Adatbázisok elmélete 20. előadás

Csima Judit
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 136/b
csima@cs.bme.hu

2003. Április 22.

Adatbázisrendszerek megvalósítása

Eddig az adatbáziskezelők működéséről tanultunk. Az év hátralevő részében az ilyen rendszerek belső működését tanulmányozzuk egy kicsit.
Három nagyobb témakör:

1. Fizikai szervezés, tárkezelés: hogyan tároljuk a relációkat oly módon, hogy gyorsan lehessen keresni, illetve módosítani?
2. Lekérdezésfeldolgozás: hogyan értékelődnek ki a lekérdezések, milyen módszerek vannak a lekérdezések végrehajtására?
3. Tranzakciókezelés: többfelhasználós működés biztosítása, illetve rendszerhibák elleni védelem.

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

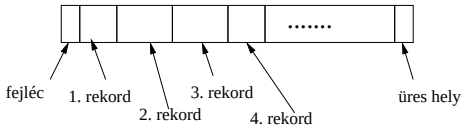
Az állomány felépítése

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



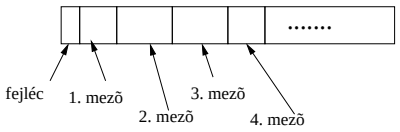
A fejléc tartalmazza a blokkra vonatkozó infókat, (pl: melyik relációhoz tartozik, mennyi a szabad hely benne, hol kezdődik); ezután jönnek a rekordok egymás után, a végén általában marad üres hely.

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)

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 muatathat 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.

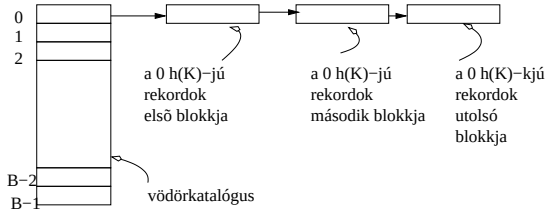
Hash

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)



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

Növelhető hash

Kiküszöböli a vödrösh hash azon hibáját, hogy nem tud alkalmazkodni a gyorsan növő állományhoz

Fő elvek:

- kiszámoljuk $h(K)$-t, de az eredménynek csak az első pár jegye számít abban, hogy melyik rekord hova kerül
- dinamikusan változik, hogy hány bit számít és ezzel együtt az is, hogy hány vödör lesz
- a vödrök mérete fix, tipikusan egy blokkból állnak. Így majd a műveletek 1 lapeléréssel menni fognak, egy kis belső memóriában való keresgélés után.

Jellemzők:

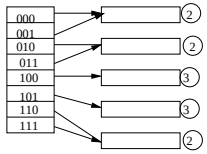
- a d változó mindig a struktúra aktuális globális mélységét tárolja, ennyi bitig számít $h(K)$ értéke
- 2^d bejegyzés lesz a vödörkatalógusban, mert 2^d darab d hosszú bitsorozat van
- a vödörkatalógus most is egy mutatókból álló (most éppen 2^d elemű) tömb
- a vödrök száma s^d-nél lehet kevesebb is, ha több mutató is mutat ugyanarra a vödörre

A műveletek közös vonása: kiszámoljuk $h(K)$-t d bitig és aztán a vödörkatalógus $h(K)$-hoz tartozó mutatója alapján elmegyünk abba a vödörbe (blokkhoz), ahol az ezen d bittel kezdődő $h(K)$ értékű rekordok vannak.
Ha két mutató ugyanoda mutat, akkor különböző d hosszú bitsorozattal kezdődő $h(K)$ értékű rekordok ugyanoda kerülnek.

Minden vödörnek van egy lokális mélysége, ezt a d' változó tárolja. Ez azt mutatja, hogy az ebbe a vödörbe kerülésnél az első hány darab bit számít. Természetesen $d' \leq d$ áll minden vödörre, $d' < d$ akkor van, ha több mutató ugyanoda mutat.

A struktúra felépítése

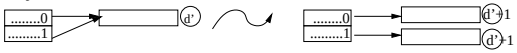
A struktúra vázlata $d = 3$ esetén:



Mivel $d = 3$, ezért az első három bitig számoljuk ki $h(K)$-t. Most összesen öt vödör van a lehetséges maximális nyolc helyett, mert azok a rekordok, amiknek a $h(K)$-ja 00-val kezdődik elférnek egy vödörben, itt már az első két bit alapján tudjuk, hogy melyik vödörbe kerül a rekord. Hasonló a helyzet a 01 és 11 kezdetű $h(K)$ értékekkel is. Egyedül az 10 első két bit esetén számít, hogy mi a harmadik (az 10 kezdetűek nem férnek el egy vödörben).

Műveletek megvalósítása

- **keresés**: az adott K keresési kulcsra kiszámoljuk $h(K)$ első d bitjét, a vödörkatalógus megfelelő mutatója alapján tudjuk, hogy honnan kell beolvasni az egy blokkból álló vödröt, amiben a rekordak lennie kell
- **beszúrás**: beszúrni kívánt rekord K értékére kiszámoljuk $h(K)$-t, vödörkatalógus mutatóját követve behozzuk a blokkot, ahova kerülnie kell. Ha belefér ez a rekord is, akkor belerakjuk és kiírjuk a blokkot; ha nem fér bele, akkor szét kell szedni a blokkot két részre, egy vödör helyett lesz kettő. Két eset van:
  - $d' < d$, a vödör lokális mélysége kisebb d-nél: $d' := d' + 1 \leq d$ lesz, a vödörkatalógus nem változik, csak az eddig ugyanarra a vödörre mutató két mutató most majd két külön blokkra mutat:

    Természetesen az eddig egy vödörben levő rekordokat szét kell válogatni aszerint, hogy a $d'+1$-edik bitje mi a $h(K)$-nak

  - Ha $d' = d$, azaz már nem lehet növelni a lokális mélységet d változtatása nélkül: $d := d + 1$, a vödörkatalógust megduplázom egy $x_1 \ldots x_d$ bejegyzés helyett lesz kettő: $x_1 \ldots x_d\ 0$ és $x_1 \ldots x_d\ 1$, az ezekből kiinduló két mutató ugyanoda megy, ahova a korábbi egy, a lokális mélység nem változik.

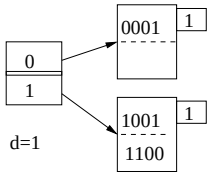
    

    Ezek után már végrehajtható a vödörszétvágás úgy, mint az előbb, mert most már $d' < d$ mindenhol az új d-vel.
- **törlés**: keresés, aztán pedig törlés a vödörből; ha ettől olyan helyzet áll elő, hogy a vödör összevonható a párjával (ahol az első $d' - 1$ bit egyezik, de a d'-edik más), akkor összevonás (két mutató ugyanoda fog mutatni) és d' csökkentése eggyel. Ha minden $d' < d$, akkor d-t is csökkentjük eggyel.

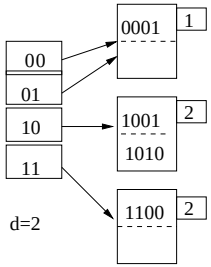
A fentiek miatt csak akkor mutathat két mutató ugyanarra a d' lokális mélységű vödörre, ha a két megfelelő érték első d' bitje megegyezik.

Példa

Növelhető hash segítségével akarjuk tárolni az adatainkat. Feltesszük, hogy egy lapra két rekord fér. A hash függvény 4 bites számot ad vissza, de jelenleg még csak egy bitet használunk ($d = 1$), mivel eddig csak három elem (0001, 1001, 1100) van a táblázatban (az egyszerűség kedvéért az elemek helyett azt az értéket irjuk be, amit a hash függvény visszaad).
Tehát most így néz ki a tábla, van egy bejegyzés a 0-hoz és egy az 1-hez:

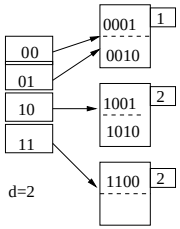


Ha most beszúrni szeretnénk egy olyan elemet, melyre a hash függvény az 1010 értéket adja, akkor az új tábla ilyen lesz:

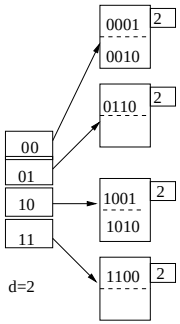


Mivel az 1-hez tartozó lap már betelt, ezért itt a lokális mélységet növelni kellett 1-ről 2-re (két bitet akarunk figyelembe venni), de így a d értékét is növelnünk kellett.

Ha most jön egy 0010 hash-értékű elem, akkor azt simán be tudjuk rakni a helyére:



De ha ezután olyan jön, ahol a hash függvény értéke 0110, akkor új lapot kell létrehoznunk, de a d értékét nem kell növelnünk, mert elég lesz az eddig ugyanoda mutató két mutatót szétszedni és ezen új lapoknál a lokális mélységet 2-re állítani.



Ha ezután jönne egy olyan elem, aminek az értéke 00-val vagy 10-val kezdődik, akkor újabb lapra lenne szükségünk, de ehhez már a d értékét is növelni kellene.