolt** (**clustered**), ha megközelítőleg $B(R)$ darab blokkban helyezkedik el.
- $T(R)$ jelöli R sorainak a számát.
- Ha R sorai szétszórva helyezkednek el a blokkokban – pl. klaszter esetén –, feltételezzük, hogy **R minden egyes sorát külön blokkban olvassuk be.**
- $V(R, A)$ jelenti az R A attribútumához tartozó különböző értékek számát. $V(R, [A_1, \dots, A_n])$ az A_1, \dots, A_n attribútumokhoz tartozó különböző n -esek számát adja meg.

Egymenetes algoritmusok

- Itt az adatokat csak egyszer kell a lemezről beolvasni.
Általában csak akkor működnek, ha az argumentumoknak legalább egyike befér a memóriába.
- A műveletek felosztása:
 - soronkénti, unáris műveletek: projekció, szelekció. Ezek nem igénylik, hogy a teljes reláció a memóriában legyen, így a blokkokat egyenként olvashatjuk be.
 - Unáris, teljes relációs műveletek: δ , γ , τ . A sorok nagyobb részét kell egyszerre a memóriában tartani. Az egymenetes algoritmusok ebben az esetben csak akkor működnek, ha a teljes reláció befér a memóriába.
 - Bináris, teljes relációs műveletek.

Soronkénti műveletek egymenetes algoritmus

- R blokkjait egyenként beolvassuk, s a műveleteket (**projekció** vagy **szelekció**) minden sorra elvégezzük.
- Ha R nyalábolt, az I/O műveletek száma: **$B(R)$** . Különben: **$T(R)$** .
- Kivételt képez az az eset, amikor a szelekcióban szereplő attribútumok némelyikére indexet hoztunk létre.

Ismétlődések kiküszöbölése egy menetben

- A memória egy pufferébe R sorait olvassuk be egymás után.
- A megmaradó $M-1$ pufferben tároljuk a már előfordult sorok másolatát.
- Itt egy olyan adatszerkezetre van szükség, amely lehetővé teszi:
 - új sorok hozzáadását,
 - annak gyors (majdnem konstans idejű) meghatározását, hogy egy adott sor már szerepelt-e.
- Ilyen szerkezet lehet egy **tördelőtábla nagyszámú kosárral**, vagy a **kiegyensúlyozott bináris keresési fák** egy formája.
- **Itt $B(\delta(R))$ mérete nem lehet nagyobb $M-1$ -nél.** (A segédstruktúra által igényelt plusz memóriaterületet nem vesszük figyelembe.)
- Ha rosszul becslünk, annak komoly ára lehet. (**Mehetünk ebédelni, amíg a művelet befejeződik.**)

Csoportosítás egy menetben

- γ_L nulla vagy több csoportosító attribútumot, és nulla vagy több összesített attribútumot ad vissza.
- A memória M-1 pufferében a csoportosító attribútumok minden értékének egy bejegyzést hozunk létre, miközben a maradék egy blokkba R sorait olvassuk be egymás után.
- Egy csoporthoz tartozó bejegyzés (entry) csoportosító attribútumok értékéből és a kumulált értékekből áll.
- A kumulált értékek számítása a MIN, MAX, COUNT, SUM összesítő függvények esetén egyértelmű.
- AVG esetén a SUM és COUNT értékeket kell tárolni, majd a beolvasás végeztével ezek hányadosát kell venni.
- A szükséges memória becslésénél gondot jelent, hogy egy-egy csoport bejegyzése lehet az R sorainál rövidebb, de akár hosszabb is.

Rendezés egy menetben

- A végrehajtás nagyon egyszerű. Az adatokat beolvassuk a memóriába, majd itt valamelyik népszerű rendezési algoritmussal (**QuickSort**) rendezünk.
- Mindhárom esetben (ismétlődések kiküszöbölése, csoportosítás, rendezés) a műveletigény (I/O): **$B(R)$** , ha R nyálábolt, különben **$T(R)$** .

Halmazműveletek egy menetben

- A továbbiakban feltesszük, hogy S a kisebbik reláció.
- Ekkor S -et beolvassuk $M-1$ pufferbe, majd olyan keresési struktúrát építünk fel (tördelőtáblázat, kiegyensúlyozott bináris fa), ahol a keresési kulcs maga a teljes sor. A maradék pufferbe R sorait olvassuk.
- Feltesszük, hogy S ismétlődéseit a beolvasás során megszüntettük.
- **Unió**: R egy t sorát bemásoljuk a kimenetbe, ha nem szerepel S sorai közt. R beolvasása után S -et is a kimenetbe másoljuk.
- **Metszet**: ha t egy sora S -ben is szerepel, a kimenetbe kerül.
- $R - S$, $S - R$ kiszámítása ugyanígy triviális.
- Szükséges feltétel: $\min(B(S), B(R)) \leq M-1$.
- Műveletigény (I/O): $B(R) + B(S)$, ha mindkét reláció nyalábolt.

Multihalmaz műveletek egy menetben

- **Unió:** triviális. R és S beolvasásához, mérettől függetlenül, elegendő egy puffert felhasználni. Innen minden kerül azonnal az eredménybe.
- S-et beolvassuk M-1 pufferbe. A sorokhoz (az ismétlődő sorok egyszer szerepelnek) **egy számlálót rendelünk**, ami megadja, hogy a szóban forgó sor S-ben hányszor szerepel.
- **Metszet:** ha R egy t sora elfordul S-ben úgy, hogy a hozzá tartozó számláló nem nulla, akkor t a kimenetbe kerül, és a számláló értéke csökken 1-gyel.
- **R – S:** ha R egy t sora nem szerepel S-ben, vagy, ha az S-beli számláló 0, akkor a kimenetbe kerül. Különben nem, de a megfelelő S-beli sor számlálóját 1-gyel csökkenteni kell.
- **Szükséges feltétel:** $\min(B(S), B(R)) \leq M-1$, **műveletigény** (I/O): $B(R) + B(S)$, ha mindkét reláció nyálábolt.

Szorzat és természetes összekapcsolás egy menetben

- **Szorzat**: S-et beolvassuk M-1 pufferbe. Itt semmiféle különleges adatszerkezetre nincs szükség. A maradék egy pufferbe R sorait olvasva egy R-beli t sort minden S-beli sorral párosítunk.
- **Természetes összekapcsolás**: tegyük fel, hogy $R(X, Y)$, $S(Y, Z)$ alakú, ahol Y jelöli a közös attribútumok halmazát.
- S M-1 pufferbe történő beolvasásakor olyan keresési struktúrát alkalmazunk, ahol Y attribútumai alkotják a keresési kulcsot. (Az algoritmus leírása triviálisan folytatható).
- Szükséges feltétel: $\min(B(S), B(R)) \leq M-1$, műveletigény (I/O): $B(R) + B(S)$, ha mindkét reláció nyálábolt.

Beágyazott ciklusú összekapcsolások I.

- A beágyazott ciklusú összekapcsolások **tetszőleges méretű relációk esetén használhatóak.**
- Manapság inkább hatékonyabb összekapcsolási algoritmusok szubrutinjaként használatos.
- Legegyszerűbb fajtája: **a sor alapú beágyazott ciklusú összekapcsolás:**

```
FOR S minden s sorára DO
    FOR R minden r sorára DO
        IF s és r összekapcsolható egy t sorrá
            THEN t kiírása;
```

- Ez azonban $T(R) \cdot T(S)$ műveletet is igényelhet.

Beágyazott ciklusú összekapcsolások II.

- Jobban járunk, ha S lehető legnagyobb részét beolvassuk $M-1$ memóriapufferbe, ahol a keresési struktúra az Y attribútumaira épül.
- Itt ismételten feltettük, hogy $R(X, Y)$, $S(Y, Z)$ a relációk sémája, valamint $B(S) \leq B(R)$. Feltesszük még, hogy $M-1 \leq B(S)$ is teljesül.
- A maradék egy blokkba R sorait olvassuk be, és a megfelelő sorokat összekapcsoljuk. Mikor R sorait végigolvastuk, S egy újabb szeletét olvassuk be az $M-1$ pufferbe.
- A műveletigény:
$$\frac{B(S)}{M-1} (M-1 + B(R))$$

Feladat

- Tegyük fel, hogy $B(R) = B(S) = 10000$. Ekkor M milyen értéke mellett lenne:
 - a. 100000,
 - b. 25000I/O műveletre szükség a beágyazott ciklusú összekapcsolás végrehajtásához?

Összefésülő rendezés (Merge-Sort)

- Két **rendezett** lista **összefésülése**:
 - a listák legkisebb megmaradt kulcsértékeit hasonlítjuk össze. A kisebbiket töröljük a listájából és az eredménybe tesszük.
 - Ezt az eljárást addig ismételjük, míg végül egyetlen lista marad, melynek elemeit változtatás nélkül másoljuk be az eredménybe.
- Az algoritmus:
 - **indukciós alap**: ha egy lista egyetlen elemet tartalmaz, akkor kész vagyunk.
 - **Indukció**: egynél több elem esetén, akkor osszuk fel a listát két egyenlő részre, ezeket rendezzük rekurzívan, majd az eredményül kapott részlistákat fésüljük össze.

$T(n) = 2T(n/2) + an$ ahol $T(n)$ a műveletek számát adja meg n hosszú lista esetén. Így:

$$T(n) = O(n \log n)$$

Kétfázisú, többutas, összefésülő rendezés

- Feltesszük, hogy a teljes fájl **nem fér be a központi memóriába**. Az előbbi algoritmus egy változataként :
 - az **első fázisban** olvassuk be az adatokat és az M blokknyi részeket rendezzük a központi memóriában. Tegyük fel, hogy így **k** db rendezett részlistánk keletkezett.
 - A **második fázisban** minden rendezett listából olvassuk be az első blokkot és lépésenként a legkisebb elemeket tegyük be az eredménybe. Ha egy blokk kiürül, olvassuk be a megfelelő részlista következő blokkját.
- **Megjegyzés**: a legkisebb elemet a részlisták számának logaritmusával arányos idő alatt lehet megtalálni.
- Az alkalmazás feltétele: **$k \leq M-1$** , hiszen egy blokkot az eredmény számára kell fenntartanunk, vagyis:

$$\frac{B(R)}{M} \leq M - 1 \quad \text{kell, hogy teljesüljön.}$$

Többfázisú, többutas, összefésülő rendezés

- Ha az előbbi feltétel nem teljesül ($k > M-1$), akkor a második fázisban az első $M-1$ részlistát kell az előbbi módszerrel összefésülnünk, aztán a következő $M-1$ darabot é.í.t. Tegyük fel, hogy így s számú rendezett részlistánk keletkezett.
- Ha $s \leq M-1$, vagyis a részlisták első blokkjai már beférnek a központi memóriába, és egy blokk még üresen is marad, akkor a harmadik fázisban a részlistákat összefésülhetjük.
- A háromfázisú többutas, összefésülő rendezés alkalmazásának feltétele:

$$\frac{B(R)}{M(M-1)} \leq M-1.$$

- k fázis esetén $(2k-1) * B(R)$ I/O műveletet kell elvégezni, ha R nyalábolt.

Feladat

- Hajtsuk végre a többfázisú, többutas, összefésülő rendezést $M = 4$ mellett a következő táblára (A szerint rendezünk):

A	B
12	2
2	6
9	30
15	2
2	2
3	6
10	1
6	1
6	5
6	14
2	9
13	13

Kétmenetes algoritmusok

- Azokat az algoritmusokat, amelyek azt az esetet kezelik, amikor a **vizsgált relációk nem férnek be a központi memóriába**, három csoportba szokták osztani:
 - rendezésen alapuló,
 - tördelésen alapuló és
 - index alapú algoritmusok.

Rendezésen alapuló algoritmusok

- A rendezésen alapuló algoritmusok **első lépése mindig ugyanaz** és megegyezik a többutas, összefésülő rendezés első lépésével.
- R-nek **M** darab blokkját beolvassuk a memóriába.
- Rendezünk, majd az eredményt visszaírjuk a háttértárolóra.
- Aztán vesszük a következő **M** darab blokkot é.í.t.
- Egy-egy ilyen listára **rendezett részlistaként** hivatkozunk majd.
- Tegyük fel a továbbiakban, hogy **k** darab rendezett részlistánk van.

Ismétlődések kiküszöbölése rendezéssel

- A **második fázisban** beolvassuk a **k** darab részlista első blokkjait a memóriába. (Kell, hogy $k \leq M-1$ teljesüljön.)
- Vesszük az egyes blokkok első, még nem vizsgált sorát és ezek közül kiválasztjuk a legkisebbet, legyen ez **t**.
- **t** bekerül az eredménybe, az összes többi példányát pedig eltávolítjuk.
- Ha egy blokk kiürül, beolvassuk a rendezett részlista következő blokkját.
- Ha **R** **nyalábolt**, a műveleti költség (I/O): $3 * B(R)$. Ha nem nyalábolt: $3 * T(R)$.
- Ha $k > M-1$, ugyanaz az ötlet alkalmazható, mint ami a többutas, összefésülő rendezésnél szerepelt.
- $k \leq M-1$ teljesülésének feltétele szintén ugyanaz, mint ott.

Csoportosítás és összesítés rendezéssel

- $\gamma_L(R)$ esetén R sorait az első fázisban **L attribútumai szerint csoportosítjuk**, azaz:
 - ha L -ben sorra A_1, \dots, A_k a csoportosító attribútumok, akkor először A_1 szerint rendezünk. Aztán azon soroknál, amelyek A_1 -beli értéke megegyezik, A_2 szerint é.í.t.
- A második fázisban ugyanúgy beolvassuk a részlisták első blokkjait, és **megkeressük a legkisebb értéket az L rendezési kulcs szerint**. Legyen ez az érték λ .
- Vesszük azokat a sorokat, amelyek L attribútumokon felvett értéke λ (ezek értelemszerűen a beolvasott blokkok elején helyezkednek el), s az összesítést az egymenetes algoritmusoknál ismertetett módon hajtjuk végre.
- **Műveletigény** és **alkalmazhatóság** szempontjából ugyanaz mondható el, mint korábban.

Halmazműveletek rendezés segítségével I.

- Az **első fázisban** **R és S soraiból is elkészítjük a rendezett részlistákat**, majd a részlisták első blokkjait a memóriába töltjük (M-1 helyre). Újra és újra vesszük a legkisebb t sort.
- **Unió** esetén t-t bemásoljuk az kimenetbe, az azonos sorokat pedig eltávolítjuk.
- **Metszet** esetén t akkor kerül be a kimenetbe, ha R-ben és S-ben is szerepel.
- **Multihalmazmetszet** esetén t annyiszor kerül a kimenetbe, amennyi az R-beli és S-beli előfordulások minimuma.
- **$R - S$** esetén t akkor kerül be a kimenetbe, ha S-ben nem szerepel.
- **Multihalmazkülönbség** esetén t annyiszor kerül ki a kimenetbe, amennyi az R-beli és S-beli előfordulások különbsége. Ha ez negatív, akkor nyilván egyszer sem.

Halmazműveletek rendezés segítségével II.

- A műveletigény (I/O) két menet esetén: $3 \cdot (B(R) + B(S))$, ha mind a két tábla nyálábolt.
- Az alkalmazhatóság feltétele két menet esetén:

$$B(R) + B(S) \leq M(M - 1)$$

Összekapcsolás rendezéssel

- Ismét $R(X, Y)$ és $S(Y, Z)$ relációkat szeretnénk összekapcsolni, ahol X, Y, Z is attribútumhalmaz.
- Itt előfordulhat, hogy az Y attribútumokat tekintve olyan sok az y értékű sor a két relációban, hogy azok nem férnek be a memóriába. Ebben az esetben – jobb híján – venni kell e két sorhalmaz beágyazott ciklusú összekapcsolását.
- Fontos lehet tehát, hogy minél több szabad pufferünk legyen a közös értékekkel rendelkező sorok számára.

Egy egyszerű algoritmus I.

- Az alábbi algoritmus a **lehető legtöbb szabad puffert garantálja** az azonos értékkel bíró sorok számára.
- **Az algoritmus:**
 - i. R-et és S-et is rendezzük kétfázisú (vagy többfázisú), többutas összefésüléssel.
 - ii. R és S aktuális blokkját beolvassuk egy-egy pufferbe.
 - iii. Megkeressük a legkisebb **y** értéket.
 - iv. Ha **y** nem jelenik meg a másik relációban, akkor eltávolítjuk az **y** értékű sorokat
 - v. Egyébként azonosítjuk az összes **y** értékű sort. Ehhez, ha kell, további blokkokat olvasunk R-ből vagy S-ből. Ezeket a sorokat **M-1** pufferben tárolhatjuk.
 - vi. A kimenetbe másoljuk e sorok összekapcsoltjait.

Egy egyszerű algoritmus II.

- Ha valamilyen y értékre az ilyen értékű sorok nem férnek be $M-1$ pufferbe, akkor két eset lehetséges:
 - i. ha valamelyik reláció, mondjuk R , y értékű sorai beférnek $M-1$ pufferbe, akkor a maradék egy puffert használva beolvassuk S y értékű sorait, és összekapcsoljuk a sorokat. Ez olyan, mintha egy menetes összekapcsolást használnánk az y értékű sorok összekapcsolására.
 - ii. Ha az előbbi feltétel nem teljesül, beágyazott ciklusú összekapcsolást kell használnunk
- Műveletigény az előbbi komplikációk nélkül (I/O):
 $5 \cdot (B(R) + B(S))$, ha a relációink nyaláboltak és kétfázisú, többutas rendezést használhattunk.
- Ekkor az alkalmazhatóság két szükséges feltétele:
 $B(R) \leq M(M - 1), B(S) \leq M(M - 1)$.

Egy hatékonyabb algoritmus

- Ha nem kell aggódni az azonos y értékkel rendelkező sorok nagy száma miatt, akkor a korábban használt algoritmus ehhez a helyzethez igazított változata használható.
- Y rendezési kulcsot használva R -et és S -et beolvasva rendezett részlistákat készítünk, majd e részlisták első blokkjait újfent beolvassuk a memóriába.
- Itt újra és újra megkeressük a legkisebb y értéket, majd azokat a sorokat, amelyek y értékkel bírnak, s vesszük a sorok összekapcsoltjait. Itt használhatjuk az üresen maradó memóriapuffereket, ha vannak.
- Műveletigény (I/O): $3 \cdot (B(R) + B(S))$.
- Alkalmazhatóság feltétele:
 $B(R) + B(S) \leq M(M - 1)$.

Mikor nem kell aggódnunk?

- **Például:** ha **Y R kulcsa**, akkor csak egyetlen olyan R-beli sor van, ahol **y** előfordulhat. Ekkor az előbbi algoritmusban S **y** értékű sorait már kapcsolhatjuk össze ezzel a sorral, **nincs is szükség pluszhelyre**.
- **Vagy:** $B(R) + B(S)$ sokkal kisebb $M \cdot (M-1)$ -nél, ekkor **sok kihasználatlan pufferünk marad**, ahol az azonos értékű sorokat tárolhatjuk.

Feladat

- Tegyük fel, hogy $B(R) = B(S) = 10000$ és $M = 1000$. Mi lesz ekkor *a)* halmazegyesítés (metszet), *b)* egyszerű rendezéses összekapcsolás, *c)* hatékonyabb rendezéses összekapcsolás végrehajtási költsége (csak az I/O műveletek számítanak)?

<i>Operátor</i>	<i>Szükséges M kb.</i>	<i>Lemez I/O-művelet</i>	<i>Rész</i>
γ, δ	\sqrt{B}	$3B$	6.5.1., 6.5.2.
$\sqcup, \cap, -$	$\sqrt{B(R + B(S))}$	$3(B(R) + B(S))$	6.5.3., 6.5.4.
\bowtie	$\sqrt{\max(B(R), B(S))}$	$5(B(R) + B(S))$	6.5.5.
\bowtie	$\sqrt{(B(R) + B(S))}$	$3(B(R) + B(S))$	6.5.7.

6.16. ábra. A rendezés alapú algoritmusok memória- és lemez I/O-művelet követelményei

Tördelésen alapuló algoritmusok

- Ha az adatok mennyisége túl nagy ahhoz, hogy a memóriában tároljuk, akkor a megvalósítandó művelethez választott megfelelő tördelőkulcs segítségével kosarakba oszthatjuk a sorokat, majd **az egyes kosarakra végezzük el a műveletet.**

Hogyan tördeljük be R-et M-1 kosárba?

```
inicializáljunk M-1 kosarat M-1 üres puffer használatával;  
FOR R minden b blokkjára DO  
    olvassuk be b-t az M-dik pufferbe;  
    FOR b minden t sorára DO  
        IF a h(t) kosárban nincs hely t számára THEN  
            BEGIN  
                másoljuk ki a puffert a lemezre;  
                inicializáljunk egy új üres blokkot;  
            END;  
        másoljuk a t sort a h(t) kosár pufferébe;  
    END;  
END;  
FOR minden kosárra DO  
    IF az adott kosár puffere nem üres THEN írjuk ki;  
END;
```

Ismétlődések kiküszöbölése tördeléssel

- Az R relációt az előbbi algoritmussal M-1 kosárba tördeljük. A tördelést az összes attribútum értéke alapján végezzük.
- A módszer akkor működik jól, ha ezek a kosarak már egyenként beférnek a memóriába, mert akkor mindegyikükre az egymenetes tördelés algoritmus használható.
- Ennek feltétele:

$$B(R) \leq M(M - 1).$$

- Ekkor a műveletigény: $3 * B(R)$, ha R nyalábolt. Pontosan úgy, mint a rendezés esetén.

Csoportosítás és összesítés tördeléssel

- $\gamma_L(R)$ esetén a **tördelőkulcs kizárólag L csoportosító attribútumaitól függ.**
- Ha a tördelés megtörtént, a csoportosítás és összesítés kosaranként elvégezhető a memóriában.
- Ugyanakkor **itt nem kell, hogy a teljes kosár beférjen a memóriába.** Hiszen itt elegendő egyetlen rekordot napra készen (aktuálisan) tartani minden egyes, a kosárban szereplő, különböző csoportosító attribútum értékkel rendelkező rekord számára.
- Tehát a **$B(R) \leq M^*(M-1)$** feltételt kielégítő relációnál sokkal nagyobbak is feldolgozhatóak esetenként ezzel a módszerrel.
- Másrészt, ha **$M-1$** nagyobb a csoportok számánál, akkor **a kevés kosarunk túl nagyá válhat.**
- A **$B(R) \leq M^*(M-1)$** feltétel tehát inkább csak közelítés. A műveletigény viszont továbbra is **$3*B(R)$** , ha R **nyalábolt.**

Halmazműveletek tördeléssel

- Tördeljük R-et is S-et M-1 kosárba. Ügyeljünk arra, hogy **ugyanazt a tördelőfüggvényt alkalmazzuk** mind a két esetben. Az összes attribútum együtt alkotja tördelőkulcsot.
- A kosarakat jelölje: $R_1, \dots, R_{M-1}, S_1, \dots, S_{M-1}$.
- Hajtsuk végre a megfelelő halmaz-, multihalmaz műveletet R_i és S_i kosárpárokra ($1 \leq i \leq M-1$).
- **Műveletigény** (I/O): $3 \cdot (B(R) + B(S))$.
- Hogy az egymenetes algoritmus használhassuk kell, hogy **a kisebbik operandus beférjen M-1 pufferbe**. Vagyis az **alkalmazhatóság** feltétele:

$$\min(B(R), B(S)) \leq M(M - 1).$$

Összekapcsolás tördeléssel

- $R(X, Y)$ és $S(Y, Z)$ összekapcsolásánál **csak az Y attribútumait kell használni tördelőkulcsként.**
- Ekkor biztosak lehetünk benne, hogy ha R és S egy-egy sora összekapcsolódik, akkor R_i és S_i kosarakba kerülnek.
- Így **az összekapcsolást ismételten végrehajthatjuk kosaranként.**
- Műveletigény (nem meglepő módon): $3 \cdot (B(R) + B(S))$.
- Az alkalmazhatóság egy szükséges feltétele:

$$\min(B(R), B(S)) \leq M(M - 1).$$

I/O műveletek megtakarítása I.

- Tegyük fel, hogy R és S összekapcsolására k kosarat kell létrehozni, ahol k sokkal kisebb M -nél. Tegyük fel emellett még, hogy S a kisebb reláció.
- Hibrid tördeléses összekapcsolás:
 - Amikor S relációt tördeljük a k kosár közül m -et teljesen a memóriában tartunk, a fennmaradó $k-m$ kosarak létrehozásához pedig egy-egy (memóriában elhelyezkedő) blokkot használunk.
 - R tördelésénél szintén csak ezt a $k-m$ kosarat használjuk, hiszen ha egy t sor olyan kosárba esne, melynek S-beli megfelelője a memóriában van, akkor t azonnal összekapcsolható e kosár megfelelő elemeivel.
- Így kevesebb I/O műveletet kell végrehajtanunk.
- A következő megszorításnak kell teljesülnie.

$$\frac{mB(S)}{k} + k - m \leq M, \quad \text{hiszen egy-egy kosár mérete } B(S)/k.$$

I/O műveletek megtakarítása II.

- S memóriában tartott kosarait természetesen hatékony struktúrába kell szervezni.
- A megtakarítás így:

$2(m/k)(B(R) + B(S))$, mivel a kosarak m/k hányada van a Memóriában, S és R kosarak blokkjainak száma szor 2.
(ki/be)

- Az m/k hányadost szeretnénk maximalizálni.
- Optimális megoldás, ha $m=1$, a kosarak viszont minél nagyobbak, azaz k a lehető legkisebb.
- Egy kosár legnagyobb lehetséges mérete M , azaz kis egyszerűsítéssel $k = B(S)/M$.
- Így csak 1 kosár számára van hely, tehát $m=1$. (Egy kosár valójában nem is lehet M méretű, hiszen a bent tartott kosár mellett még $k-1$ szabad puffernek rendelkezésre kell állnia.)

I/O műveletek megtakarítása III.

- Ezekkel az egyszerűsítő feltételezésekkel a **megtakarítás**:

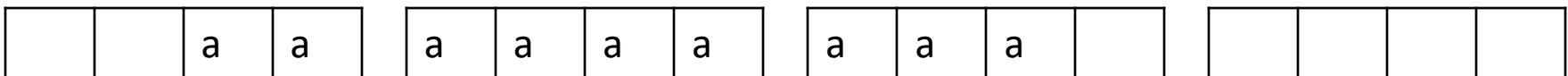
$$\frac{2M}{B(S)} (B(R) + B(S)).$$

- Így a **teljes költség**:

$$(3 - \frac{2M}{B(S)}) (B(R) + B(S)).$$

Index alapú algoritmusok

- Egy index **nyalábolt** (**clustered**), ha a keresési kulcs egy rögzített értékéhez tartozó adattábla sorok nagyjából annyi helyen helyezkednek el, ahány blokkban minimálisan elférnének.
- **Megjegyzés:** emlékezzünk vissza, szó volt már nyalábolt fájlservezésről és nyalábolt relációkról is.
- Nem nyalábolt relációra **nem hozhatunk létre** nyalábolt indexet, míg egy nyalábolt relációnak **lehet** nem nyalábolt indexe. **Miért?**



az összes **a** értékű sor az adatfájlban

Index alapú kiválasztás

- $\sigma_{A=a}(R)$ kiszámításánál, ha az A attribútumra vonatkozó index **nyalábolt**, egyenletes értékeloszlást feltételezve: $B(R)/V(R, a)$ I/O művelet szükséges a lekérdezés végrehajtásához.
- A gyakorlatban ez **egy "picit" több lehet**, mivel:
 - gyakran nem a teljes indexet tároljuk a memóriában.
 - Lehet, hogy az $A=a$ értékű sorok beleférnének **b** blokkba, mégis **b+1** blokkban tárolódnak, mert nem valamelyik blokk elején kezdődnek.
 - A blokkok csak félig vannak megtöltve helyet hagyva R módosulásainak: ekkor **2b** helyen férnek el csak a sorok.
 - R lehet, hogy **nyalábolt fájlrendezésű**, ezért nem csak R sorai vannak egy-egy blokkban.
- Ha az index nem nyalábolt a szükséges I/O műveletek száma: $T(R)/V(R, a)$.

Összekapcsolás indexek használatával

- $R(X, Y)$ és $S(Y, Z)$ összekapcsolásánál tegyük fel, hogy létezik indexünk az Y attribútum(halmazra).
- A **legegyszerűbb módszer szerint** beolvashatjuk R minden blokkját, s egy-egy blokk minden y értékű t sorát az index segítségével összekapcsolhatjuk S y értékű soraival.
- Itt a szükséges I/O műveletek száma, ha az index **nyalábolt**, illetve **nem nyalábolt**:

$$T(R) \left(\max \left(1, \frac{B(S)}{V(S, Y)} \right) \right), T(R) \frac{T(S)}{V(S, Y)}.$$

- Itt $B(S)/V(S, Y)$ lehet, hogy kisebb mint 1 , 1 blokkbeolvasás viszont mindenképp szükséges.

Mikor hasznos az előbbi algoritmus? I.

- Bár az előbbi algoritmus **nem tűnik túlságosan hatékonynak**, mégis, ha **R kicsi** **V(S, Y) pedig nagy**, sokszor ezt éri meg alkalmazni.
- **Példa:** a **Szerepel (filmcím, év, név)**,
Színész (név, kor, fizetés, születés) relációkon a:

```
SELECT név As színésznév  
FROM Szerepel Sze, Színész Szí  
WHERE Sze.név = Szí.név AND filmcím = 'Star Wars' AND  
      év = 1978;
```

lekérdezés.

Mikor hasznos az előbbi algoritmus? II.

- A lekérdezéshez tartozó relációs algebrai kifejezés:

$$\Pi_{\text{színésznev}} ((\sigma_{\text{filmcím}='Star Wars' \wedge \text{év}=1978} (\text{Sze})) \bowtie (\text{Szí})).$$

Összekapcsolás rendezéssel és indexszel

- Tegyük fel továbbra is, hogy $R(X, Y)$, $S(Y, Z)$ relációkat szeretnénk összekapcsolni, és S Y attribútum(halmaz)ára már létrehoztunk egy indexet.
- Ekkor elkészíthetjük R rendezett listáit a korábban látottakhoz hasonlóan, majd beolvashatjuk e listák első blokkjait $M-1$ helyre.
- Aztán újra és újra a legkisebb y értékeket vizsgálva az indexet (és a fennmaradó egy szabad puffert) használva S y értékű sorait összekapcsolhatjuk R y értékű soraival.
- A korábbi rendezéses megvalósításhoz képest itt:
 - nem kellett rendeznünk S sorait.
 - Ráadásul R y értékű sorainak tárolására $M-1$ puffer áll a rendelkezésre, tehát ritkán fordulhat elő, hogy beágyazott ciklusú összekapcsolásra van szükség, mert az összes ilyen R -beli sor befér a memóriába.
- A műveletigény így (durván): $3 * B(R) + B(S)$ (az eddigi legjobb!).

Cikk-cakk összekapcsolás

- Ha R Y attribútum(halmaz)ára is létrehoztunk már indexet, akkor R rendezésére sincsen szükség, csak az indexeket használva, kisebb értékektől a nagyobbak felé haladva kapcsoljuk össze a sorokat.
- A B-fáink vannak, akkor balról-jobbra haladva olvashatjuk a leveleket.
- Ráadásul, ha R-nek nincs y értékű sora, S-nek viszont van, ez már az indexek használatával kideríthető, így S y értékű sorait sem kell beolvasni. Itt nyilván R és S szerepe fölcserélhető.
- Ha az indexblokk beolvasások számát elhanyagoljuk (ahogy ezt az előbb is tettük) az I/O műveletek száma: $B(R) + B(S)$ lesz nyalábolt indexek esetén.