DW
7. előadás
ETL-folyamatok, fizikai struktúra

ETL-témakörök

- Dimenziók kezelése:
  - a módosult adatok kiemelése az OLTP-ből,
  - változó dimenziók kezelése,
  - duplikáció megszüntetése és közelítő keresés.
- Tényadatok kezelése:
  - tényadatok kiemelése az OLTP-ből,
  - tényadat változtatások kezelése,
  - tényadatok purgálása,
  - adatminőség biztosítása,
  - tényadatok transzformációja,
  - surrogate key kezelése,
  - adatok betöltése,
  - OLAP elemzések végzése.
- Adat integráció kezelése:
  - kulcs generálása,
  - érték ellenőrzése,
  - illesztése.

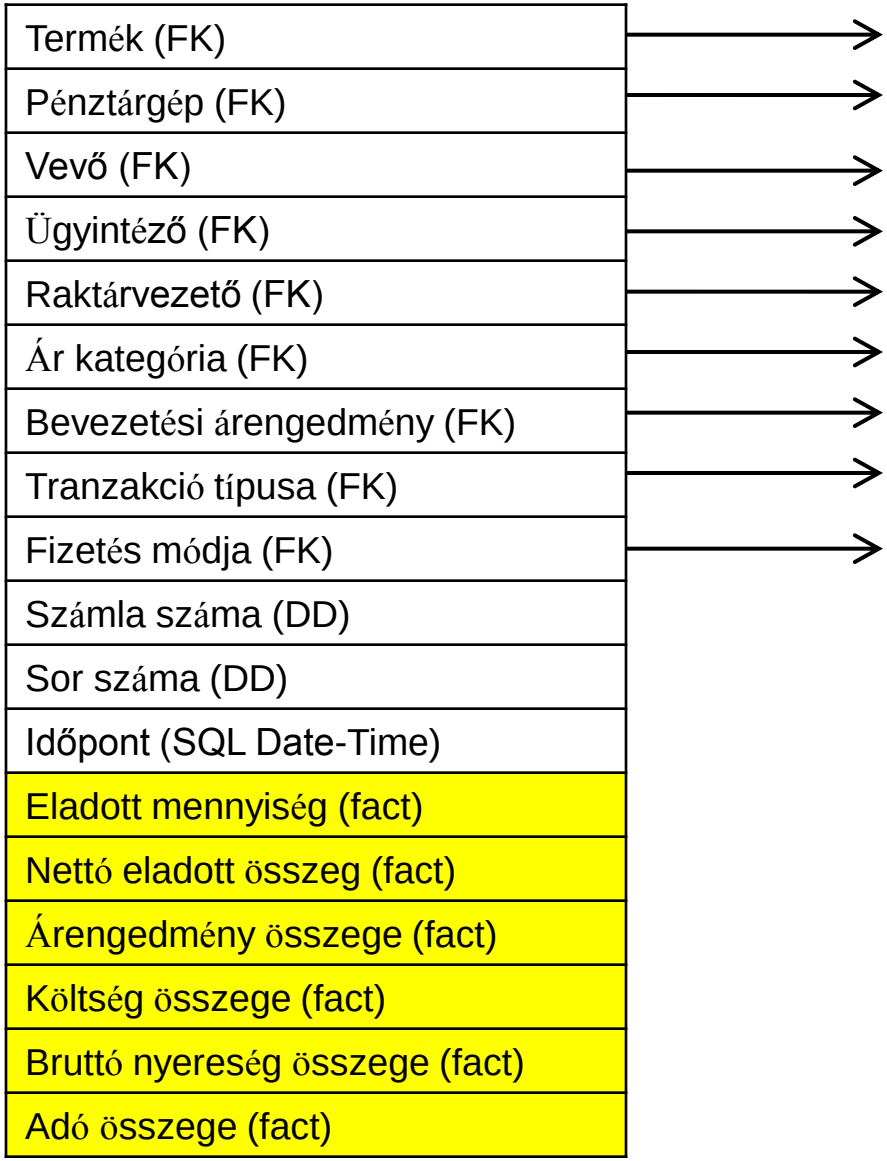
ETL-folyamatok automatizálása

Az ETL-rendszer alapvetően automatikus működésű:
– reggeli adatgyűjtés,
– hétvégi adatgyűjtés,
– adatmentések,
– adat purgálása,
– jelentések készítése.

Metaadatok kezelése:
- metaadat is adat.

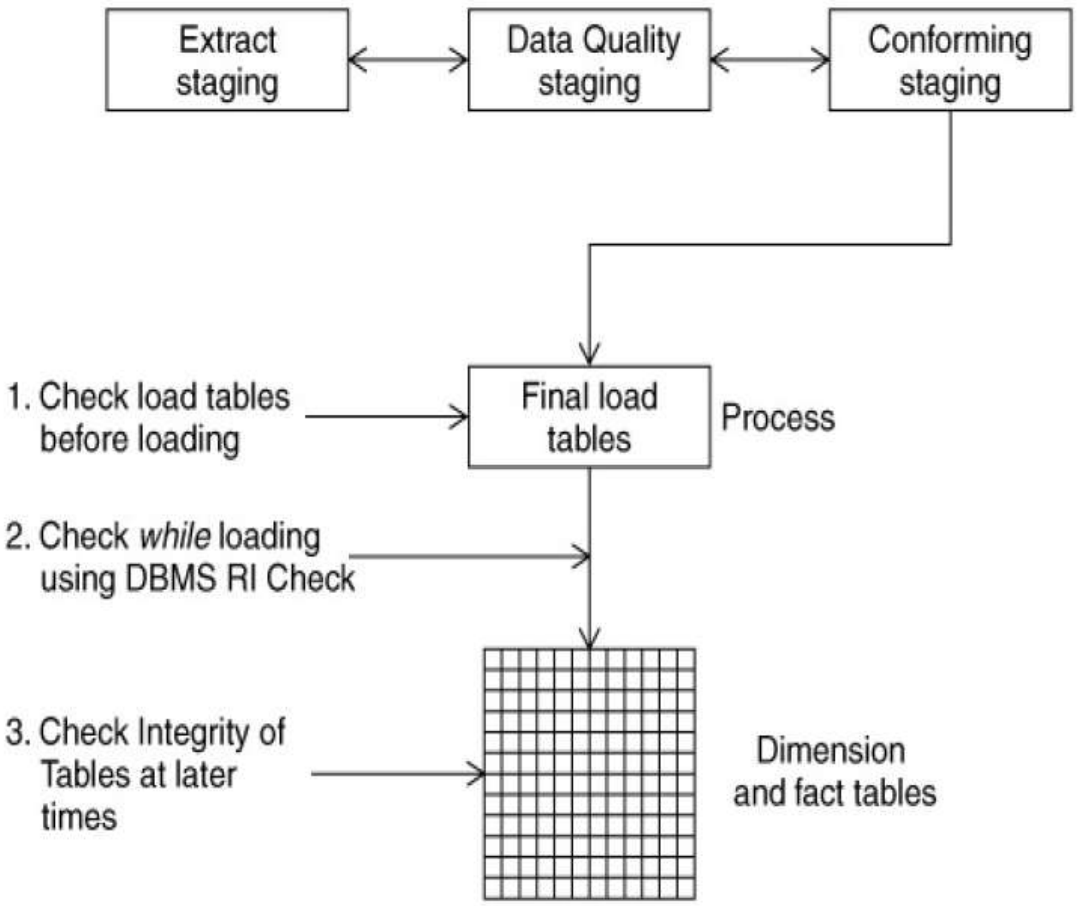
Forrás ténytábla elemzése

Eladások-ténytábla



- Komplexicitás:
  - tárolt információk,
  - tárolt kapcsolatok,
  - mezők darabszáma.
- Idegenkulcsok:
  - kapcsolt dimenziók,
  - független mennyiségek.
- Degenerált dimenziók:
  - a FK-ban kódolt,
  - nincs külön kódtábla.
- Idő dimenziók:
  - nem FK jelleg.
- Elsődleges kulcs:
  - az FK-k részhalmaza,
  - lehet időjellegű.
- Fact mezők:
  - változók.

Hivatkozások épségének ellenőrzése



1. Betöltés előtti ellenőrzés:
- új rekord ellenőrzése,
- rekord módosítás, törlés előtt,
- ajánlott módszer.

2. Betöltés alatti ellenőrzés:
- DBMS ellenőrzi a RI-t (Referential Integrity),
- egyszerű, de rendszerint lassú,
- vannak kivételek: Red Brick, DBMS igen gyors.

3. Betöltés utáni ellenőrzés:
- ha nincs RI támogatás a DBMS-ben,
- időszakosan, periodikusan kell ellenőrizni,
- igen lassú,
- lefogja a DW erőforrásait.

Surrogate Key kezelése

Surrogate Key: mesterséges azonosító
- belső azonosító,
- a verziók megkülönböztetése
(kívül nem látszik).

- A betöltés alatt, az ETL motor egyik eleme, a természetes PK értékeket a dimenzióknál átkonvertálja Surrogate Key-re.
  – leképzési táblát használ

Surrogate Key kezelés problémái

- A dimenzió adatok időben változhatnak.
- Érvényességi tartományt rendelik az értékekhez.
- Időbeliség követése:
  - verziók nyilvántartása,
  - utolsó verzió használata,
  - külön dimenzióérték használata.
- A tény és dimenzió adatok aszinkron is érkezhetnek.
- A Lookup tábla nem mindig használható.
- A dimenzióban tárolódik az érvényesség.
- Dimenzió verziók nyilvántartása szükséges.

Ténytábla típusok

Ténytábla adatok három típusa:

- Tranzakció szintű kezelés
  - Transaction Log áthozatala,
  - részletes adatok,
  - probléma lehet a sparse adatok kezelése.

- Időszakos Snapshot
  - ütemezett aggregált áttöltés,
  - jó helykihasználás, de kevesebb információ.

- Aggregált snapshot
  - korábbi változások együttes hatását hozza át.

Indexek kezelése

Indexek hasznos segédeszközök lekérdezéskor,
de a DML során teljesítménykorlátozást okoznak.

1. INSERT és UPDATE műveletek élesen külön kezelése.
2. Csak a módosítást támogató indexek maradjanak meg.
3. UPDATE elvégzése.
4. Indexek törlése.
5. INSERT elvégzése (Bulk mód – nagy mennyiség –).
6. Indexek újraépítése.

Tábla betöltése

- Insert és Update parancsok szeparálása:
  - először UPDATE,
  - utána INSERT.
- Bulk-betöltőt használjunk!
  - Redukált overhead (többletmunka, -terhelés)
- Párhuzamosítás használata:
  - adatok logikai szegmensekre bontása.
- Minimalizáljuk az UPDATE-ek számát!
  - Jobb törölni és újra felvinni.
- Ne a RDBMS-ben végezzünk aggregálást!
  - SQL aggregáció lassú.
- Sok UPDATE esetén a teljes táblát újratölthetjük.

Tényadatok módosítása

- Rendszerint nem kellene,
  de hibás adatbevitel esetén szükséges.
- Alternatívák:
  - tényadat stornózása:
    - tényadat megismétlése ellentétes előjellel,
    - változtatás naplózása,
  - tényadat módosítása,
  - tényadat törlése és betöltése újra:
    - lehet logikai és fizikai törlése,
    - konzisztencia sérülés problémája.

DW fizikai struktúra hatékonysága

A MD műveletek időigényesek, költségesek:

- fold,
- slice and dice,
- roll up.

Költségkritikus elemek:

- ténycellák megkeresése,
- aggregáció elvégzése.

Specifikus megoldások:

- több-dimenziós indexek,
- pointerláncok,
- elszámítások (MV - materialized view).

Hatékonysági kérdések

Időigényes műveletek (ahol lehet javítani):

$\sigma_{f(v)}(C)$, $\sigma_{f(D.a)}(C)$, $\nu_{D.a}(C)$, $\phi_{D,\ aggr}(C)$

(és ahol nem…)

$C_1 \times C_2$

Adott kulcsú cellák megkeresése

A: szekvenciális keresés ?
reménytelen !!

k1	k2	k3	v1	v2	v3

B: index alapú keresés ?
k1+k2+k3 : ha mindegyik kulcs adott
költség : log (N^3) ez OK !!

de asszimnetrikus !

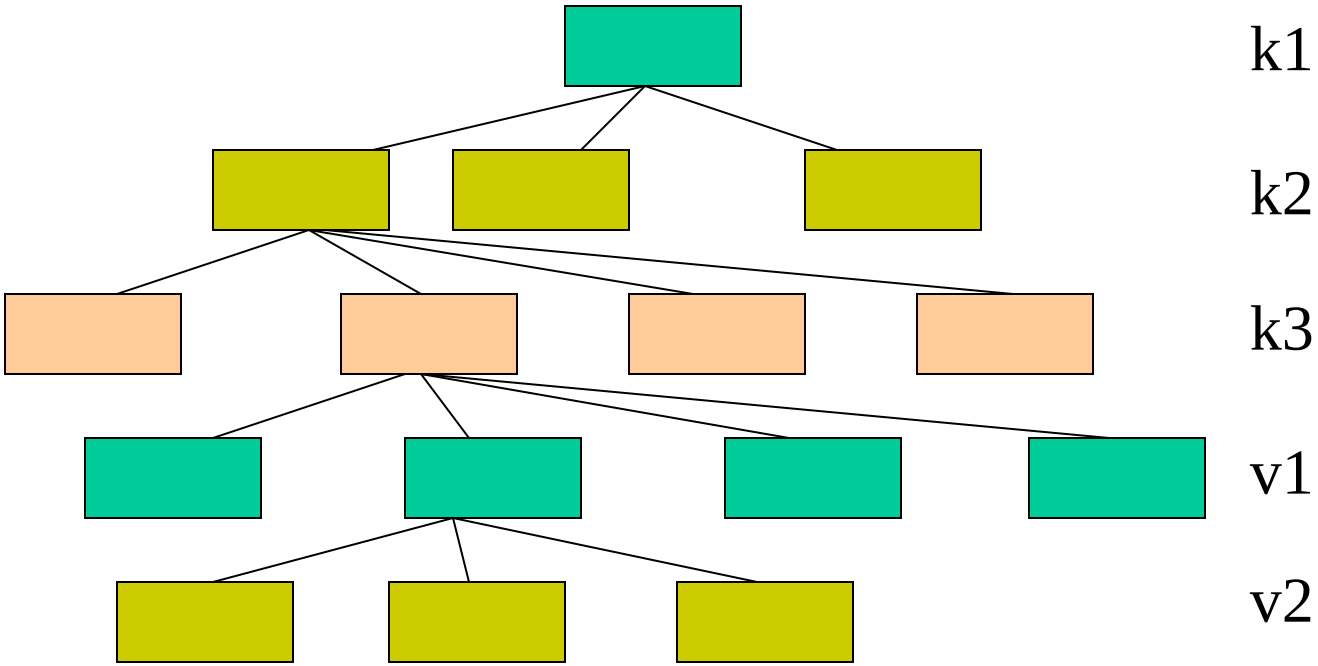
Hatékonysági kérdések

KD-indexfa

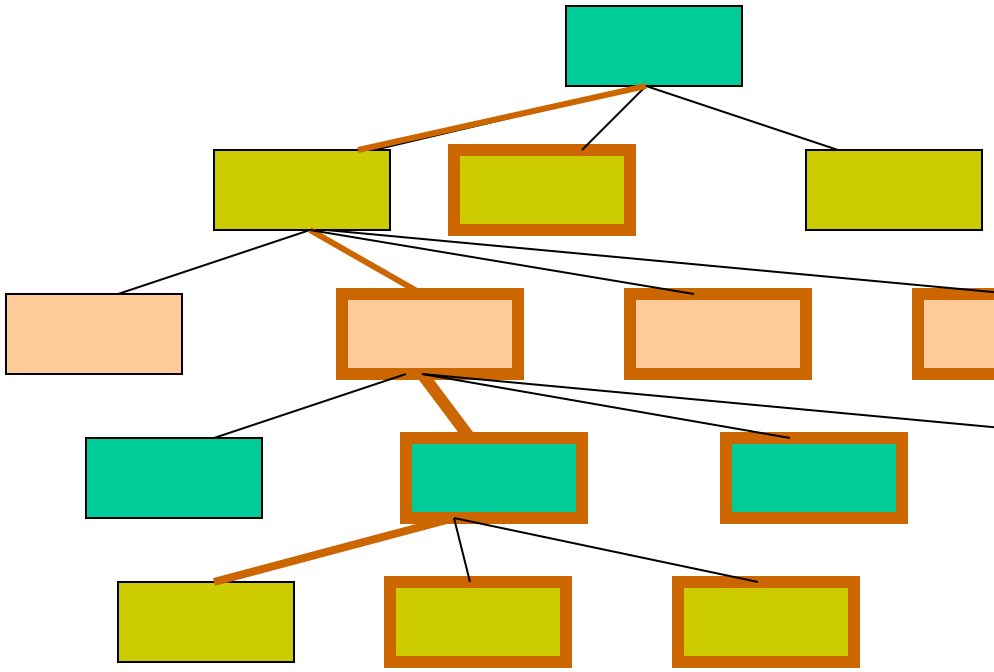
(k-dimenziós bináris fa)

k1	k2	k3	v1	v2	v3

Több-dimenziós keresőfa :
minden szint egy-egy dimenzióhoz rendelt ciklikusan



Hatékonysági kérdések



ha valamelyik kulcs tetszőleges (intervallum), akkor több ágon fut a keresés

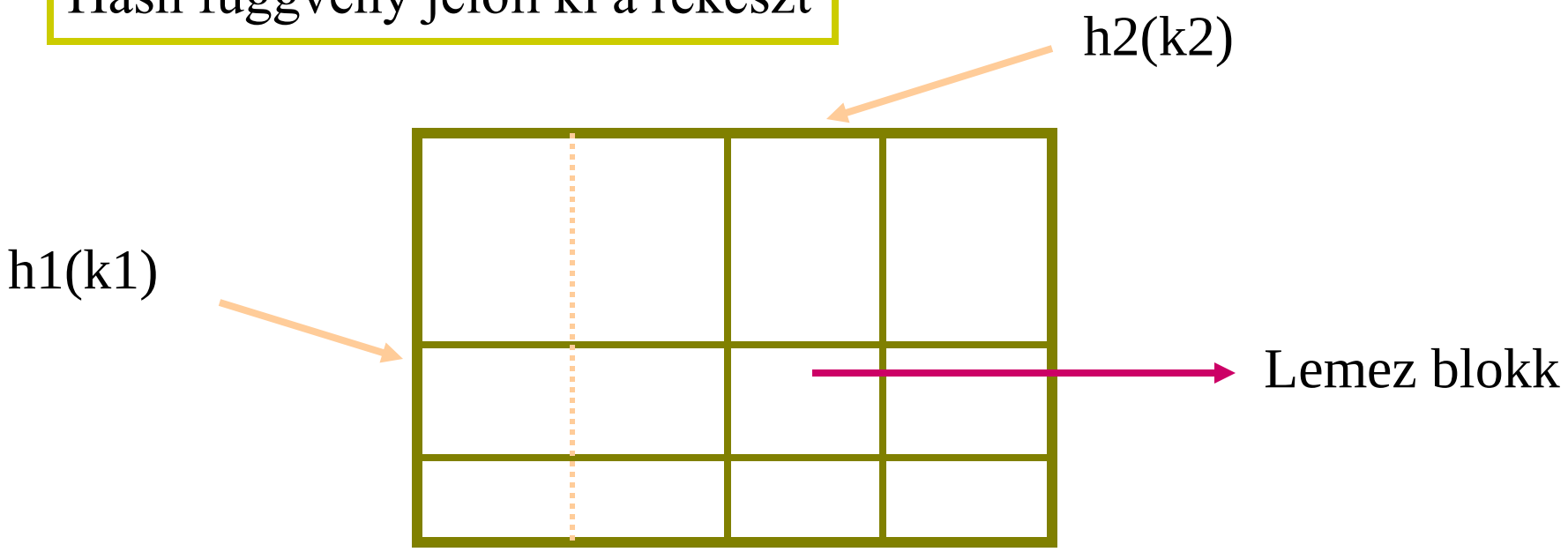
Ha keresési feltétel: k1 = %, k2=x, k3=%
csomópontok száma: L, L^2, L^3 (L a B-fa fokszáma)
költség: első k2 szinten: L, második k2 szinten: $L*L^{2,}$
i. k2 szinten: L^{2*i+1} ez is túl nagy még !!!

Grid-file, grid-struktúra

Hatékonysági kérdések

Szimmetrikus az egyes dimenziókra

Hash függvény jelöli ki a rekeszt



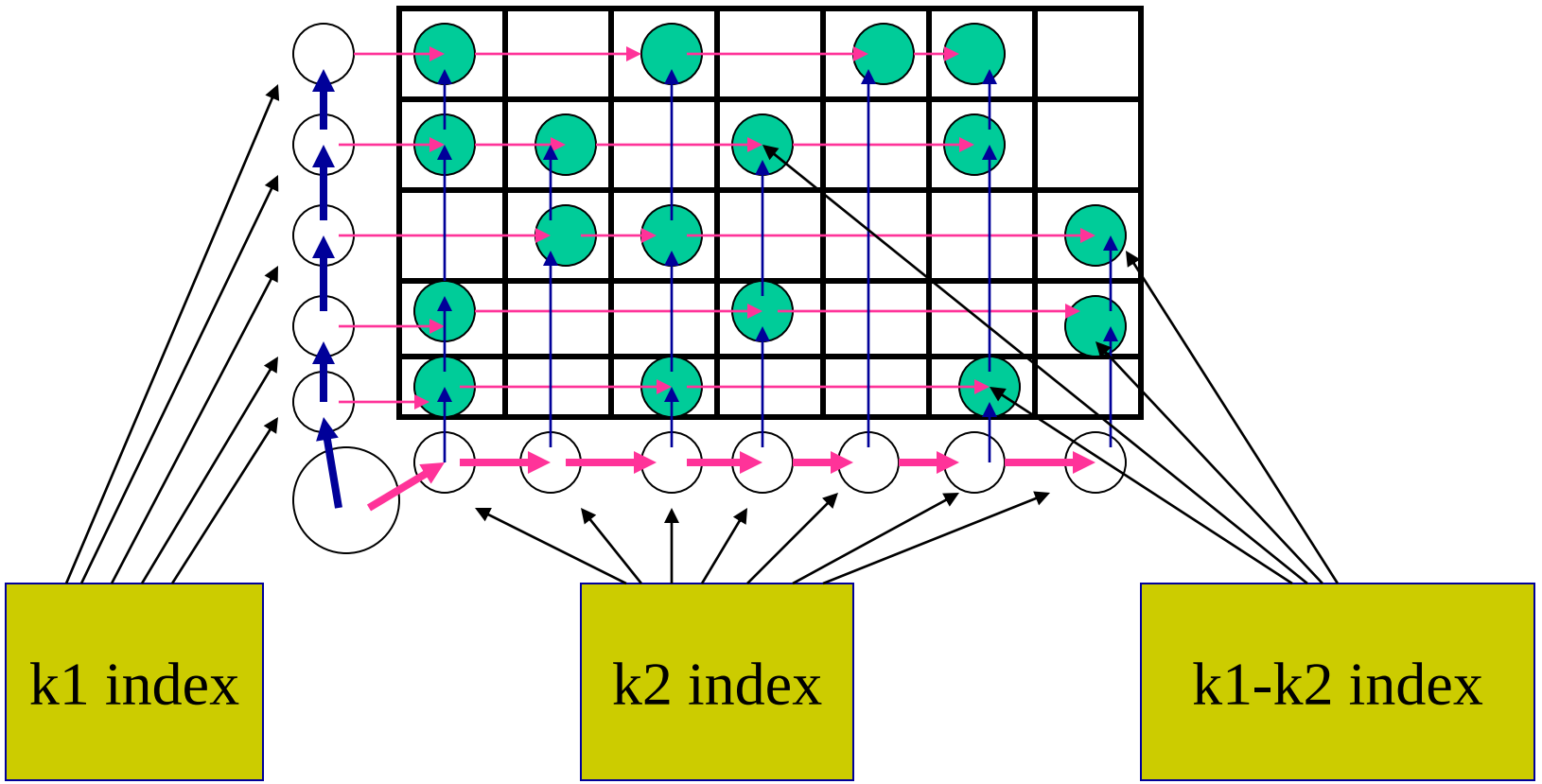
Ha túlcsordul, egyik dimenzió mentén felhasad

Hátránya : reláció-őrző hash függvény kellene

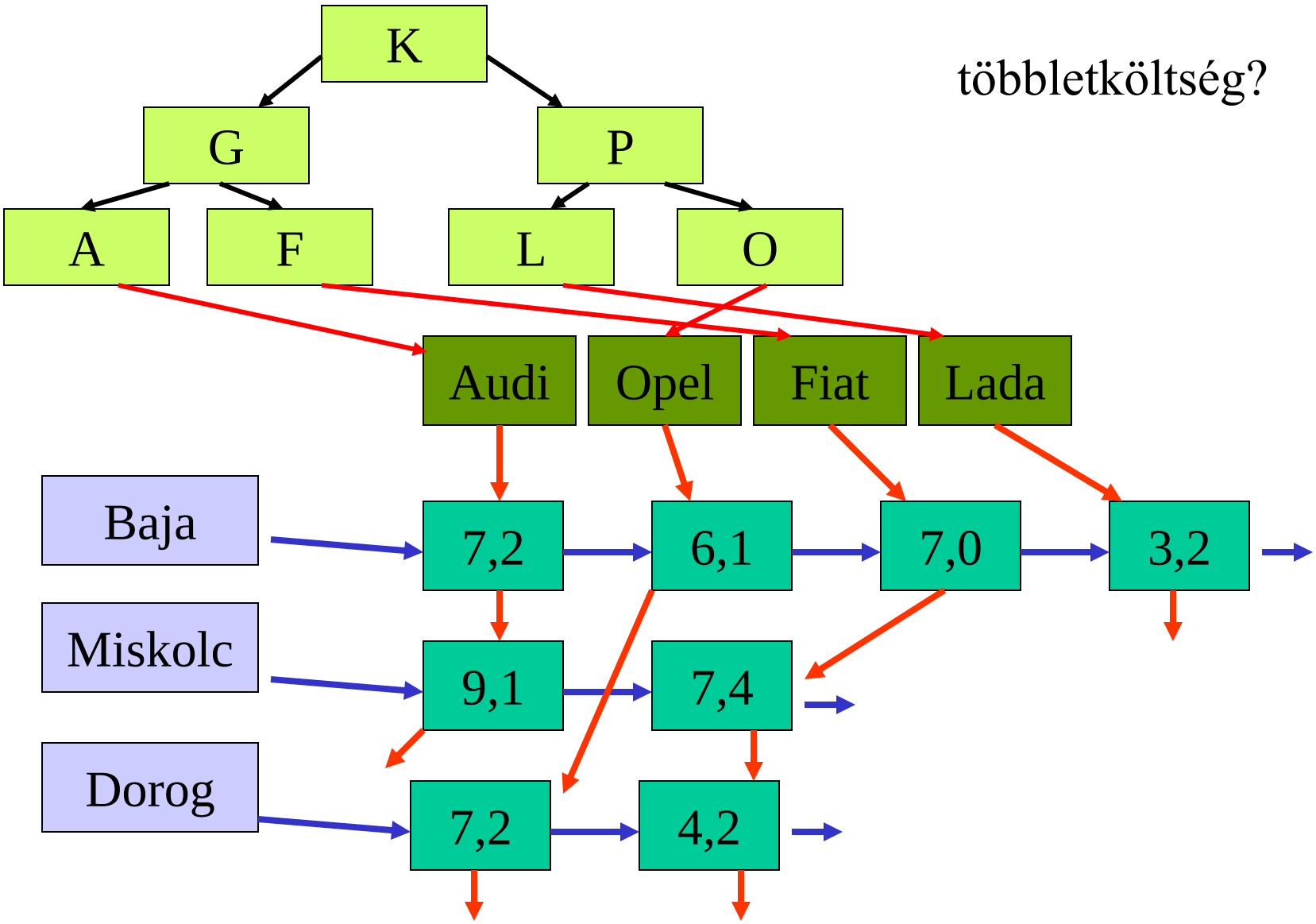
Pointer-lánc és index integrálása

Hatékonysági kérdések

Legbiztosabb megoldás: többszörös indexelés, többszintű pointer láncolat

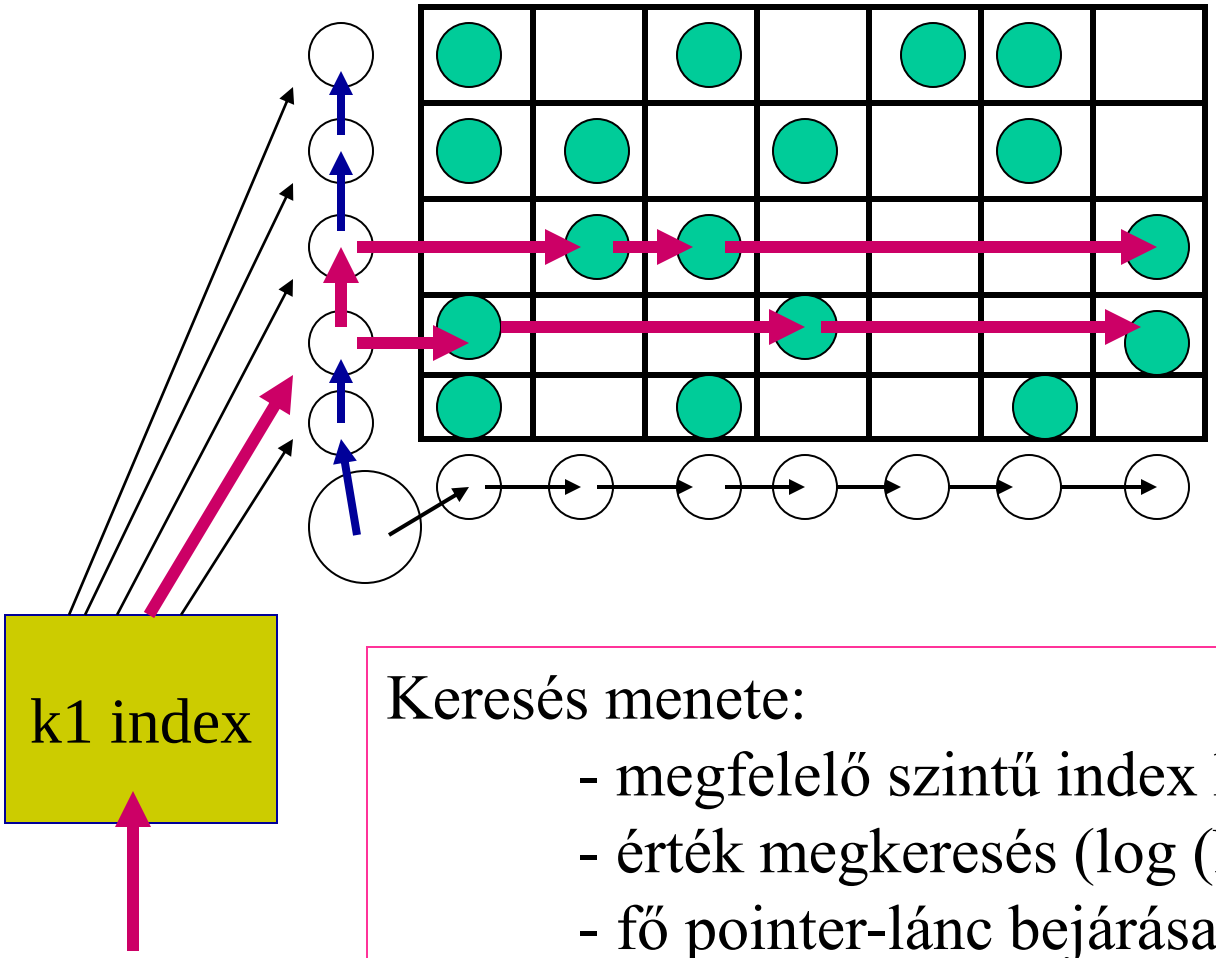


Fizikai megvalósítás



többletköltség?

Pointer-lánc és index integrálása



Keresés menete:

- megfelelő szintű index kiválasztása,
- érték megkeresés (log (N^k)),
- fő pointer-lánc bejárása,
- mellék pointer-láncok bejárása.

Csak a szükséges elemeket érinti.

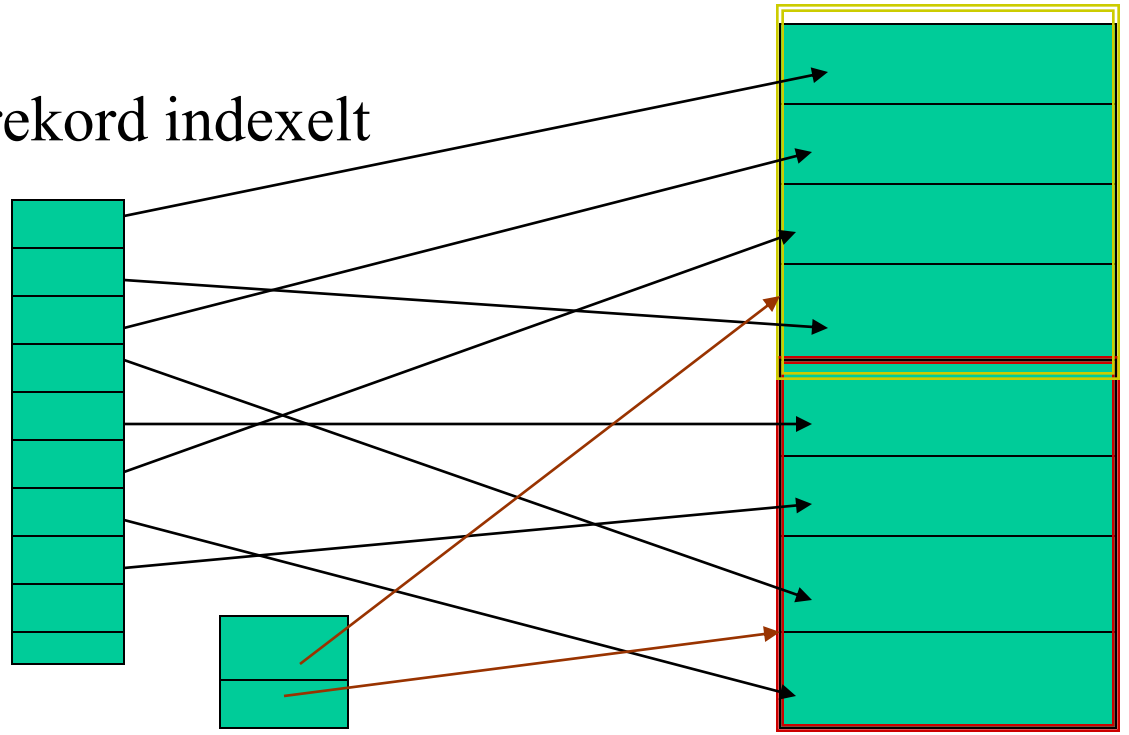
Indexelés: sparse - dense index / ritka és sűrű indexek

ritka index: nem minden rekord indexelt
(index szekvenciális)
feltétel: rendezettség
szélső elemre mutat
lsd: cluster index

sűrű index: minden rekord indexelt

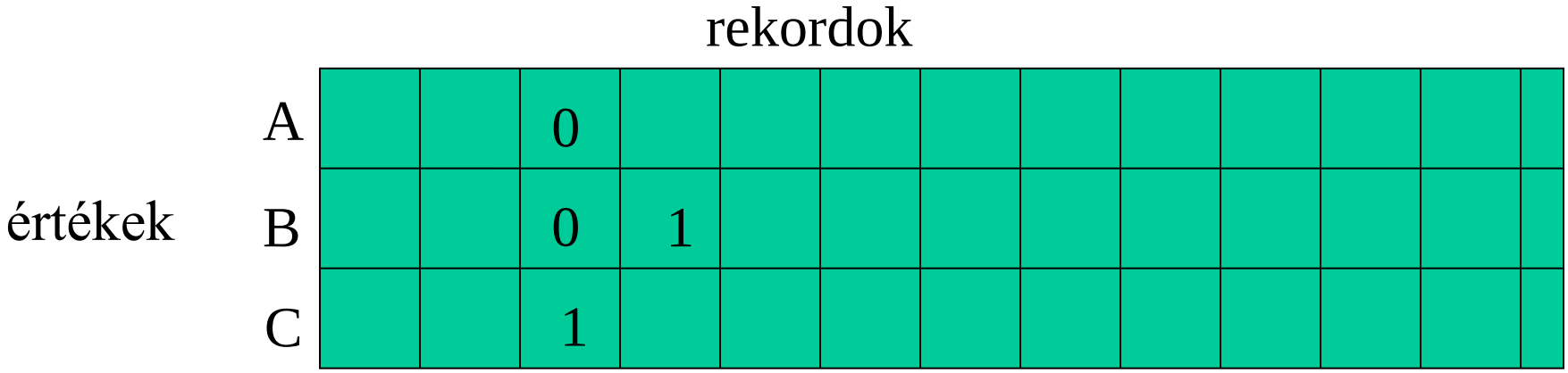
idő:
$\log_M(N/k) + [k]$
$= \log_M N - \log_M k + [k]$

hely:
N/k/M blokk

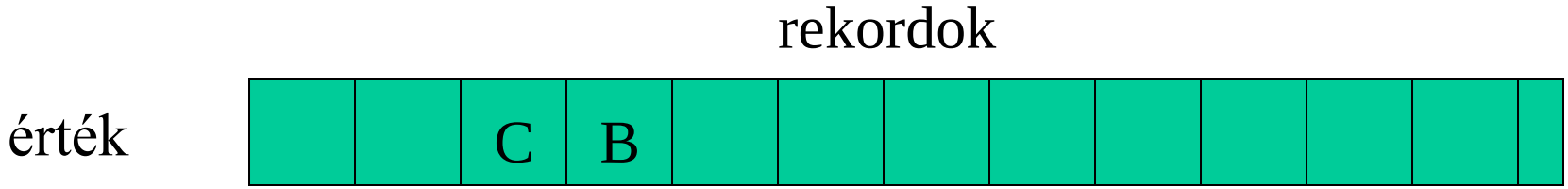


Indexelés: bitmap index

táblázatos tárolás (nincs rendezve)



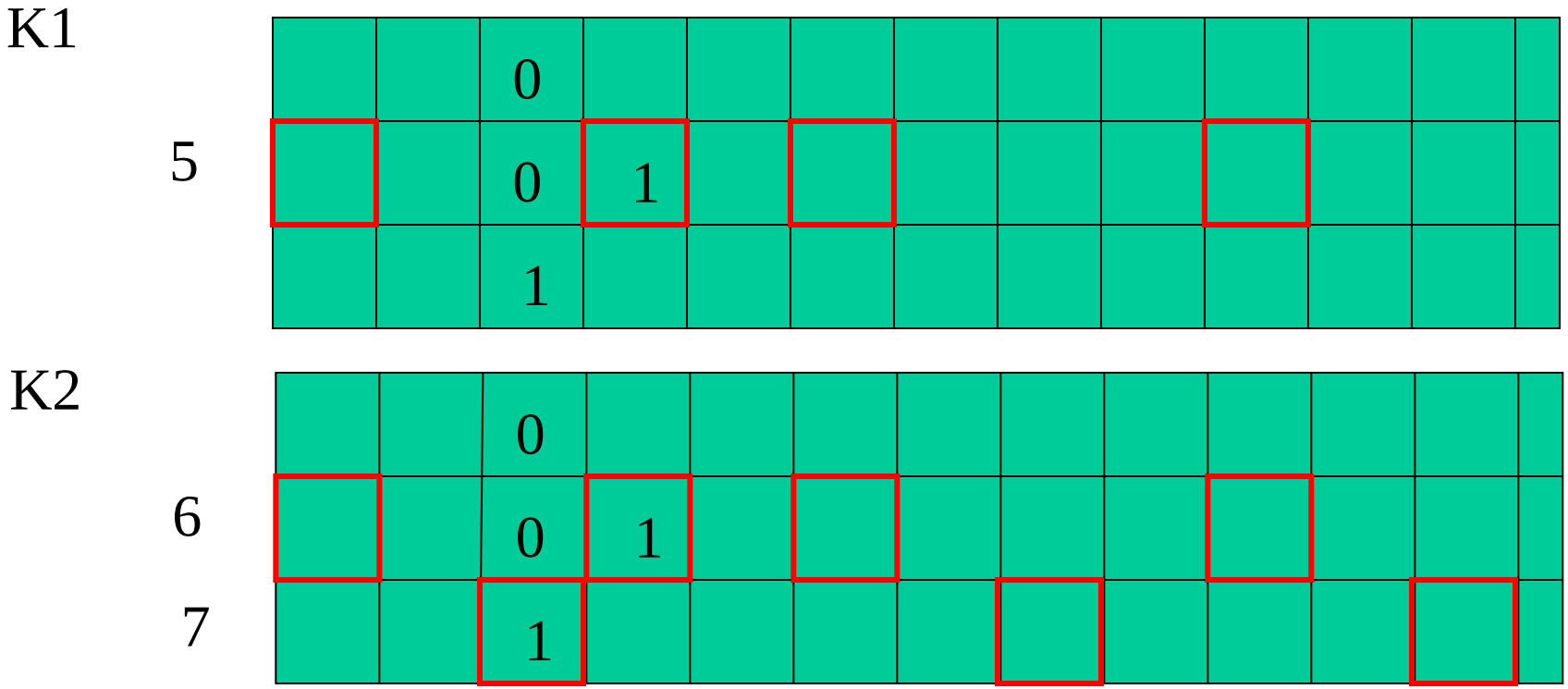
szűk domain esetére
támogatja a logikai operátorokat
gyakran társítják a projekciós index-szel



Összetett feltételek, több kulcs esetére is alkalmas a bitmap index

fetétel: K1 = 5 and K2 > 5

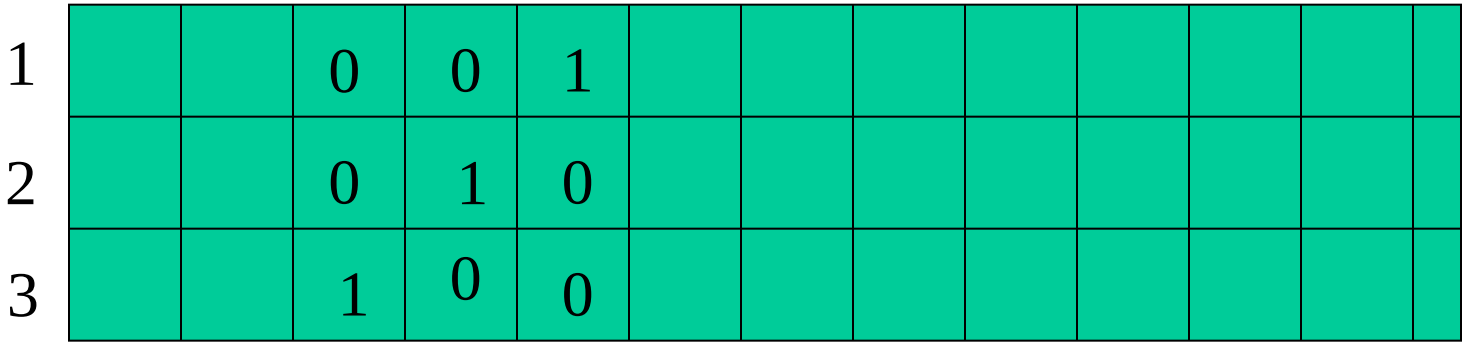
B-fa szerint: K1=5 megkeresése majd szekvencia a K2> 5 ellenőrzésre



+ metszetképzés / unió / komplementer

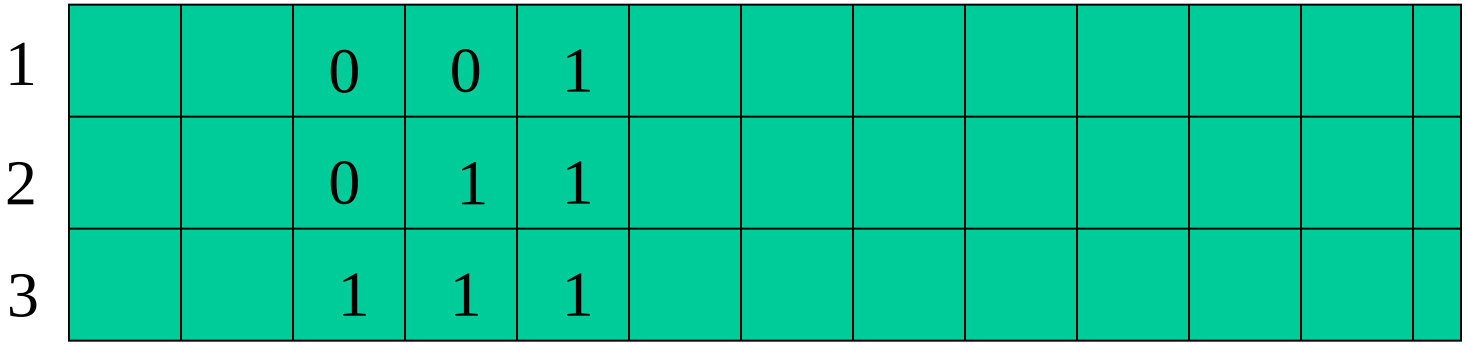
Bitindexek típusai

normál mód: minden érték egy sor



előnyös egzakt érték keresésnél

tartomány mód: minden érték több sor (a tőle nagyobb értékek)



előnyös tartomány érték keresésnél

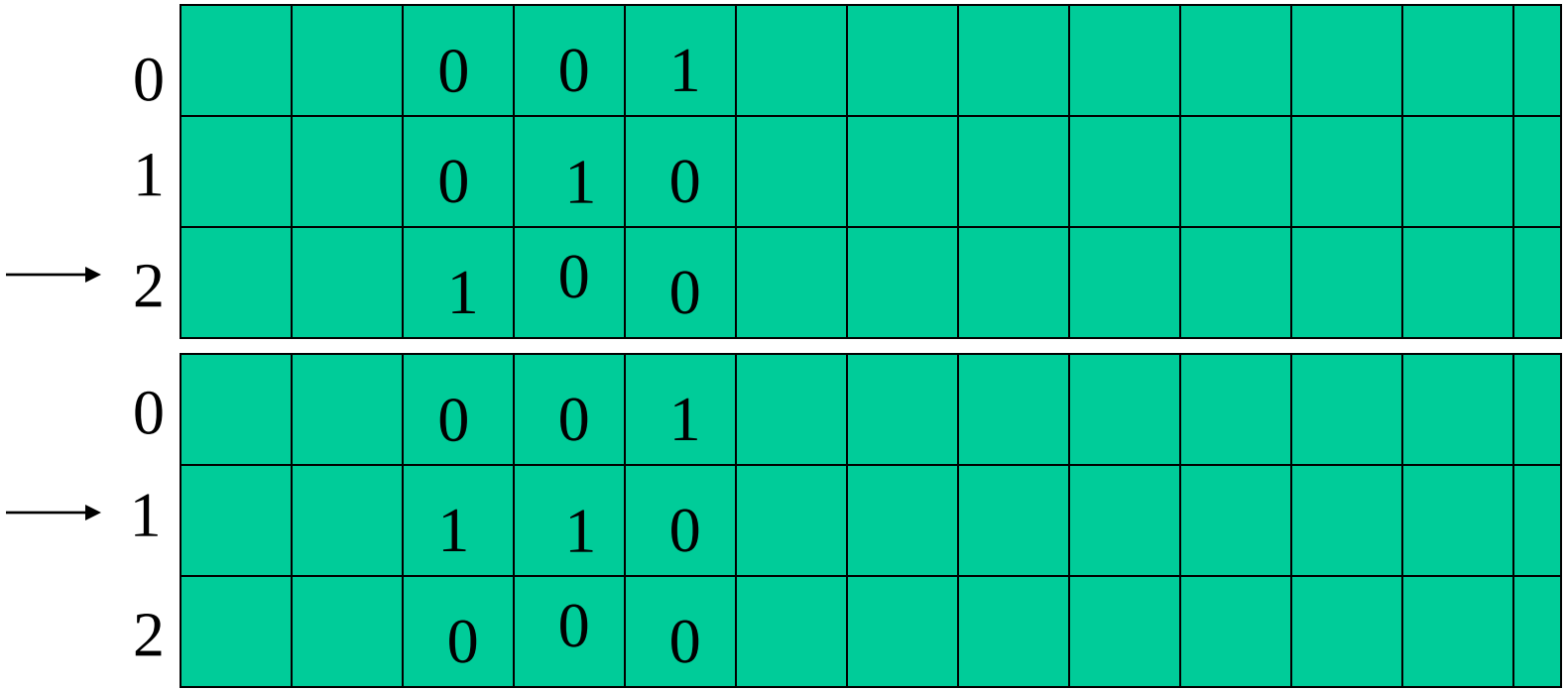
Bitindexek típusai

dekompozíciós mód: a tábla több résztáblára bontott

<r_n,…,r_2,r_1> szám n-es alapján képzi a bitmintát:

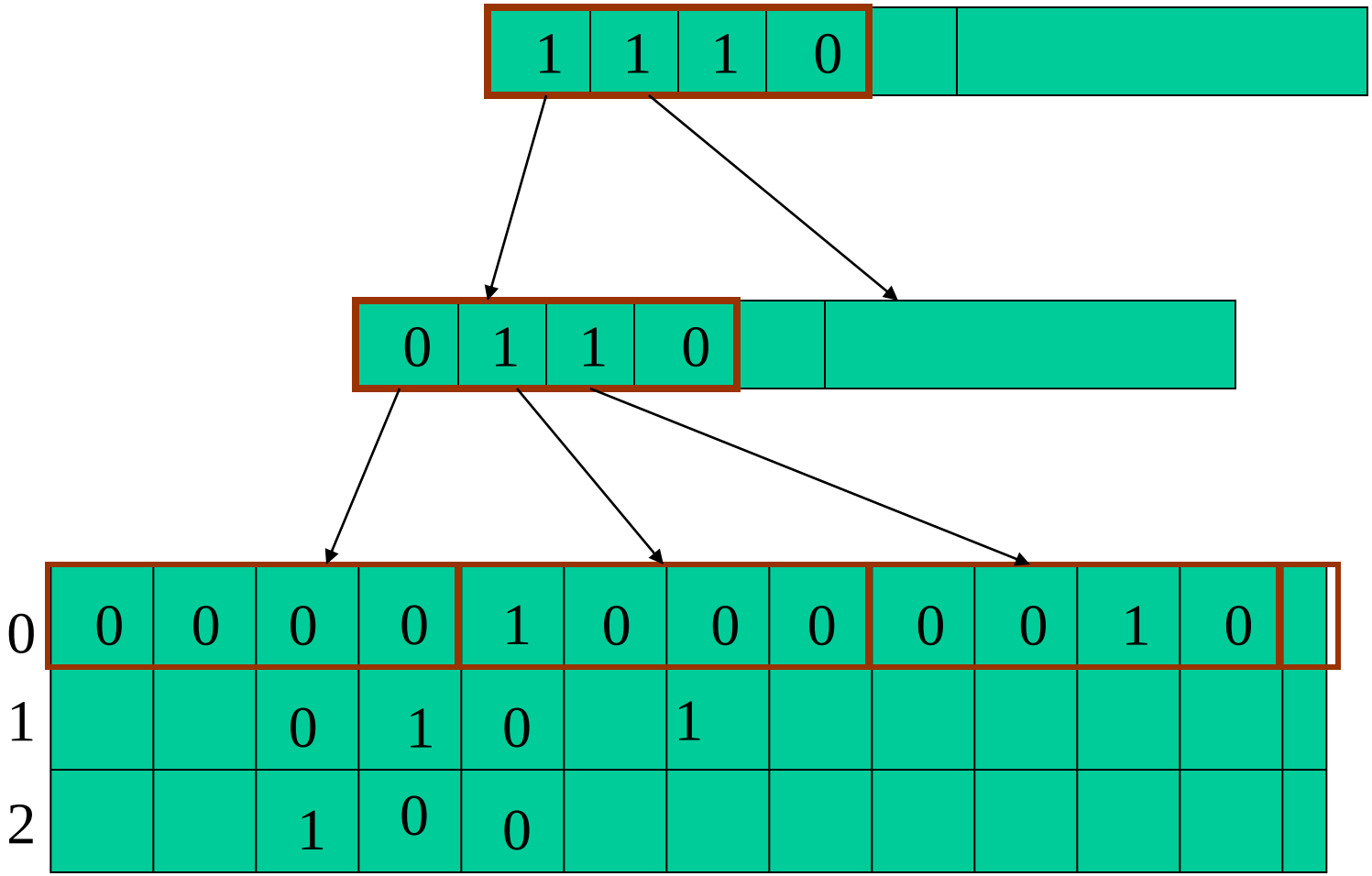
$$C = b_1 + b_2 * r_1 + \ldots + b_{n+1} * r_n \qquad r_n = D/szorzat_{n-1}(r_i)$$

Pl. n =1 , b_1 = 3 –hez: 5 = 1*3 + 2



Bitindexek típusai

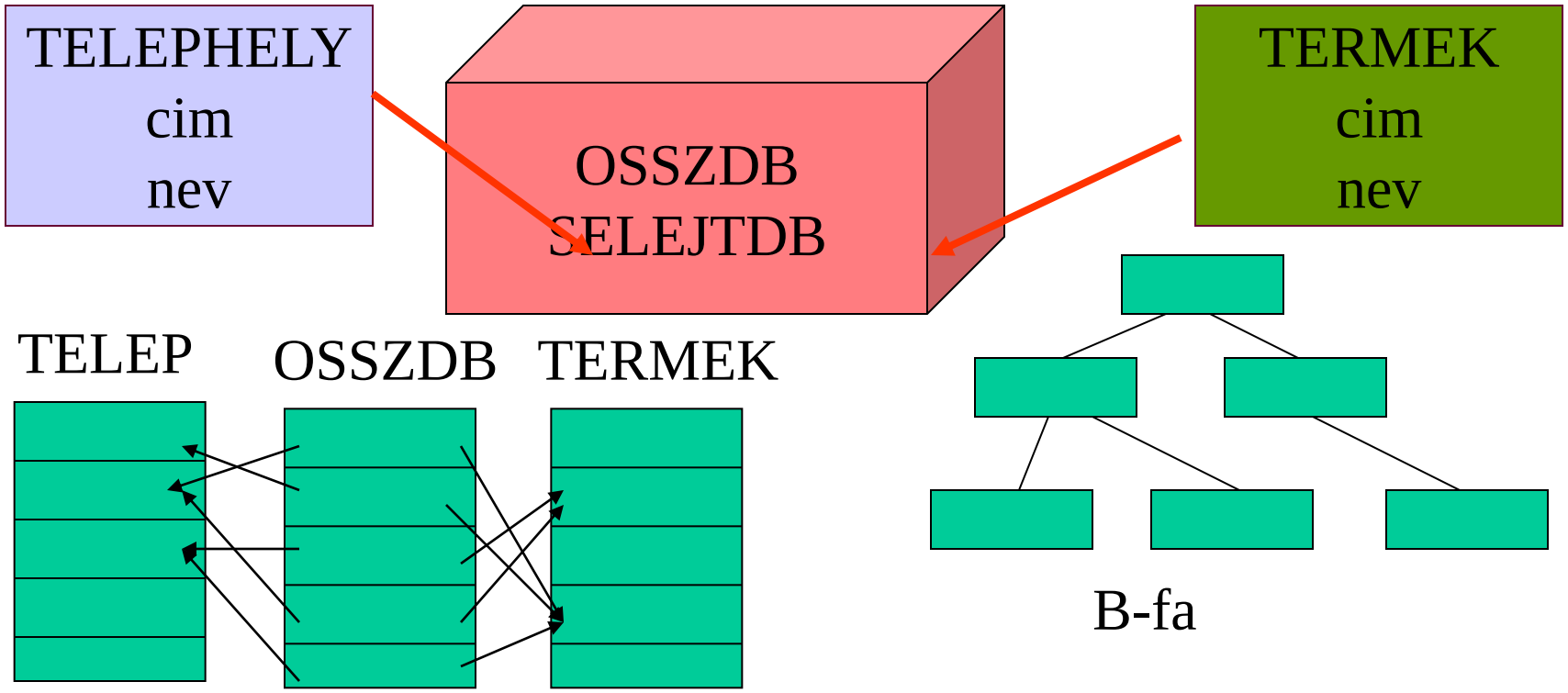
hierarchikus bitindex: a zéró helyeket felsőbb szinten jelzi



Join index

A kapcsolódó rekordpárok kijelölése kulcs szerint rendezve

tény-kulcshoz dimenzió-kulcsot vagy
dimenzió-kulcshoz tény-kulcsot rendel



Nem kell explicit keresést végezni

Cella keresési költségek, szelekciós költségek

módszer	teljes kulcs	részkulcs	intervallum
B-fa	$\log_M N$	1 fa: N k-fa: K' $\log_M N$ + K'*N' (metszetek)	$\log_M N+N/(2B)$
Grid-file	1	$H^{K-K'}$	$(H/2)^K$
bitmap	V*N/B'	V'*N/B'	V*N/2/B'

N :elemszám, M fokszám V : értékek száma, K' kiválasztott dim. db
B: blokkméret, H: bucketek száma, K:dimenziók száma

Aggregációs számítások

SELECT sum(fiz) FROM dolgozok WHERE beosztas = ‘irnok’

a feltételt teljesítő rekordok pozíciót ismertnek tekintjük (B_f)

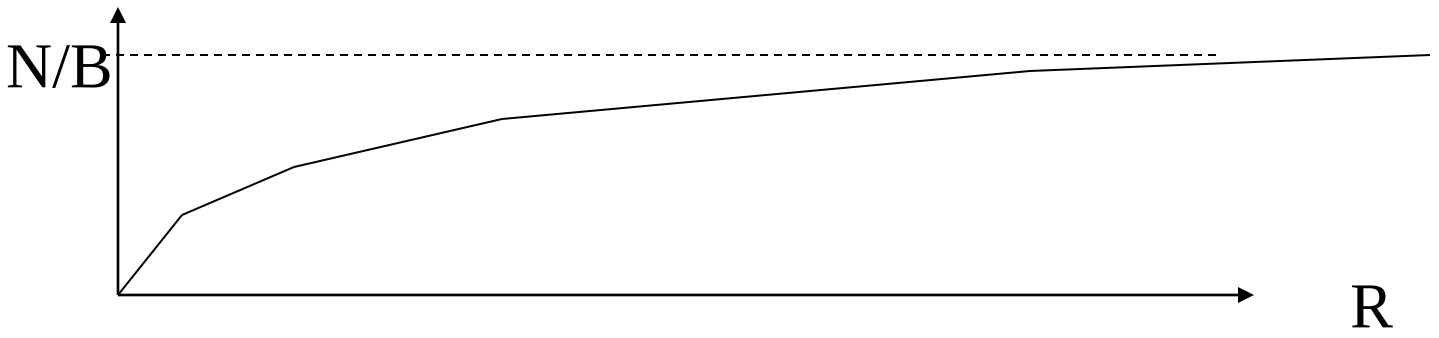
- alap módszer: a kijelölt rekordok közvetlen beolvasása

N: rekord db., B: blokk méret, R: eredmény rekord db.
érintendő blokkok száma: $O(N/B * (1 – e^{-RN/B}))$

nem egyszerű feladat

$a = szumma_B(B* P(B))$

$P(B) = ((N:B))szorzat_i((S:i))$: $1 <= i <= S$



Aggregációs számítások

- projekciós index módszer: a kijelölt mező értékeket a a projekciós indexből vesszük be.

előny: kevesebb blokk olvasás (B' >> B)

$$O(N/B' * (1 - e^{-RN/B'}))$$

- mező indexen keresztül (B'' >> B'):

```
sum = 0
foreach v in DOM(fiz)
{
  B_v = foundset az index alapján
  sum = sum + v*|B_f metszet B_v |
}
```

$$O(V*N/B'' + N/B')$$

Aggregációs számítások

A költség függ az aggregáció jellegétől

SELECT max(fiz) FROM dolgozok WHERE beosztas = ‘irnok’

- közvetlen elérés:

$$O(N/B * (1 – e^{-RN/B}))$$

- bitmap index az elemre (metszet a B_f-fel):

$$O(V*N/B'')$$

- projekciós index:

$$O(N/B' * (1 – e^{-RN/B'}))$$

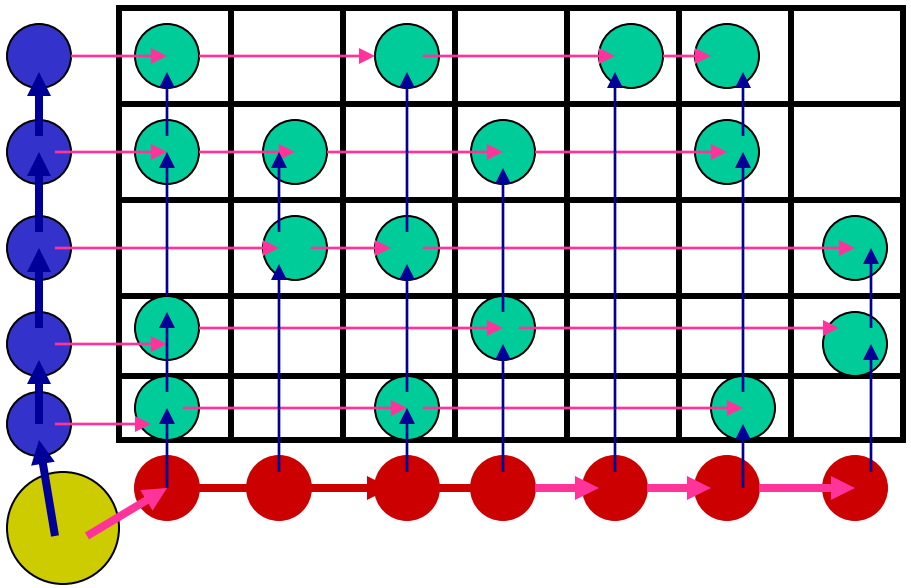
- mező index

$$O(V/2*N/B'' + N/B')$$

Aggregációs költségek

Hatékonysági kérdések

Az aggregáció gyorsítása: elő-aggregációk tárolása



elemi érték (k1,k2)

k1 szerinti aggregáció

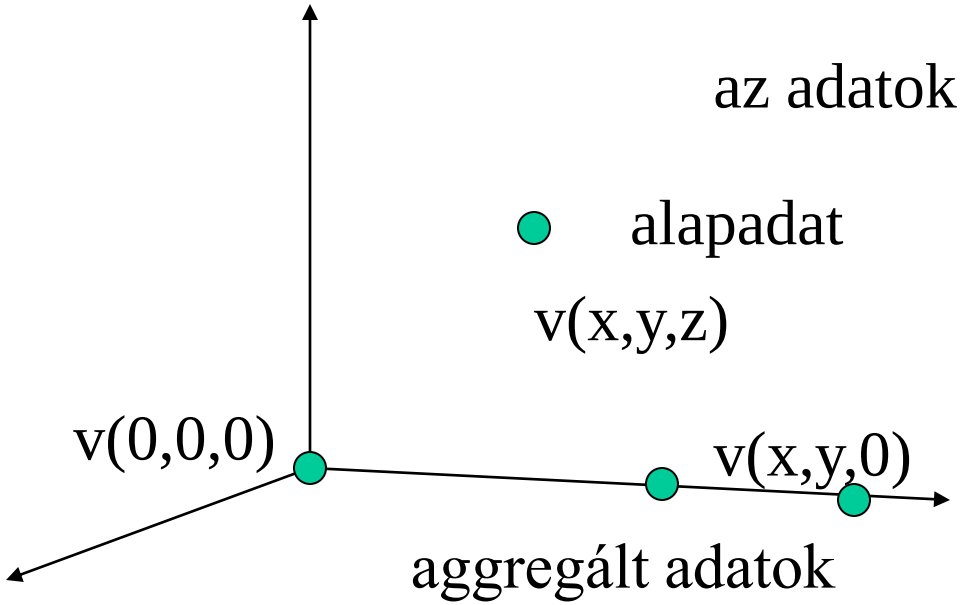
k2 szerinti aggregáció

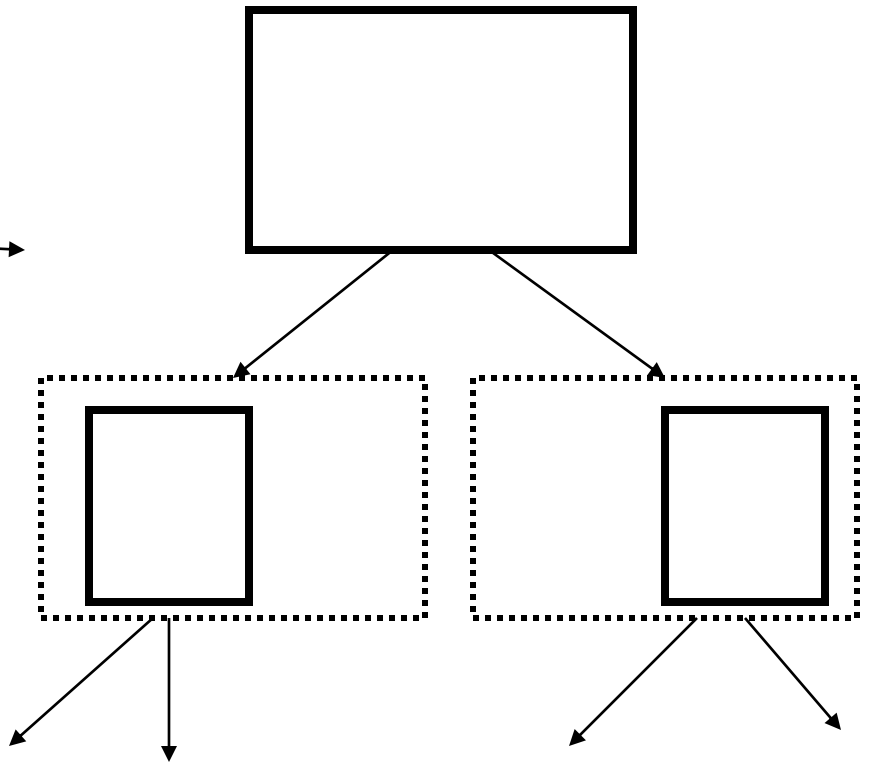
k1.k2 szerinti aggregáció

Cube-tree struktúra

az alapadatok és az aggregált adatok hatékony elérésére szolgál

az adatok R-tree struktúrában tároltak





R-tree (Antonin Guttman, 1984)
spatial access methods
minimum bounding rectangle
kiegyensúlyozott fa

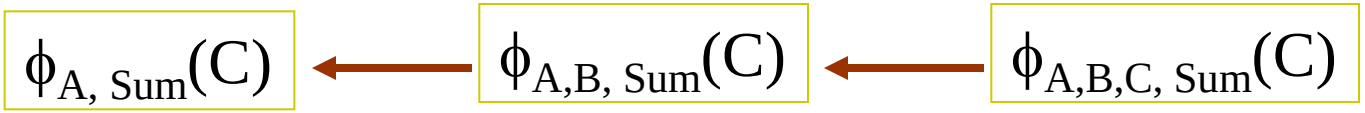
Aggregáció elszámítása

Előaggregáció előnye:
gyorsabb válaszadás
Hátránya:
több helyfoglalás
lassabb módosítás (load)

Cél az egyéb költségek minimalizálása

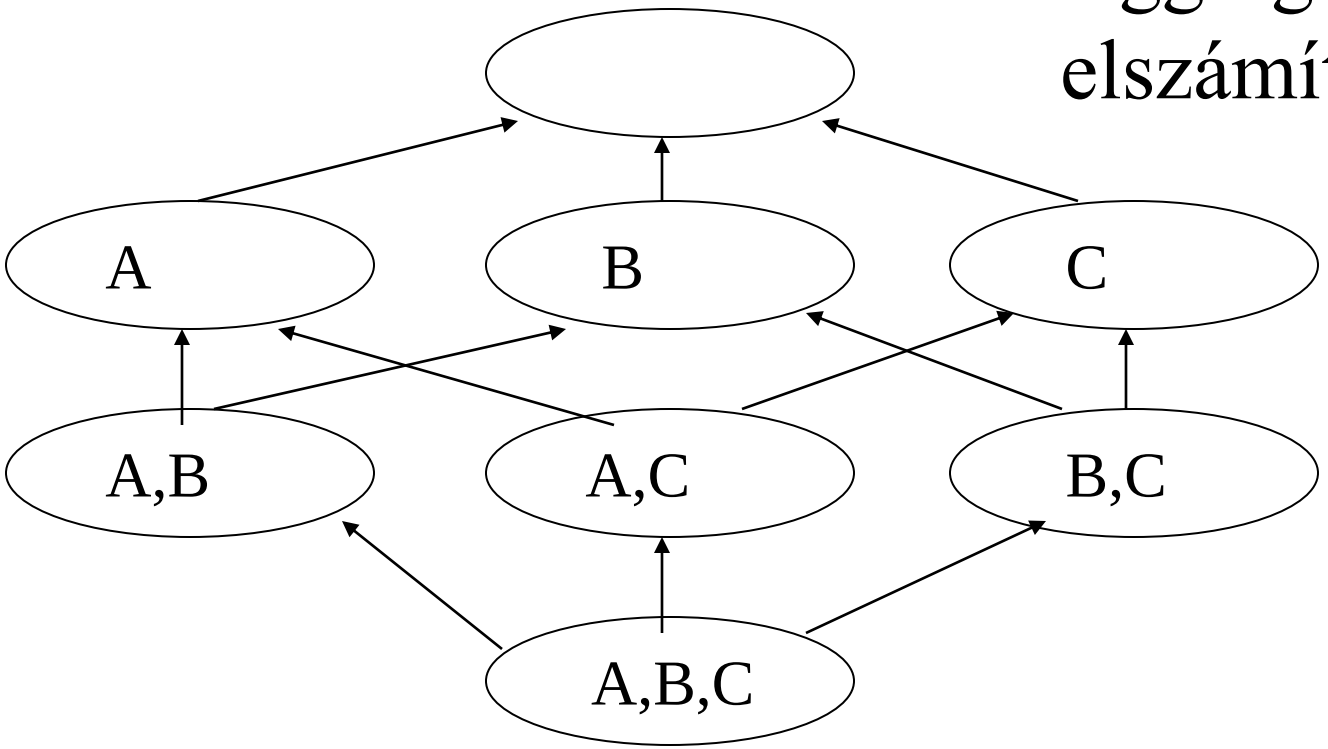
Csak a szükséges előszámításokat végezzük el

Tapasztalat: egyes aggregációk kiszámíthatók más aggregációs értékekből



Származtatási háló ez egyes aggregációs szintek között

Aggregáció elszámítása



Minden view-hoz helyköltség, minden élhez időköltség rendelhető

$\alpha \Sigma$ hely-költség $+ \beta \Sigma$ idő-költség $\Rightarrow$ minimális

Greedy-algoritmus

addig növeli a bevont elemek halmazát, amíg egy szigorú korlátba nem ütközik

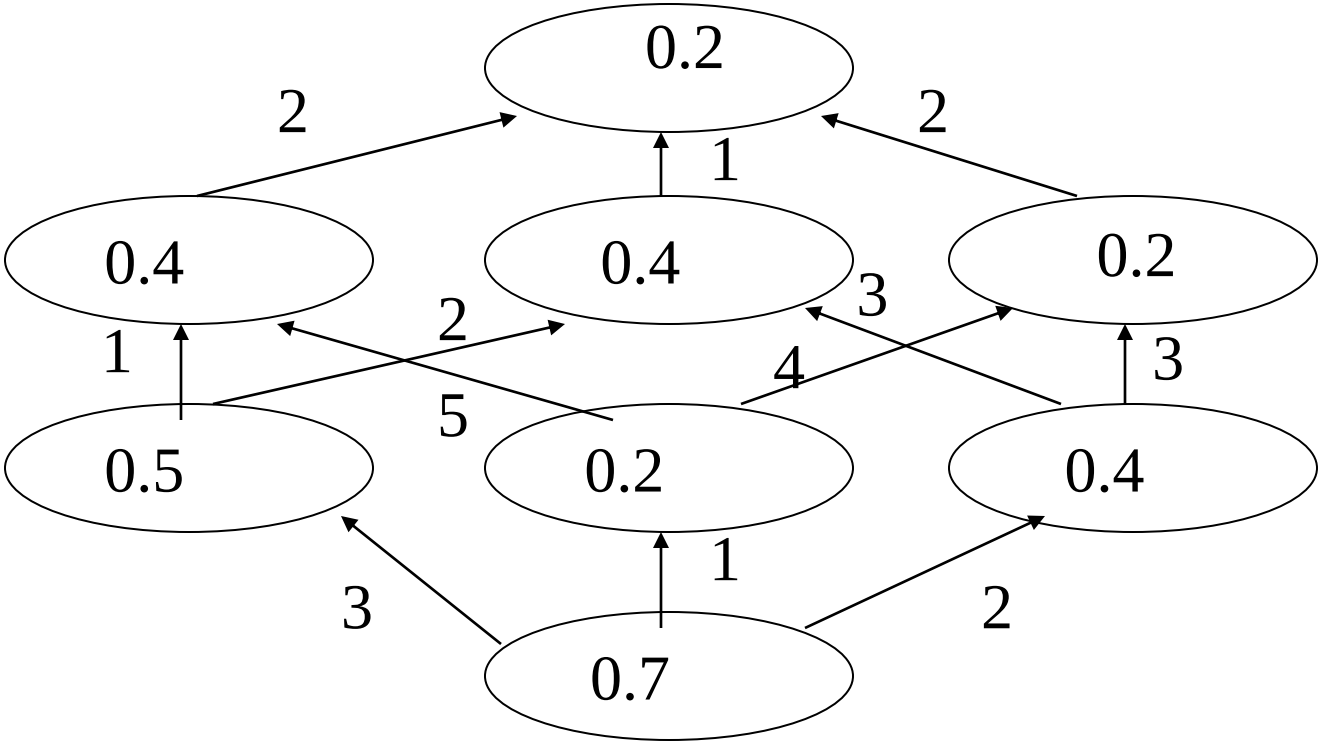
S = {alap-view}
while (Cost(S) < korlát)
{
    $v_i = \text{argmax}_i \{B(v_i,S) : v_i \notin S\}$
    $S = S \cup \{v_i\}$
}

$B(v,S) = \Sigma_{w < v} B_w$

$B_w = \text{Cost}_S(w) - \text{Cost}_v(w)$, ha $\text{Cost}_S(w) > \text{Cost}_v(w)$
0 , különben

S: a megvalósított aggregációk halmaza
v_i : egy aggregáció a még meg nem valósítottak közül
$B(v_i,S)$: a v_i megvalósításával elérhető nyereség mértéke

Greedy-algoritmus



Az eredő nyereség kiszámításánál a v hivatkozási valószínűségét is figyelembe lehet venni.

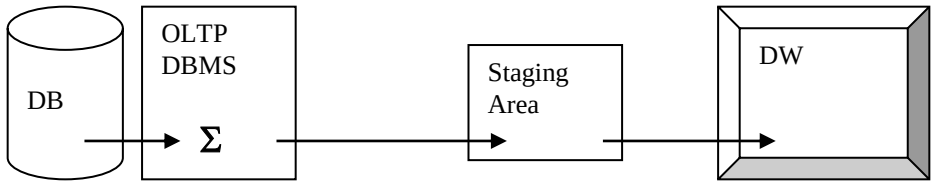
Mikor történjen az aggregációk frissítése?

- a forrás adatbázisban,
- az átemelési területen,
- a betöltés alatt,
- a betöltés után.

Frissítés a forrás adatbázisban

előnye: tehermentesíti az adattárházat az extra frissítésektől,

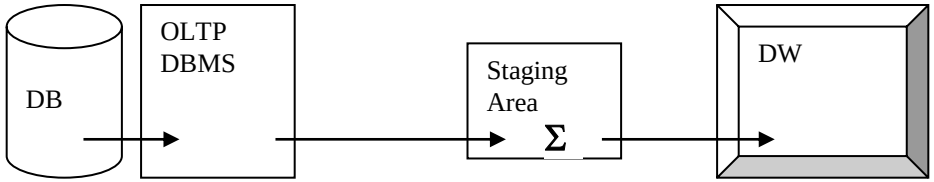
hátránya: forrást módosítani kell, hogy ott történjen meg az aggr., csak az egyazon forrásból származó adatoknál lehetséges az aggr., különböző forrásokból származó adatok összesítése a DW-ban lehetséges.



Frissítés az átemelési területen, az adatok közös tárolási formába való átültetése után történik meg az összesítés

előnye: tehermentesíti az adattárházat az extra frissítésektől, rendelkezésre állnak a különböző forrásból származó adatok,

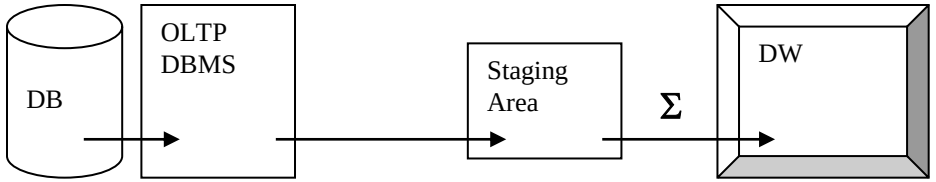
hátránya: az aggr.-hoz csak az új adatok állnak rendelkezésre, a DW-ban korábban letárolt adatokkal csak később lehet elvégezni az összesítést.



Frissítés a betöltés során

előnye: minden adat rendelkezésre áll az aggr. elvégzésére,

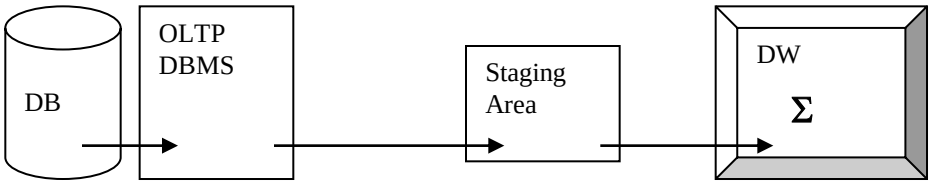
hátránya: a frissítés a betöltés része, ez a DW működési költségét növeli, a frissítés lassítja a betöltést, és a DW-funkciók folyamatát.



Frissítés a betöltés után, a betöltés és a frissítés szétválik,

előnye: minden adat rendelkezésre áll az aggr. elvégzésére,
a betöltési leterhelés után is végbe mehet,
amikor a rendszer leterhelése már kisebb,

hátránya: a DW működési költségét növeli,
a rendszer alap és aggregált adatai nem konzisztensek egymással
a betöltés és a frissítés közötti időben.



Lekérdezések kapcsolati viszonya

Milyen feltételek mellett lehet egy Q lekérdezést más V view-kból leszármaztatni?

relációk: view-k egyenértékűsége és
view-k egymásbafoglalása.

Általános esetre nem ismert még a megoldás

speciális eset: pontos illeszkedés V1 < V2:

- V1 mezői, hivatkozásai V2-ben is benne vannak,
- azonos aggregáció,
- V2 szelekciói V1-ben is benne vannak,
- V1 szelekciói szűkebbek.

```
V2: SELECT a,sum(b), avg(c), d FROM t1 WHERE t1.c > 5
        GROUP BY a
V1: SELECT a,sum(b) FROM t1 WHERE t1.c > 5  AND t1.d = 5
        GROUP BY a
```

Hatékonyabb művelet optimalizálás

a hagyományos QGM (Query Graph Model) optimalizálás is módosulhat, eddig csak SPJ műveleteket vettünk:

szelekcio(proj) - join – szelekcio – csoportképzés? –aggregáció?

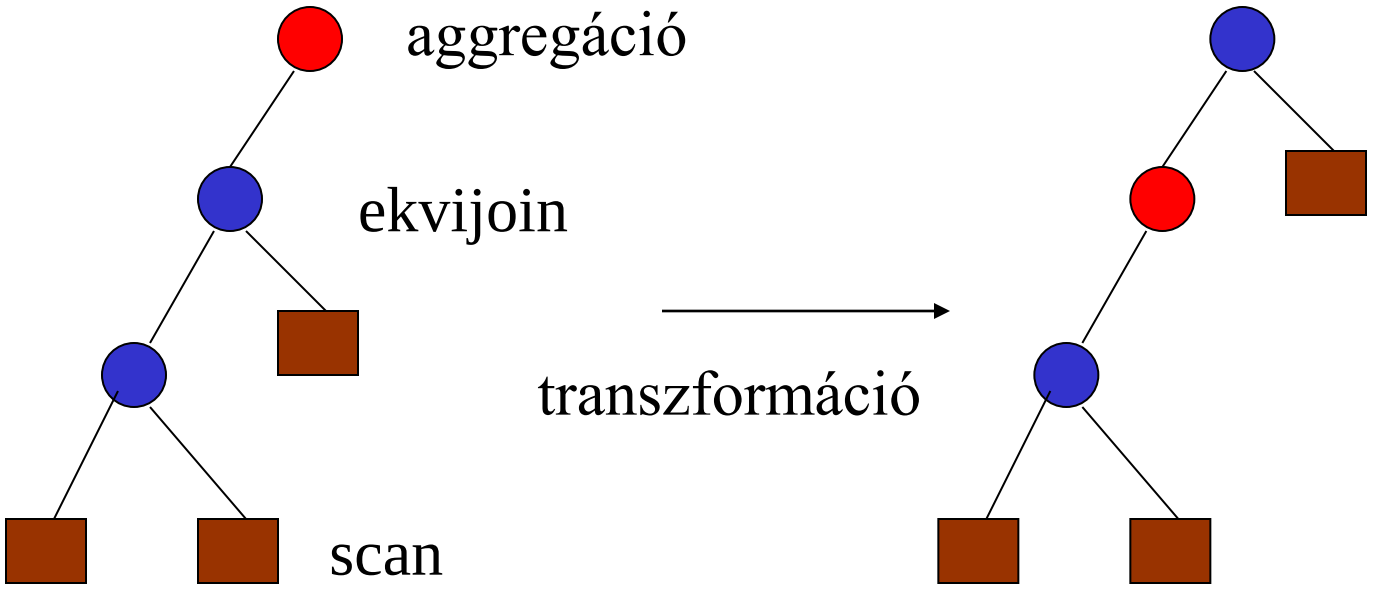
SELECT B.b, sum(A.a) FROM A,B,C WHERE A.m=B.n AND A.h = C.f AND C.l = x GROUP BY B.b

De!
Az A és B tábla join-ja közben lehet aggregálni és csoportosítani, nem kell utána újra átfutni a táblát.

Hatékonyabb művelet optimalizálás

szabály : a csoportosítás lesüllyesztése minél lentebbre

SELECT B.b, sum(A.a) FROM A,B,C WHERE A.m=B.n AND A.h = C.f AND C.l = x GROUP BY B.b



alap left-deep join

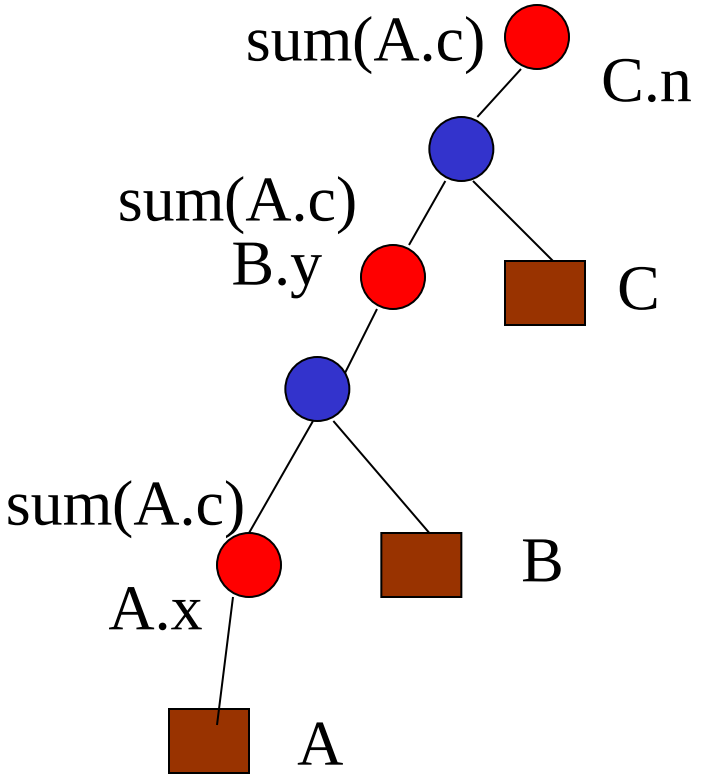
Lesüllyesztés feltételei

- a csomópont felett csak kulcs- idegenkulcs alapú join műveletek vannak,
- a csomópontban minden aggregációs mező szerepel,
- minden felettes join mező egyben csoportképzési mező is.

ez szűk esetben teljesül

kapcsolt lesüllyesztések:

- több csomópontra szétbontott




```
select C.n,sum(A.c) from A,B,C
where A.x = B.x and B.y=C.y
group by C.n
```

Adattisztítás

Hibák típusai:

- hiányzó érték,
- elírás,
- érvénytelen érték,
- nem konzisztens érték.

Adattisztítás

Ellenőrzés típusai:

- mező szintű
  - érték tartomány,
- domain szintű
  - kódolás,
- egyed szintű
  - mezők kapcsolata,
- DB szintű
  - táblák kapcsolata,
  - kulcs generálás,
- globális:
  - külső szótárak használata.

Rendszerint több szintről kell az információkat összegyűjteni. (pl. hiányzó érték)

Adattisztítás

Példa: postai címtisztítás menete

- sztring parsing,

- szavak hozzárendelése funkciókhoz
  (valószínűségi hozzárendelés, változatok),

- külső szótárak alapján helyesség ellenőrzése,

- ellentmondások feltárása,

- konfliktus feloldási javaslatok rangsorolása,

- tiszta adat generálása.