Adatbázisok elmélete 17. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu
http://www.cs.bme.hu/˜kiskat

2005

Optimalizálás

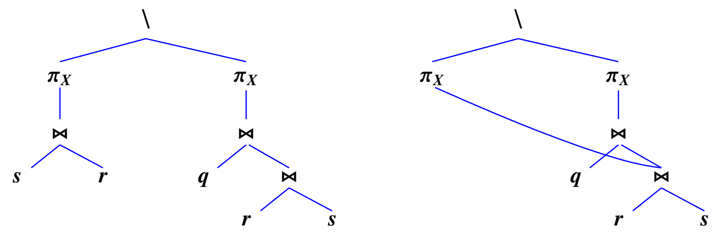
Triviális egyszerűsítések (főleg generált lekérdezések esetén hasznos):

- $r \cap r = r$; $r \bowtie r = r$; $r \cup \emptyset = r$; $\sigma_C(\emptyset) = \emptyset$
- $\pi_X(r \cup s) = \pi_X(r) \cup \pi_X(s)$
- $\sigma_{A=B \wedge B=C \wedge A=C}(r) = \sigma_{A=B \wedge B=C}(r)$

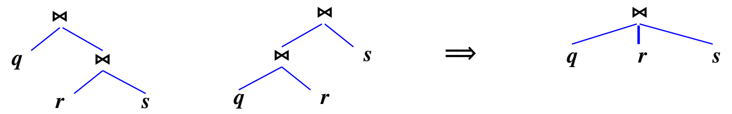
Optimalizálás

Ami többször előfordul, nem biztos, hogy érdemes mindig kiszámolni:

Pl. $\pi_X(s \bowtie r) \setminus \pi_X(q \bowtie r \bowtie s)$



Asszociativitás kihasználható:



Optimalizálás

Milyen sorrendben érdemes kiszámolni $r(A,B) \bowtie s(B,C) \bowtie q(C,D)$-t?

Szélsőséges esetben lehet, hogy bár r, s, q mindegyikének 1000 sora van, de $r \bowtie s$-nek csak 1 sora, és $s \bowtie q$-nak 1000000 sora.
$\Longrightarrow$ Sokkal gyorsabb $(r(A,B) \bowtie s(B,C)) \bowtie q(C,D)$ kiszámolása.

Ezt persze előre nem lehet tudni biztosan. $\Longrightarrow$ Statisztikákat vezetünk a relációk attribútumaiban előforduló értékekről
Ebből lehet becsülni a költségeket + dinamikus programozás vagy mohó algoritmus.

Igazi optimumot nehéz megtalálni:
Tétel. *Annak eldöntése NP-teljes, hogy néhány reláció természetes illesztésének van-e legalább egy sora.*

Tétel. *Az optimum megtalálása NP-nehéz probléma.*

Bizonyítás

Tétel. *Annak eldöntése NP-nehéz, hogy néhány reláció természetes illesztésének van-e legalább egy sora.*

Bizonyítás: Visszavezetjük rá a 3-SZÍN problémát. Adott egy gráf, kérdés színezhető-e 3 színnel.

A gráf minden $e = \{x, y\}$ éléhez vegyünk fel egy-egy relációt:

e	X	Y
	piros	kék
	piros	sárga
	kék	piros
	kék	sárga
	sárga	piros
	sárga	kék

e'	X	Z
	piros	kék
	piros	sárga
	kék	piros
	kék	sárga
	sárga	piros
	sárga	kék

Ha a természetes illesztésnek van sora $\Longrightarrow$ egy sor minden csúcshoz rendel egy színt. Mivel az illesztés megfelel az egyes relációknak, egy él két végpontján nem lesz ugyanolyan szín.

Ha van színezés $\Longrightarrow$ a színezésben minden élre vegyük ki a megfelelő színpárt. Ezek a sorok összeillenek. $\surd$

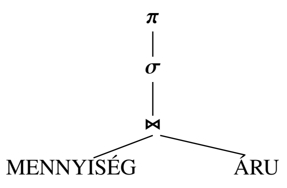
Kiválasztás tologatása

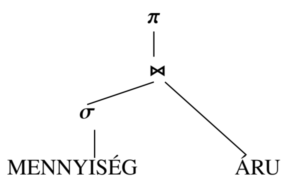
ÁRU(ÁRUKÓD, ÁRUNÉV, EGYSÉGÁR)
MENNYISÉG(DÁTUM, ÁRUKÓD, DB)

Hány darabot adtak el 2002. jan. 15-én az A123 kódú áruból, mi a neve és az ára?

$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG} \bowtie \text{ÁRU}\right)\right) \Longrightarrow$

$\pi_{\text{DB, ÁRUNÉV, EGYSÉGÁR}}\left(\sigma_{\text{ÁRUKÓD='A123'} \wedge \text{DÁTUM='2002-01-15'}}\left(\text{MENNYISÉG}\right) \bowtie \text{ÁRU}\right)$



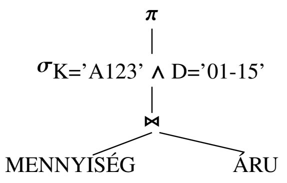


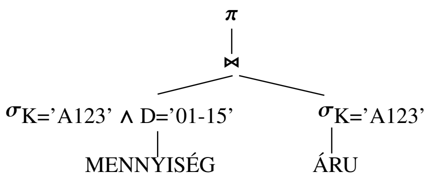
Felhasznált azonosság:
$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$, ha minden C-beli attribútum szerepel R-ben.

Hasonló azonosságok:
$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$,
$\sigma_C(R \times S) = \sigma_C(R) \times S$, ha minden C-beli attribútum szerepel R-ben.
$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie \sigma_C(S)$, ha minden C-beli attribútum szerepel R-ben és S-ben is.

Kiválasztás tologatása

Összetett C szétszedhető:
$\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$,
$\sigma_{C_1 \vee C_2}(R) = \sigma_{C_1}(R) \cup_H \sigma_{C_2}(R)$, ha R nem multihalmaz.





Lehet, hogy érdemes előbb feltolni, aztán le.

Más műveletekre vonatkozó szabályok

Hasonló szabályok π, δ, γ-ra is vannak, de ezek nem annyira csökkentik a műveletigényt. De csak olyan attribútumot lehet eltüntetni, amire nem hivatkozunk feljebb.

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(R))$, ahol M az R olyan attribútumai, hogy vagy összekapcsolási attribútum, vagy L-beli, N pedig …
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(\sigma_C(R)) = \sigma_C(\delta(R))$
- $\delta(\gamma_L(R)) = \gamma_L(R)$

De pl. δ nem tolható át $\cup_M, \pi$-n.

Még egy fontos kérdés az alkérdések kezelése, de erről most nem szólunk.

Összefoglalás

Sok ilyen szabály alkalmazásával több féle logikai terv előállítható.

Ezeknek megbecsüljük a költségét és választunk egyet. Ehhez készítünk fizikai tervet.

Jobb rendszerekben ez automatikus. Ilyenekben kérdéses, hogy a lekérdezés beírásakor mire kell figyelni.

Inkább olyan egyszerűsítéseket érdemes csak elvégezni, ami a függések következménye, mert ezeket nehezebben lehet automatizálni.

Pl. Oracle-ban van rá mód, hogy megnézzük mi a logikai és fizikai terv és meg lehet adni, hogy pontosan mit csináljon.

Fizikai szervezés, tárkezelés

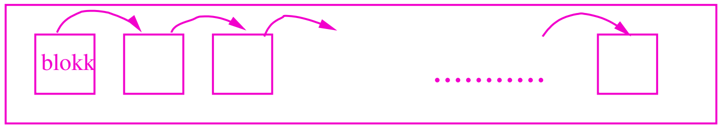
Célja: a rekordokból (egy rekord = a reláció egy sora) álló állomány kezelése úgy, hogy az adatokhoz való hozzáférés gyors legyen.

Fontos jellemzők:

- külső táras adatkezelés, mert sok az adat $\Longrightarrow$ ha valamivel dolgozni akarunk, akkor be kell hozni a belső memóriába $\Longrightarrow$ a költséget a beolvasás/kiírás jelenti $\Longrightarrow$ az I/O műveletek számára akarunk optimalizálni
- a műveletek, amiket gyorsan meg kell tudni csinálni: rekordok beillesztése, törlése, módosítása, keresése

Az állomány felépítése

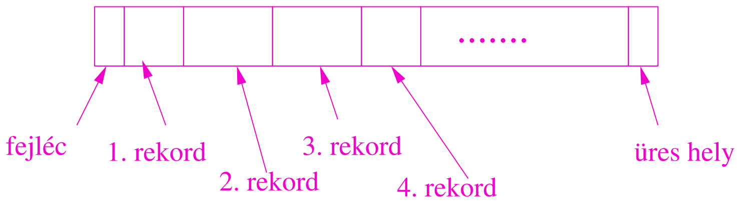
Az adatállomány a külső táron van, blokkok (lapok) elérésfolytonos sorozatán.



- egyszerre egy blokk írható ki/olvasható be
- blokk méret fix (ált. 2^{10}, 2^{12} byte)
- az operációs rendszer tartja nyilván, hogy melyik reláció rekordjai hol vannak és ő biztosítja az elérésfolytonosságot is

Blokkokról általában

Tipikus blokk



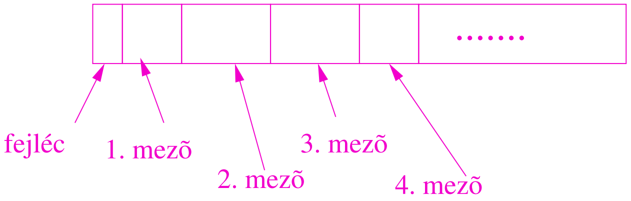
A fejléc tartalmazza a blokkra vonatkozó infókat, (pl: melyik relációhoz tartozik, mennyi a szabad hely benne, hol kezdődik); ezután jönnek a rekordok egymás után, a végén általában marad üres hely.

Fontos feltevés: rekordok blokkhatárt nem lépnek át, ezért általában van üres, fel nem használható hely a blokkok végén. (Ha nagyok a rekordok, pl. képfile-ok, és mégis át kell lépni laphatárt, akkor extra technikák kellenek, de ezzel most nem foglalkozunk.)

Rekordok típusai

Kötött formátum

Ekkor a mezők száma, mérete, típusa és sorrendje fix



Fejléc:

- a rekord kezelésével kapcsolatos infók: törölt-e, melyik relációhoz tartozik
- a mezők típusa
- időbélyeg (mikor módosult utoljára)

Rekordok típusai

Változó formátum

- Mezők hossza esetleg nem fix (szöveget tartalmazó adatbázisok)
- Ismétlődő mezők lehetnek, és az ismétlések száma nem fix vagy pedig többértékű mezők (szereplők felsorolása filmnél)

Ilyenkor bonyolultabb a fejléc, kevésbé lehet gazdaságosan/előre tervezhetően tárolni a rekordokat, ezért érjük el inkább, hogy ne legyen ez az eset:
Vezessük vissza ezt az esetet a kötöttre, pl. mutatók alkalmazásával a problémás helyeken
$\Longrightarrow$ Mostantól feltesszük, hogy a rekordok kötött formátumúak és hogy az egész állományon belül ugyanaz a formátum van.

Fontos fogalmak

1. **mutató:** blokk vagy rekord címét tartalmazó bejegyzés
2. **kötött blokk/rekord:** mutathat rá mutató, ezért nem mozgatható el szabadon. Ez típusszinten adott, azaz ha egy reláció rekordjaira/blokkjaira mutathat mutató, akkor még akkor is kötöttnek számít, ha éppen nem mutat egyre se semmi.
3. **szabad blokk/rekord:** nem mutathat rá mutató
4. **Kulcs, keresési kulcs** (néha csak kulcsnak hívjuk):
   - a rekordok mezőinek egy kitüntetett halmaza (a reláció attribútumainak egy részhalmaza)
   - ez alapján megy a keresés (ezeknek az értékét adjuk meg és azokat a rekordokat (sorokat a relációban) keressük, amiknél pont ezek az értékek szerepelnek)
   - a keresési kulcs nem egyezik meg feltétlenül a reláció egyik kulcsával sem (pl. név a telefonkönyvnél)
   - de az azért elvárás, hogy ne legyen nagyon sok egy-egy értékre illeszkedő rekord

Alapvető állományszervezési technikák

Milyen struktúrát hozzunk létre az adatok tárolására?

Lehetőségek:

1. Szekvenciális tárolás
2. Hash
3. Indexek

Szekvenciális tárolás

Nincs semmi struktúra, lineárisan töltjük fel az állományt, az új rekord az első alkalmas szabad helyre kerül.

- **keresés**: egyesével beolvassuk a lapokat a belső memóriába, lineáris keresés, ha N lap van, akkor átlagosan $\frac{N}{2}$ I/O művelet
- **törlés**: keresés, aztán törlésbittel (ha sok törlés volt, esetleg garbage collection időnként)
- **beszúrás**: keresünk szabad helyet és oda rakjuk. Ehhez egyesével beolvassuk a blokkokat, ha van üres hely oda rakjuk, ha nincs sehol, akkor az állomány végére. Ha az utolsó lapra se fér: új lapot kérünk
- **módosítás**: keresés, majd ha befér az eredeti helyére, akkor oda teszem vissza a módosítás után, különben meg törlés és beszúrás

Akkor jó így tárolni, ha kevés az adat. Előnye, hogy nem kell adatszerkezettel vesződni.

Hash (tördelés)

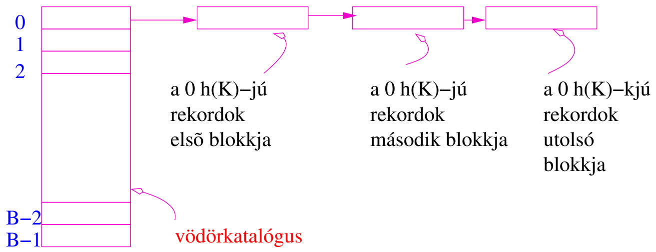
Külső táras tárolás miatt vödrös hash (nyílt címzés nem menne)

Alapvető szerkezet: B vödör, ezekbe rakjuk majd a rekordokat. A keresési kulcs (K) értékétől függően.

Adott egy hash függvény, ami leképezi a tárolandó rekordokat (a keresési kulcsokat) a $[0, B-1]$ intervallum egész értékeire, azaz $h: K \to h(K) \in [0, B-1]$
Ez adja meg, hogy egy rekord melyik vödörbe kerüljön.
(A lehetséges K-k, a keresési kulcsok értékei, egy nagy univerzumból kerülnek ki, de az összes előfordulásuk száma ennél jóval kisebb. B-t úgy választjuk meg, hogy a várható blokkszámmal legyen nagyjából egyenlő)

Elvárások a hash-függvénnyel szemben: gyorsan számolható legyen, kevés ütközést okozzon. Jó pl. a szorzó vagy az osztómódszer (lásd algel)

Hash



Vödörkatalógus: tipikusan a belső memóriában tároljuk, ebben csak B darab mutató van, ez alapján tudjuk, hogy adott $h(K)$ esetén hol van az első blokkja a h(K) hashértékű rekordoknak.

Egy vödör: azonos h(K)-jú rekordok halmaza, ezek néhány (jó esetben egy, de esetleg sok) blokkban vannak. Az egy vödörbeli blokkok között mutatókon tudunk mozogni. A vödrön belül rendezettség semmi nincs, a rekordok az érkezési sorrendjükben vannak.

Műveletek

- **keresés**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödröt végigkeressük szekvenciálisan
- **beszúrás**: K alapján meghatározzuk $h(K)$-t, a vödörkatalógusban a $h(K)$-hoz tartozó vödörbe rakjuk be a rekordot az első szabad helyre
- **törlés**: keresés, majd törlésbittel

Költség: ha jól van megválasztva B (nem nő túlságosan az állomány), akkor átlagosan konstans I/O művelettel megvan minden (legrosszabb esetben azonban nagyon rossz is lehet, annyira, mint a szekvenciális szervezés). Akkor jó, ha rövid blokkláncokból állnak a vödrök, ezért kellett B-t annyinak választani, mint a várható blokkszám.

Baj: ha elrontjuk B választását, túl dinamikusan nő az állomány $\Longrightarrow$ hosszú blokkláncok, lassú műveletek