Evaluarea operatorilor algebrici relaționali

1. Factori de reducție

Statistici și cataloage

SGBD încearcă să facă niște estimări. Alegerea unei tehnici sau a alteia se bazează pe *estimarea* numărului de pagini pe care trebuie să le manipulăm în fiecare dintre aceste tehnici. Pentru a face aceste estimări, e nevoie de niște informații, care sunt de obicei stocate în **catalogul** bazei de date, pe lângă structura obiectelor ce compun baza de date.

Catalogul unei baze de date conține cel puțin următoarele informații despre tabele și indecși:

- numărul de înregistrări (*NTuples*) și numărul de pagini (*NPages*) ale fiecărei tabele
- numărul de valori distincte ale cheilor de indexare (*NKeys*) și numărul de pagini (*NPages*) pentru fiecare index
- înălțimea și valorile minime și maxime ale cheilor (*Height / Low / High*) pentru fiecare index cu structură de arbore

Cataloagele sunt actualizate periodic (nu imediat după o actualizare, pentru că ar fi foarte costisitor). Uneori, pentru anumite tabele, ne poate ajuta să avem stocate informații mai detaliate, cum ar fi histograme (histogramele pentru valorile unui câmp nu sunt memorate decât la cerere).

Factori de reducție – ne ajută să înțelegem la ce cardinalitate ne referim, la câte pagini să ne așteptăm.

Factorul de reducție (FR) – este asociat unui anumit termen – *term* (care poate să apară într-o clauză where) și reflectă care este impactul acelui termen în reducerea dimensiunii rezultatului.

```
SELECT attribute list
FROM relation list
WHERE term1 AND ... AND termk
```

- *col* = *val* are FR: *1 / NKEYS(I)*, pentru indexul I pe col
- $col_1 = col_2$ are FR: $1 / MAX(NKeys(I_1), NKeys(I_2))$
- *col* > *val* are FR: *(High(I) – val) / (High(I) – Low(I))*

2. Evaluarea operatorilor selecție și proiecție

Selecția simplă

```
SELECT *
FROM   Students S
WHERE  S.sname < 'C%'
```

- Are forma $\sigma_{R.câmp\ OP\ val}(S)$
- Dimensiunea rezultatului este aproximată de dimensiunea lui S * factorul de reducție (dimensiunea lui S = câte înregistrări sunt în tabela S)
- Presupunând că avem 30 de litere, factorul de reducție este 3 / 30 (căutăm studenții care încep doar cu 3 litere – A, B și C, și există 30 de litere din alfabet) = 1 / 10
- Tabela *nu are index* și e *nesortată*: trebuie scanată întreaga tabelă → costul este N (numărul de pagini în S)
- Tabela *nu are index* și e *sortată*: căutare binară → costul este Log_2N
- Tabela *are index* pentru atributul de selecție (S.name): folosește indexul pentru determinarea înregistrărilor

Utilizarea unui index pentru selecții:

Costul depinde și de tipul de index pe care îl folosim (poate fi un index cu acces direct sau un index în forbă de arbore-B), și dacă este grupat sau nu.

!!! Se adaugă și costul returnării înregistrărilor (poate fi mare fără clusterizare).

Rafinare importantă a indecșilor ne-clusterizați:

1. găsirea înregistrărilor
2. sortarea acestora după rid (adresa / identificatorul fizic al înregistrărilor pe hard disk)
3. se citesc rid în ordine; se asigură că fiecare pagină de date este adusă în memoria internă o singură dată

Condiții de selecție generale

$$(day<8/9/94 \text{ AND } grade=10) \text{ OR } cid=5 \text{ OR } sid=3$$

- Fiecare condiție de selecție este adusă la *forma normală conjunctivă* (FNC)

$$(day<8/9/94 \text{ OR } cid=5 \text{ OR } sid=3) \text{ AND}$$
$$(grade=10 \text{ OR } cid=5 \text{ OR } sid=3)$$

- Indexul trebuie să conțină în prefix toți termenii care apar în condiție. De exemplu, dacă a = 5 AND b = 3, se potrivește un index pe <a, b, c>, dar nu pe b = 3.

Abordări ale selecțiilor generale:

1. Găsirea celei mai selective căi de acces
   - utilizarea acelui index care ne restrânge cel mai mult numărul de înregistrări
   - de exemplu: *day < 8/9/94* AND *cid = 5* AND *sid = 3* → se poate utiliza un index B-arbore pe *day*; apoi, *cid = 5* și *sid = 3* trebuie verificate pentru fiecare înregistrare *returnată*; similar, poate fi folosit index pe <*cid, sid*>, iar mai apoi trebuie verificat *day < 8/9/94*
   - folosind acei factori de reducție, încercăm să estimăm care dintre căi (folosind indexul pe *day* sau cel pe <*cid, sid*>) ne dă cele mai puține înregistrări → cea mai selectivă cale de acces

2. Utilizarea tuturor indecșilor (dacă sunt 2 sau mai mulți indecși)
   - se obține lista de *rid* ale înregistrărilor folosind fiecare index
   - se intersectează listele de *rid*
   - dacă nu avem indecși pentru toate câmpurile utilizate, îi folosim pe toți și, pe rezultat, evaluăm condițiile care au rămas pe celelalte câmpuri care nu erau indexate

Operatorul proiecție

```
SELECT  DISTINCT
        E.sid, E.cid
FROM    Evaluations E
```

- Are forma $\pi_{cid,\ sid}$Evaluations
- Dacă nu aveam acel DISTINCT, problema era trivială: mergeam pe toate înregistrările pe tabela Evaluations, tăiam câmpurile care nu ne interesau și afișam pe ecran doar valorile câmpurilor care se cer (o simplă scanare a tabelului)
- Avem doi pași pentru implementarea proiecției:
  - se elimină câmpurile nedorite → obținem mai puține pagini
  - parcurgem iar înregistrările și eliminăm duplicatele
- Găsirea duplicatelor se realizează prin două tehnici:
  - prin *sortare*
  - folosind *funcții de dispersie*

Proiecție bazată pe sortare

- Pas 1 – Scanarea tabelei E, eliminăm câmpurile nedorite și salvăm într-o tabelă temporară doar înregistrările ce conțin *cid* și *sid* → Cost = N + T (N = numărul de pagini din E, T = numărul de pagini din tabela temporară (E') care conține doar câmpurile dorite)
- Pas 2 – Sortarea înregistrărilor → Cost = T * $\log_2$T
- Pas 3 – Scanarea rezultatelor sortate → Cost = T

Putem optimiza acest proces? Well, yes.

Sortarea înseamnă că noi inițial citim toate acele E pagini în funcție de câtă memorie avem în buffer, apoi se face sortarea. Se trece la următoarele pagini, se aduc în buffer cât ne permite memoria, și iar se sortează. Aici, putem să eliminăm direct câmpurile care nu ne interesează și salvăm în niște pagini de memorie temporare.

În momentul în care citim din acele pagini temporare și interclasăm, ne putem folosi de interclasare ca să eliminăm duplicatele.

Cost:

- Pas 1:
  - Scanare Evaluations cu 1000 I / Os
  - Dacă o înregistrare din E' e 10 octeți, se vor salva în tabela temporară E' 250 pagini
- Pas 2:
  - Având 20 pagini în buffer, se sortează E' în doi pași la costul de 2 * 2 * 250 I / O (citesc și salvez 250 de pagini în 2 pași → 2 * 2)
- Pas 3:
  - 250 I / O cost la scanarea de găsire a duplicatelor
- Cost total: 2500 I / O (estimat)

Costul variantei îmbunătățite:

- Pas 1:
  - Scanare Evaluations cu 1000 I / O
  - Salvează E' cu 250 I / O
  - Având 20 de pagini în buffer, 250 de pagini sunt salvate ca 7 subșiruri sortate, fiecare având 40 de pagini → se folosește varianta optimizată a sortării externe
- Pas 2:
  - Se citesc subșirurile sortate (250 I / O) și se interclasează
- Cost total: 1500 I / O

Proiecție bazată pe funcție de dispersie

1. Folosim funcția de dispersie pentru a împărți toate înregistrările în partiții:
- Pentru fiecare înregistrare se elimină câmpurile nedorite și se aplică o funcție de dispersie h_1 pentru a stoca înregistrarea într-unul dintre cele B – 1 pagini rămase (B – numărul de partiții)
- 2 înregistrări din 2 partiții diferite sunt distincte
2. Pentru fiecare partiție se aplică o funcție de dispersie h_2 (diferită de h_1) pentru a elimina duplicatele
- Dacă partiția nu încape în memorie se va aplica algoritmul de proiecție, recursiv.

Cost:

- Partiționare:
  - Citire: E = N I / O
  - Salvare: E' = T I / O
- Eliminarea duplicatelor:
  - Citirea partițiilor: T I / Os
- Cost total = N + 2 * T I / Os
- Exemplu *Evaluations* = 1000 + 2 * 250 = 1500 I / Os