Databázové technologie

Provádění dotazů (dokončení) & Překlad dotazu

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

Dvouprůchodové algoritmy založené na hashování

- řeší stejný problém jako dvoufázové algoritmy založené na řazení
- používá jiné prostředky
- pokud se tabulka nevleze do dostupných bufferů v paměti, je rozdělena do několika bucketů podle hashovací funkce
- každý bucket je zpracován jednoprůchodovým algoritmem pracujícím s daty v paměti
- volba hashovací funkce musí zajistit, že řádky, které se spolu účastní dané operace, jsou ve stejném bucketu
- taková funkce vždy existuje, vztahuje se buď k celému řádku (množinové operace) nebo jeho části (seskupení, spojení)
- v případě binárních operací, je nutné použít stejnou hashovací funkci pro oba operandy

Rozdělení hashovací funkcí

- s pomocí M bufferů
- a hashovací funkce h
- rozdělí vstupní tabulku do M - 1 bucketů

```
initialize M - 1 buckets with M - 1 buffers
for each block b in R:
   read block b into Mth buffer
   for each row t in block b:
      if buffer for bucket h(t) is full then
          copy the buffer to disk
          initialize new empty block in that buffer
      copy t to buffer for bucket h(t)
for each bucket:
   if the buffer of a bucket is not empty:
       write the buffer to disk
```

Dvouprůchodové algoritmy založené na hashování

- předchozí procedura rozdělí vstupní tabulku R na tabulky $R_1, R_2, \ldots R_{M-1}$
- algoritmy použitelné pokud, pro každou tabulku R_i platí $B(R_i) \leq M$ (tj. každou část původní tabulky lze zpracovat v paměti)
- unární operace aplikovány po částech a pak výsledky sjednoceny

$$f_H(R) = f_M(R_1) \cup \ldots \cup f_M(R_{M-1})$$

- anologicky pro binární operace

$$f_H(R, S) = f_M(R_1, S_1) \cup \ldots \cup f_M(R_{M-1}, S_{M-1})$$

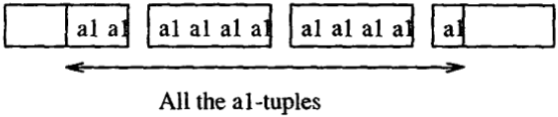
- f_H označuje dvouprůchodovou operaci založenou na hashování ($\sigma, \delta, \gamma, \bowtie, \cup, \ldots$)
- f_M označuje jednoprůchodovou operaci pracující v paměti ($\sigma, \delta, \gamma, \bowtie, \cup, \ldots$)
- γ a $\bowtie$ ((partition) hash-join) používají jen odpovídající atributy

Dvouprůchodové algoritmy založené na hashování (analýza)

- náročnost I/O: $3 \cdot B(R)$, resp. $3 \cdot (B(R) + B(S))$ (načtení, uložení, načtení)
- náročnost paměť:
  - velikost $B(R_i) = \frac{B(R)}{M-1}$
  - zpracování každé části jednoprůchodovým algoritmem vyžaduje M bufferů
  - $B(R) \leq M(M-1)$, tj. přibližně $B(R) \leq M^2$
  - analogicky pro binární operace $min(B(R), B(S)) \leq M^2$ (menší operand je v paměti)
- z pohledu nároků na I/O i paměť jsou algoritmy založené na hashování stejné

Použití indexů

- shlukovaný index – pro zvolenou hodnotu atributu (atributů) jednotlivé řádky zabírají nejmenší nutné množství bloků (s ohledem na umístění v souboru)



restrikce s použitím indexu

- restrikce $\sigma_{a=v}(R)$ a předpokládáme, že pro tabulku R a atribut a existuje shlukovaný index
- náročnost operace je přibližne $\frac{B(R)}{V(R,a)}$, kde $V(R,a)$ je počet různých hodnot atribut a v tabulce R
- hodnota může být větší, protože
  - řádky mohou přesahovat hranice bloků,
  - je potřeba uvážit čtení indexu,
  - podíl je nutné zaokrouhlit směrem nahoru.

Restrikce s použitím indexu

restrikce neshlukovaný index

- musíme předpokládat, že každý řádek bude vyžadovat čtení samostatného bloku
- počet I/O operací je pak přibližně : $\frac{T(R)}{V(R,a)}$, kde $T(R)$ označuje počet řádků relace
- hodnota ovlivňují stejné faktory jako v případě shlukovaného indexu
- reálně se dá předpokládat, že některé bloky již budou v paměti a nebudou se číst opakovaně

další možnosti

- pokud je index postavený na B+stromech operaci index-scan lze použít pro vyhledávání
- na základě nerovnosti nebo rozsahu, např. $\sigma_{(a \geq v)}(R)$ nebo $\sigma_{(a \geq v_1) \wedge (a \leq v_2)}(R)$
- složitejší podmínku rozdělit a provést nejdříve index-scan a pak restrikci, tj. $\sigma_{(a=v) \wedge \theta}(R) = \sigma_{\theta}(\sigma_{a=v}(R))$

Další operace s použitím indexu

- algoritmy pro množinové operace a unární operace přímočaré
- spojení, pokud máme index řadící řádky vhodným způsobem (B-strom), můžeme použít dříve popsané algoritmy (bez řadícího mezikroku)

spojení (varianta nested-loop join s indexem)

- tabulky $R(XY)$ a $S(YZ)$, tabulka S má index přes atributy Y

```
for each block b of R:
   read block to memory
   for each (tuple) t in b:
      using index find tuples u in S such that u(y) = t(y)
      for each u:
          output the join of t and u
```

- pro každý řádek z R musíme v průměru načíst $\frac{T(S)}{V(S,Y)}$ řádků z S
- z toho plyne počet I/O operací pro:
  - shlukovaný index nad S: $\frac{T(R)B(S)}{V(S,Y)}$
  - neshlukovaný index nad S: $\frac{T(R)T(S)}{V(S,Y)}$
- operace s R můžeme zanedbat

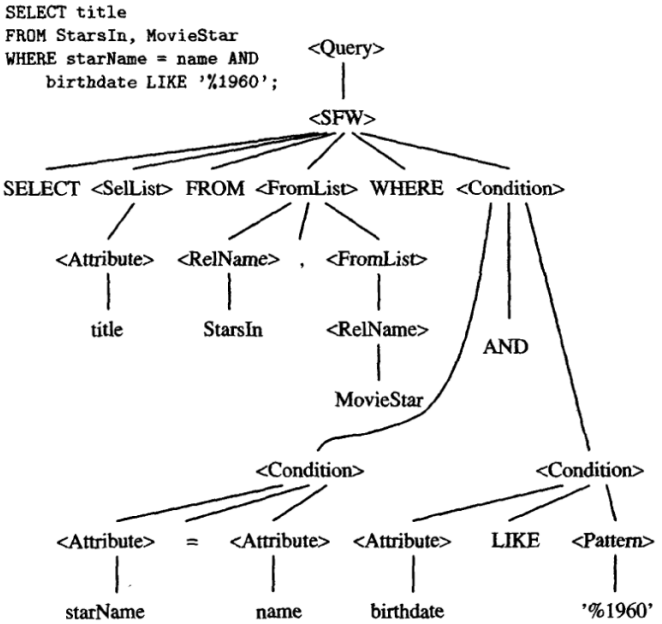
Správa bufferů

- koncepčně dvě řešení:
  (1) správu bufferů (uvolnění, přesun na disk) si řídí přímo SŘBD (tradiční databáze),
  (2) správu bufferů řídí OS, tj. buffery jsou namapovány do virtuální paměti a OS obsluhuje přesun z/na disk (main-memory databáze).
- v obou případech se používají podobné algoritmy LRU, FIFO, algoritmus druhé šance (hodinový algoritmus)
- (2) je snadnější na implementaci (nedubluje se činnost OS)
- v případě (1) má SŘBD lepší přístup k informacím a garance chování při správě paměti.
- vhodné uvažovat při implementaci algoritmů
- např. nested-block-join
  - při použití LRU nebo hodinového algoritmu
  - může docházet k uvolňování dat, které budou potřeba
  - strategičtější procházet data ve vnitřní smyčce v obou směrech

Vytvoření logického plánu

- odehrává se v několika krocích
- vytvoření logického plánu

1. parser
   - z textové reprezentace dotazu vytvoří syntaktický strom
   - atomy – klíčová slova, identifikátory, konstanty, relační operátory, spojky, ...
   - syntaktické kategorie – jména označující podčásti dotazu mající podobný význam (podmínka, tabulka, ...)
2. preprocessor
   - sémantická kontrola dotazu
   - kontrola správného použití názvů tabulek (resp. pohledů)
   - kontrola použití atributů (jestli jsou dané atributy dostupné v daném rozsahu a použity jednoznačně)
   - kontrola typů (zde jsou korektně použity hodnoty a operátory, např. nelze použít LIKE a číslo)
3. vygenerování počátečního logického plánu
   - převod syntaktického stromu do rel. algebry
4. přepis logického plánu
   - optimalizace na základě vlastností rel. algebry


SELECT title
FROM StarsIn, MovieStar
WHERE starName = name AND
birthdate LIKE '%1960';
<Query>
<SFW>
SELECT
<SelList>
FROM
<FromList>
WHERE
<Condition>
<Attribute>
<RelName>
,
<FromList>
title
StarsIn
<RelName>
MovieStar
AND
<Condition>
<Condition>
<Attribute>
=
<Attribute>
<Attribute>
LIKE
<Pattern>
starName
name
birthdate
'%1960'

Pravidla používaná při tvorbě logického plánu

- vychází z dobře známých vlastností množinových a relačních operací
- pro multimnožiny nemusí některé zákony platit
- např. neplatí zákon distributivity
- např. pro $A = B = C = \{x\}$ platí
  - $A \cap (B \cup C) = \{x\}$ (průnik odpovídá minimu)
  - $(A \cap B) \cup (A \cap C) = \{x, x\}$
- příznak `NULL` nabourává pravidla rel. algebry
- apř. $\mathcal{D} \bowtie \mathcal{D} = \mathcal{D}$ nebo $\mathcal{D}_1 \bowtie \mathcal{D}_2 = \mathcal{D}_1 \cap \mathcal{D}_2$, kde $\mathcal{D}, \mathcal{D}_1, \mathcal{D}_2$ jsou relace nad relačním schématem R.

```
CREATE TABLE foo (a INTEGER PRIMARY KEY, b INTEGER);
INSERT INTO foo (a, b) VALUES (1, 10), (2, 20), (3, NULL);

SELECT * FROM foo f1
         NATURAL JOIN foo f2;

 a | b
---+----
 1 | 10
 2 | 20
```

```
SELECT * FROM foo
INTERSECT SELECT * FROM foo;

 a | b
---+----
 1 | 10
 2 | 20
 3 |
```

Pravidla používaná při tvorbě logického plánu

komutativní a asociativitní operace

- $\times, \cup, \cap, \bowtie$
- théta join $\bowtie_\theta$ je obecně komutativní ale nikoliv asociativní, např. pro relace $R(ab), S(bc), T(cd)$ by mělo platit:

$$(R \bowtie_{R.b>S.b} S) \bowtie_{a<d} T = R \bowtie_{R.b>S.b} (S \bowtie_{a<d} T),$$

ale pravá strana rovnosti nedává smysl, protože a není součástí relačního schématu S ani T

Pravidla používaná při tvorbě logického plánu

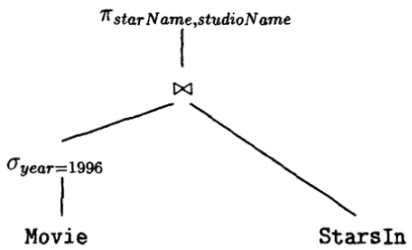
Pravidla pro restrikce

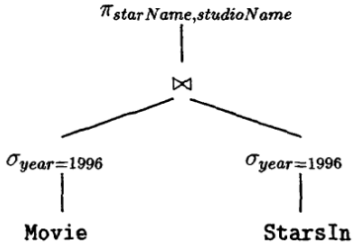
- $\sigma_{\theta_1 \wedge \theta_2}(R) = \sigma_{\theta_1}(\sigma_{\theta_2}(R))$
- $\sigma_{\theta_1 \vee \theta_2}(R) = \sigma_{\theta_1}(R) \cup \sigma_{\theta_2}(R)$
- $\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$
- $\sigma_{\theta}(R \cup S) = \sigma_{\theta}(R) \cup \sigma_{\theta}(S)$
- $\sigma_{\theta}(R - S) = \sigma_{\theta}(R) - \sigma_{\theta}(S)$ nebo $\sigma_{\theta}(R - S) = \sigma_{\theta}(R) - S$
- pokud θ obsahuje jen atributy z jedné relace (v našem případě R) platí:
- $\sigma_{\theta}(R \times S) = \sigma_{\theta}(R) \times S$
- $\sigma_{\theta}(R \bowtie S) = \sigma_{\theta}(R) \bowtie S$
- $\sigma_{\theta}(R \bowtie_{\theta'} S) = \sigma_{\theta}(R) \bowtie_{\theta'} S$

Použití pravidel pro restrikce

- ukazuje se, že je efektivní přesunout restrikce co nejníže ve stromě
- méně dat ke zpracování, možné použít index-scan
- jsou situace, kdy je vhodné přesunout restrikce co nejvýše a pak zpět co nejníže

```
CREATE TABLE StarsIn (title ..., year ..., starName);
CREATE VIEW MoviesOf1996 AS SELECT title, year, length, studioName
   FROM movie WHERE year= 1996;
SELECT starName, studioName FROM MoviesOf1996
   NATURAL JOIN StarsIn;
```





- atribut year je společný (dává smysl aplikovat restrikci na oba podvýrazy)

Pravidla používaná při tvorbě logického plánu

Pravidla pro projekce

- projekci lze vložit na libovolné místo výrazu
- pokud neeliminuje atribut, který bude použit výše ve stromu nebo jako výsledek dotazu
- projekce lze přesunout níže do stromu podobně jako restrikce (efekt omezený)

Pravidla pro spojení a kartézský součin

- $R \bowtie_\theta S = \sigma_\theta(R \times S)$
- $R \bowtie S = \pi_L(\sigma_\theta(R \times S)$
- kde θ představuje podmínku srovnávající společné atributy a L je sjednocení množin atributů relačních schémat R a S.

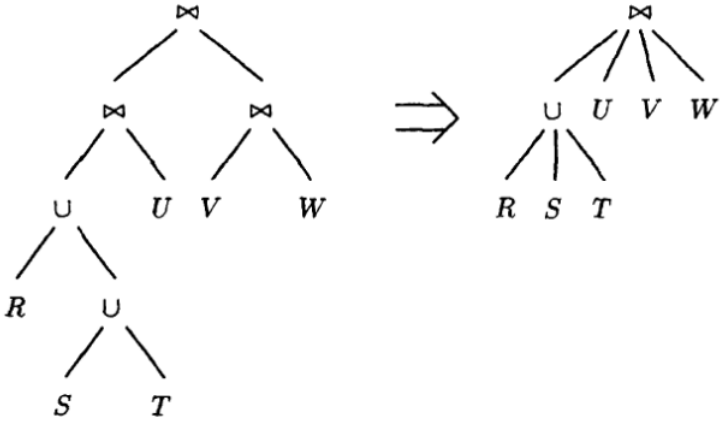
Pravidla pro eliminaci duplicit

- $\delta_\theta(R \times S) = \delta_\theta(R) \times \delta(S)$
- $\delta(R \bowtie S) = \delta(R) \bowtie \delta(S)$
- $\delta(R \bowtie_\theta S) = \delta(R) \bowtie_\theta \delta(S)$
- $\delta(\sigma_\theta(R) = \sigma_\theta(\delta(R))$
- pro množinové operace nemá δ smysl, tj. $\delta(R \cup S) = R \cup S$
- podobně pro seskupení: $\delta(\gamma_L(R)) = \gamma_L(R)$

Vytvoření logického plánu z dotazu

- pokud existuje ekvivalent části dotazu v rel. algebře, je přímo převeden
  - `SELECT` $\rightarrow \pi$ (projekce)
  - `FROM` $\rightarrow \times$ (kartézský součin)
  - `WHERE` $\rightarrow \sigma$ (restrikce)
- některé části dotazu nemají ekvivalent (vnořené dotazy) – nutné transformovat na rel. algebru (s použitím spojení, kartézského součinu, ...)
- následují optimalizace
  - přesun restrikcí směrem k listům
  - přesun nebo vložení projekcí směrem k listům
  - odstranění operace δ nebo přesun na vhodnější místo
  - konverze restrikce a kartézského součinu na (théta) spojení
- konverze komutativních a asociativních binární operací ne n-ární (technické vylepšení)

Převod binárních operací na n-ární



Poznámka

Všechny obrázky:
Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database System Implementation.