Základné výpočty relačných operátorov

Idea výpočtu dopytov

- Dopyt (select) sa sparsuje a rozbije na základné operácie relačnej algebry a relačného kalkulu (Systém R)
- Analyzujú sa možnosti, ako sa dá daný dopyt vyrátať (často je viacero spôsobov, ako sa dostať k rovnakému výsledku)
- Vytvorí sa **plán dopytu** = strom relačných operátorov s výberom algoritmu pre každý operátor.
- Dve hlavné otázky optimalizácie dopytov
  - Aké plány DBMS zvažuje?
  - Aké sú odhadované náklady na výpočet plánov?
- Ideál: Chceme nájsť najlepší plán.

  Prakticky: Chceme sa vyhnúť najhorším plánom

Výpočet relačných operátorov

- Základné operátory:
  - Selekcia
  - Projekcia
  - Spojenie (Join)
- Množinové operácie
  - Rozdiel
  - Zjednotenie
  - Prienik
- Agregácia (už rel. kalkul)
  - Minimum, maximum,...
  - Group by

Výpočet relačných operátorov

- Môžeme mať jednu (bez joinov) alebo viac relácii (s joinami)
- Každý operátor sa dá vyrátať rôznymi spôsobmi
- Každý relačný operátor má na vstupe aj výstupe relácie
- Každý relačný operátor je typicky implementovaný ako postupný producent riadkov výslednej relácie.
    - Rovnako aj vstup mnohých operátorov sa dá realizovať ako postupný konzument riadkov vstupnej relácie
    - Pri výpočtoch ceny operácií nerátame s uložením výsledku na disk

Systémový katalóg

- Obsahuje informácie o veľkosti buffer poole, veľkosti stránky a základné informácie o reláciách, indexoch a pohľadoch.
- Metainformácie typicky obsahujú aspoň:
  - štruktúra súboru, integritné obmedzenia
  - počet riadkov(NTuples) a počet stránok (NPages) pre každú reláciu
  - počet kľúčov na indexoch (NKeys) a NPages pre každý index
  - výška indexu, najmenšia/najväčšia hodnota kľúča pre každý index.
- Môže obsahovať aj nejaké štatistiky týkajúce sa relácii a indexov, napr. histogramy.
- Nemáme rovnako podrobné informácie o všetkých tabuľkách
  - Ak nemáme index, nemáme rozsah hodnôt
- Tieto štatistiky sa aktualizujú z času na čas, nie stále keď sa zmení daná relácia.
- Sú tu uložené aj informácie o používateľoch a ich prístupové práva

Zaužívané techniky

- Algoritmus na evaluáciu relačných operátorov používa niekoľko jednoduchých ideí, hlavne:
  - Indexovanie: Na získanie podmnožiny riadkov spĺňajúcich podmienku, ale aj na získavanie usporiadaných dát
  - Iterácia: Niekedy je rýchlejší table scan, akoby sme mali použiť index (a niekedy môžeme iterovať index)
  - Delenie dát: Využitím triedenia alebo hashovania môžeme vstupné riadky rozdeliť napr. na menšie dáta, ktoré vojdú do pamäte a každú časť rátať samostatne

Selekcia

- Musím získať tie riadky, ktoré spĺňajú podmienku
- Podľa organizácie súboru existujú nasledujúce možnosti:
  - Bez indexu, neusporiadané dáta
    - Table scanom musíme prejsť všetko
  - B+ strom
    - Dá sa použiť iba ak podmienka zahŕňa tie atribúty, ktoré sú prefixe vyhľadávacieho kľúča
      - Napr. ak je kľúč <a,b,c> je použiteľný pre podmienky a=5 AND b=3, aj a=5 AND b>6, ale nie b=3
  - Hash index je veľmi rýchly pri podmienkach s rovnosťou (pre rozsah sa nepoužíva)
    - Napr. ak je kľúč <a,b,c> je použiteľný pre podmienky a=5 AND b=3, ale nie pre a>5 AND b=6, ani b=3

Running example

- Predavači (pid:int; pmeno:string; rating:real, vek:int)
  - Riadok 50 bajtov
  - Stránka p_P=80 záznamov
  - Počet stránok: N = 500
  - Záznamov: 40 000
- Objednávky(pid:int; zid:int; deň:date; názov: string)
  - Riadok 40 bajtov
  - Stránka p_O=100 záznamov
  - Počet stránok: M = 1000
  - Záznamov: 100 000

Selekcia

*SELECT * FROM objednávky WHERE názov=„tehla"*

- Nech je odhadovaná selektivita 1%
- Ak máme heap a nemáme index na *názov*, použijeme table scan (1000 I/O)
- Ak by sme mali utriedený súbor podľa názvu: nájdenie ($\log_2 1000$) ≈ 10 I/0
  - Vrátenie: nájdenie + 10 I/0 = 20 I/O
- Klastrovaný Hash nájdenie cca 1,2 I/0
  - Vrátenie: nájdenie + 10 I/0 = 11 I/O

Selekcia

*SELECT * FROM objednávky WHERE názov=„tehla"*

- Klastrovaný B+strom nájdenie cca 3 I/0
  - vrátenie: nájdenie + 10 I/0 = 13 I/O

- Neklastrovaný B+strom
  - 1 list stromu cca 200 záznamov typu (názov, rid), 1% listov = cca 5 stránok
  - 100 záznamov na stránku s riadkami,
  - Hlúpe riešenie, keď pre každé rid máme 1 I/0
    - 3+5 + (5*200) =1008 I/0
  - Ak utriedim listové záznamy podľa rid, pôjdem na každú stránku s riadkami najviac raz (10 až 1000 stránok)
    - 3+5 + (10 až 1000) = 18 až 1008 I/0
    - Najvyššia pravdepodobnosť: cca 634 stránok s dobrým riadkom : 652 I/0

Normalizácia podmienok v selekciách

- Podmienky selekcie sa najprv prevedú do *konjunktívnej normálnej formy (CNF)*:

(deň<‘8.9.1994’ AND názov=‘tehla’) OR pid=5 OR zid=3

(deň <‘8.9.1994’ OR pid=5 OR zid=3) AND (názov=‘tehla’ OR pid=5 OR zid=3)

- Potom sa hľadá index, ktorý vyhovuje podmienke v CNF
- Pravidlá kedy index vyhovuje podmienke v CNF bez disjunkcie:
  - Hash index – ak existuje výraz tvaru atribút=hodnota pre každý atribút vo vyhľadávacom kľúči indexu.
  - Stromový index – ak existuje výraz tvaru atribút?hodnota pre každý atribút v prefixe vyhľadávacieho kľúča indexu.
    - ?-ľubovoľný porovnávací operátor
    - <a> a <a,b> sú prefixmi <a,b,c>, ale <a,c> a <b,c> nie

Vyhodnotenie selekcie bez disjunkcie

- Nájdi najselektívnejšiu prístupovú cestu, získaj riadky pomocou nej a aplikuj všetky výrazy, ktoré si touto cestou neoveroval:
  - *Najselektívnejšia prístupová cesta:* Index alebo table scan, o ktorom predpokladáme, že bude vyžadovať najmenej I/O operácií.
  - Použijeme najväčšiu použiteľnú časť podmienky pre danú prístupovú cestu, čím získame riadky, ktoré ju spĺňajú. Zvyšok podmienky použijeme na vyradenie tých riadkov, ktoré ju nespĺňajú, ale už nemá efekt na počet získaných riadkov/stránok
  - Uvažujme *deň<'8.9.1994' AND pid=5 AND zid=3* . Môžeme napr. použiť B+ tree na *deň* ; potom musíme otestovať (*pid=5* AND *zid=3)* pre každý získaný riadok. Podobne môžeme použiť hash index na <*pid, zid*> a potom musím otestovať *deň <'8.9.1994'.*

Selekcia s disjunkciou

- Môže sa použit table scan alebo index
- Na disjunkciu sa jeden index použiť nedá
  - Ak máme napr. (*a*=3 OR *b*>4) AND *c*=5 a máme indexy pre atribút *a* alebo *b*, alebo hoc aj <*a*,*b*>, ale nemáme index pre atribút *c*, databáza použije table scan.
  - Ak máme index pre *a* aj *b*, vieme spraviť to, že získame všetky riadky spĺňajúce *a*=3 a všetky riadky spĺňajúce *b*>4 a spravíme z nich zjednotenie a nad týmto zjednotením nakoniec spravíme selekciu na podmienku *c*=5
    - Táto možnosť sa použije iba v prípade, že predpokláme výraznú selektivitu výrazu (*a*=3 OR *b*>4) nad danou tabuľkou.
    - pozn: o tom, ako spraviť zjednotenie alebo prienik, povieme neskôr

Zjednotenie a prienik a podľa rid

- V prípade, že index nad *a* aj nad *b* sú neklastrované, robíme zjednotenie iba indexových záznamov podľa *rid.*
  - Následne sa vrátia riadky tabuľky, ktoré výsledné *rid* majú a overí sa zvyšok podmienky
  - Tieto *rid* sa dajú zotriediť, aby sme nechodili na niektoré stránky viac krát
- Prienik podľa *rid* je oveľa pravdepodobnejšia operácia (pri konjunkcii)
  - Každý ďalší výraz konjunkcie zmenšuje výslednú množinu
    - Robíme prienik
  - Každý ďalší výraz disjunkcie zväčšuje výslednú množinu
    - Robíme zjednotenie

Projekcia s DISTINCT založená na triedení

```
SELECT DISTINCT zid, názov FROM Objednávky
```

- Prejdi tabuľku a vytvor množinu riadkov, ktoré obsahujú iba potrebné riadky. Získame menšie riadky, ako sú v pôvodnej tabuľke.
- Usporiadaj túto množinu podľa všetkých jej atribútov. Duplicitné riadky budú pod sebou.
- Prejdi túto množinu a odstráň duplicity. Získame výsledok.
- Možnosť vykonania prvého a druhého kroku v jednom prechode.
- Cena: V 0-tom prechode prečítame pôvodnú reláciu (1000 R), zapíšeme rovnako veľa riadkov, ale menšie (napr. 500 W). Pri spájaní sa postupne zapisuje stále menej riadkov pre každý prechod
  - Ak utriedim na 2 prechody: $(1000\ R + 500\ W)_0 + (500\ R + 300\ W)_1 + (300\ R)_2 = 2600$ I/O
    - Ďalšie duplicity spôsobia, že výsledok môže mať napr. 250 stránok
  - Ak utriedim na 1 prechod: $(1000\ R + 500\ W)_0 + (500\ R)_1 = 2000$ I/O

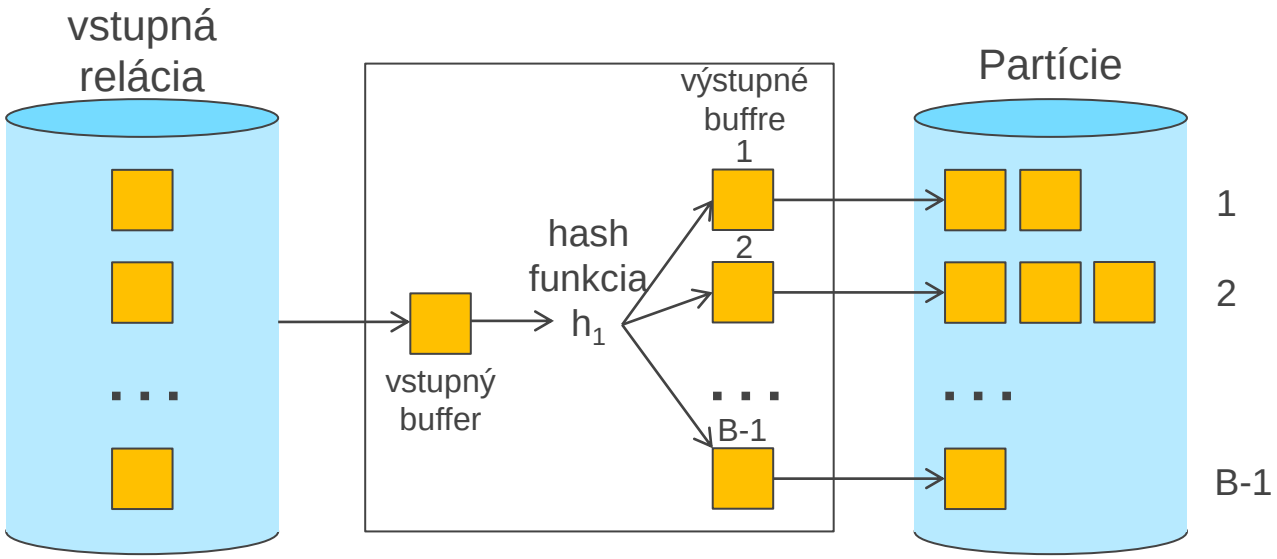
Projekcia s DISTINCT založená na triedení

- Pamäťové nároky
  - Aby som to zvládol na jeden prechod potrebujem buffer B aspoň veľkosti cca $\sqrt{R}$, kde R je počet stránok s orezanými riadkami
  - V našom príklade B ≥ $\sqrt{500}$ = 22,3
  - Lebo $\frac{R}{B} \leq B - 1$
  - V nultom prechode vyrobíme $\frac{R}{B}$ utriedených postupností, a tých nesmie byť viac ako B-1 vstupných buffrov
  - Po vypočítaní kvadratickej nerovnice je to presnejšie B ≥ $\sqrt{522}$ = 22,8

Projekcia s DISTINCT založená na hashovaní

- *Deliaca fáza:* Prečítať vstupnú reláciu použitím vstupného buffra. Pre každý riadok odstrániť neželané stĺpce, aplikovať hash funkciu h_1
    - h_1 – funkcia, ktorá rovnomerne rozdelí riadky do B-1 partícií
    - Duplicity sa dostanú do rovnakej partícií, keďže majú rovnaký hash
- *Eliminačná fáza:* Každú partíciu načítať do pamäte po jednom a vytvoriť HashSet použitím hashovacej funkcie h_2 (rôznej od h_1) na všetky riadky, čím sa odstránia duplicity. Výsledok zapísať, vyčítať ďalšiu partíciu na miesto predošlej
    - Ak sa partícia nezmestí do pamäte, môžeme na ňu použiť rekurzívne deliaci algoritmus, pomocou ďalšej hasovacej funkcie
- Cena: Prečítame celú tabuľku (1000 R), zapíšeme redukované riadky (napr. 500 W), tieto postupne prečítame a riadky bez duplicít posielame na výstup (500 R) = 2000 I/0

Projekcia s DISTINCT založená na hashovaní



Projekcia s DISTINCT

- Prístup založený na triedení lepšie pracuje s duplicitami a šikmými dátami. Výhodou je, že výsledok je navyše usporiadaný.
- Ak index na relácii obsahuje všetky požadované atribúty v svojom kľúči, možeme robiť iba čítanie stránok indexu
  - Aplikujeme techniky projekcie na listové data
- Ak utriedený index obsahuje všetky požadované atribúty ako prefix kľúča, možeme spraviť lepšie:
  - Získať listové dáta usporiadané (index-only scan), už pri čítaní odstraňovať duplicity a generovať priamo výstup

Implementácie spájania JOIN

Running example

- Predavači (pid:int; pmeno:string; rating:real, vek:int)
  - Riadok 50 bajtov
  - Stránka p_P=80 záznamov
  - Počet stránok: N = 500
  - Záznamov: 40 000
- Objednávky(pid:int; zid:int; deň:date; názov: string)
  - Riadok 40 bajtov
  - Stránka p_O=100 záznamov
  - Počet stránok: M = 1000
  - Záznamov: 100 000

Spojenia s rovnosťou

```
SELECT * FROM Objednávky O, Predavači P WHERE O.pid=P.pid
```

- 2 hlavné rozdelenia:
  - Prechádzajú všetky riadky karteziánskeho súčinu a vyhodia tie, ktoré nevyhovujú podmienke:
    - Nested loops join
    - Block tested loops join
  - Bez prechádzania všetkých riadkov karteziánskeho súčinu:
    - Index nested loops join
    - Sort-merge join
    - Hash join

Nested loops join - riadky

```
foreach tuple o in O do
    foreach tuple p in P do
        if o.pid == p.pid then add <o, p> to result
```

- Pre každý riadok vo vonkajšej relácii O, čítame celú reláciu P
  - Nech prechod O potrebuje M I/O. S prečítame p_O * M-krát a každý prechod P potrebuje N I/O.
  - Cena: M + p_O * M * N.
  - Ak M=1000, p_O = 100, N = 500, tak cena je 1000 + (100*1000)*500 = 50 001 000 I/O. Ak stránku prečítame za 10 ms, bude to trvať viac ako 138 hodín

Nested loops join - riadky

```
foreach tuple o in O do
    foreach tuple p in P do
        if o.pid == p.pid then add <o, p> to result
```

- Pre každý riadok vo vonkajšej relácii O, čítame celú reláciu P
  - Nech prechod O potrebuje M I/O. S prečítame p_O * M-krát a každý prechod P potrebuje N I/O.
  - Cena: M + p_O * M * N.
  - Ak M=1000, p_O = 100, N = 500, tak cena je 1000 + (100*1000)*500 = 50 001 000 I/O. Ak stránku prečítame za 10 ms, bude to trvať viac ako 138 hodín
- Vylepšenie: vymeniť vonkajšiu a vnútornú reláciu
  - Cena = 500 + (80*500)*1000 = 40 000 500 I/O. Ak stránku prečítame za 10 ms, bude to trvať viac ako 111 hodín.

Nested loops join - stránky

```
foreach page r in O do
    foreach page s in P do
        foreach tuple r in o do
            foreach tuple s in p do
            if o.pid == s.pid then add <r, s> to result
```

- Pre každú *stránku* z O, dostaneme každú *stránku* z P, a vypíšeme zodpovedajúce dvojice riadkov <r, s>
  - Cena: M + M*N.
  - 1000 + 1000*500 = 501 000 I/O. Ak stránku prečítame za 10 ms, bude to trvať už „len“ 1.4 hodín, čiže 83,5 minút.

Nested loops join - stránky

```
foreach page r in O do
    foreach page s in P do
        foreach tuple r in o do
            foreach tuple s in p do
            if o.pid == s.pid then add <r, s> to result
```

- Pre každú *stránku* z O, dostaneme každú *stránku* z P, a vypíšeme zodpovedajúce dvojice riadkov <r, s>
  - Cena: M + M*N.
  - 1000 + 1000*500 = 501 000 I/O. Ak stránku prečítame za 10 ms, bude to trvať už „len“ 1.4 hodín, čiže 83,5 minút.
- Vylepšenie: ak vymeníme vonkajšiu a vnútornú reláciu
  - Cena = 500 + 500*1000 = 500 500 I/O. Ušetríme oproti predchádzajúcemu 5 sekúnd!

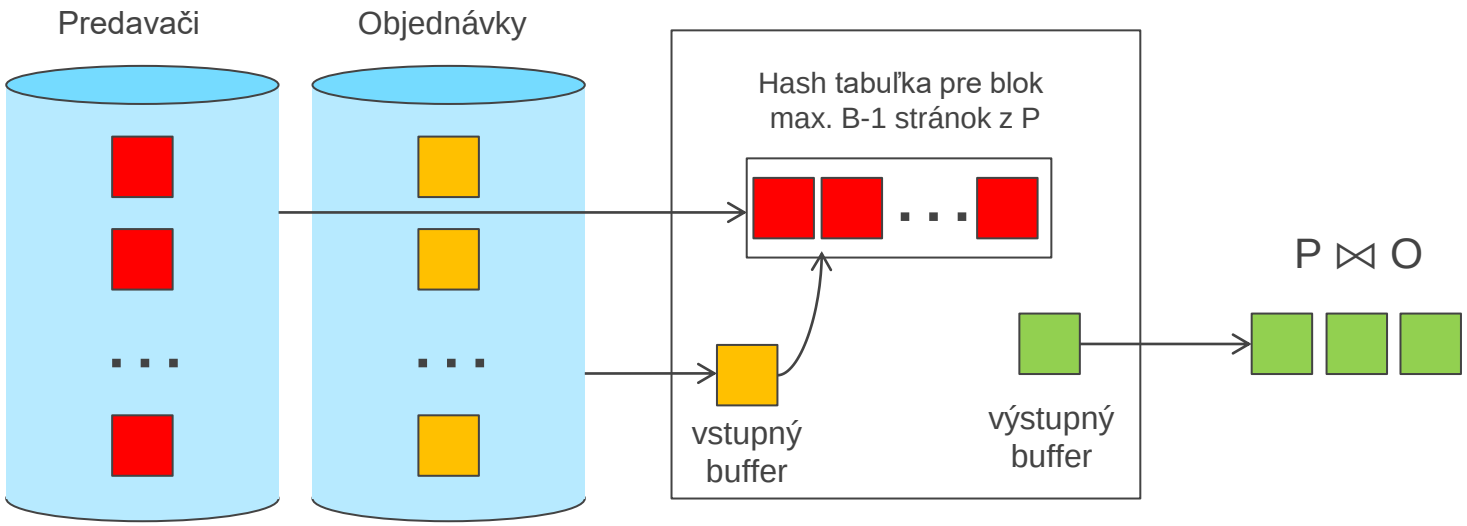
Index nested loop join

foreach tuple o in O do
    find all tuples [p_1,..,p_x] in index(P) having o.pid == p_i.pid
        for all i∈[1,..,x] add <o, p_i> to result

- Ak je index na joinovanom stĺpci na jednej z relácií, tak ju použijeme ako vnútornú.
- Cena nájdenia vhodných riadkov z P závisí od typu indexu a počtu vyhovujúcich riadkov:
  - B+strom *c* = 2-4 I/O, Hash index *c* = 1,2 I/O.
  - neklastrovaný index ešte o 1 prístup viac
- Cena: M + (M*p_O) * c = 1000 + (1000*100) * c = 101 000 * c I/O
- Ak vymeníme relácie (a máme nad P.pid index)
  - Cena: N + (N*p_P) * c = 500 + (500*80) * c = 40 500 * c I/O
  - Klastrovaný hash: 40500 * 1,2 = 48600 I/O ≈ 8 minút
- Veľmi výkonná metóda, ak vonkajšia relácia je veľmi malá!

Block nested loops join

- Ak by bolo dosť pamäte pre načítanie celej menšej relácie do pamäte s 2 buffer stránkami navyše (prvá pre väčšiu reláciu a druhá slúži ako výstupný buffer), tak cenu by sme mohli znížiť na M + N.
- Ak nemáme, tak menšiu reláciu môžeme rozdeliť do B-2 blokov, kde B je počet dostupných buffrových stránok.
- Pre každý taký blok čítame vnútornú reláciu a hľadáme zhodu v hashovacej tabuľke. Vyhovujúce dvojice posielame na výstup.



Block nested loops join

- B = 502
  - Ak O ⋈ P: 1000 + 2*500 = 2000 I/O ≈ 20 sekúnd
  - Ak P ⋈ O: 500 + 1000 = 1500 I/O (optimum) ≈ 15 sekúnd
- B = 102
  - Ak O ⋈ P: 1000 + 10*500 = 6000 I/O ≈ 60 sekúnd
  - Ak P ⋈ O: 500 + 5*1000 = 5500 I/O ≈ 55 sekúnd
- B = 37
  - Ak O ⋈ P: 1000 + 29*500 = 15500 I/O ≈ 2,5 minúty
  - Ak P ⋈ O: 500 + 15*1000 = 15500 I/O ≈ 2,5 minúty
- Vo všeobecnosti cena = $M + \left\lceil \frac{M}{B-2} \right\rceil * N$ I/0, kde M je počet stránok vonkajšej relácie a N je počet stránok vnútornej relácie

Sort-Merge join

- Usporiadame O a P na atribútoch v joine (čím sa vytvoria partície), potom ich spojíme
- Stačí teda porovnávať riadky z O a P, ktoré sú v rovnakých partíciach
- Po usporiadaní spájanie prebieha nasledovne:
  - Ak o.pid > p.pid tak čítaj riadky o relácie O
  - Ak p.pid > o.pid tak čítaj riadky p relácie P
  - Ak p.pid = o.pid všetky riadky s rovnakou hodnotou z O aj z P navzájom po dvojiciach pospájaj a pošli <o,p> na výstup
  - Potom pokračujeme ďalším čítaním O a P
- Obe usporiadané relácie sú čítané práve raz.

Sort-Merge join

- Ak relácie boli utriedené (stačí B ≥ 3):
  - O ⋈ P alebo P ⋈ O: 1000 + 500 = 1500 I/O (optimum) ≈ 15 sekúnd
- B = 502
  - Triedenie O (2 behy): 1000 R + 1000 W + 1000 R + 1000 W = 4000 I/O
  - Triedenie P (1 beh) : 500 R + 500 W = 1000 I/O
  - O ⋈ P alebo P ⋈ O: 5000 + 1500 = 6500 I/O ≈ 65 sekúnd

Sort-Merge join

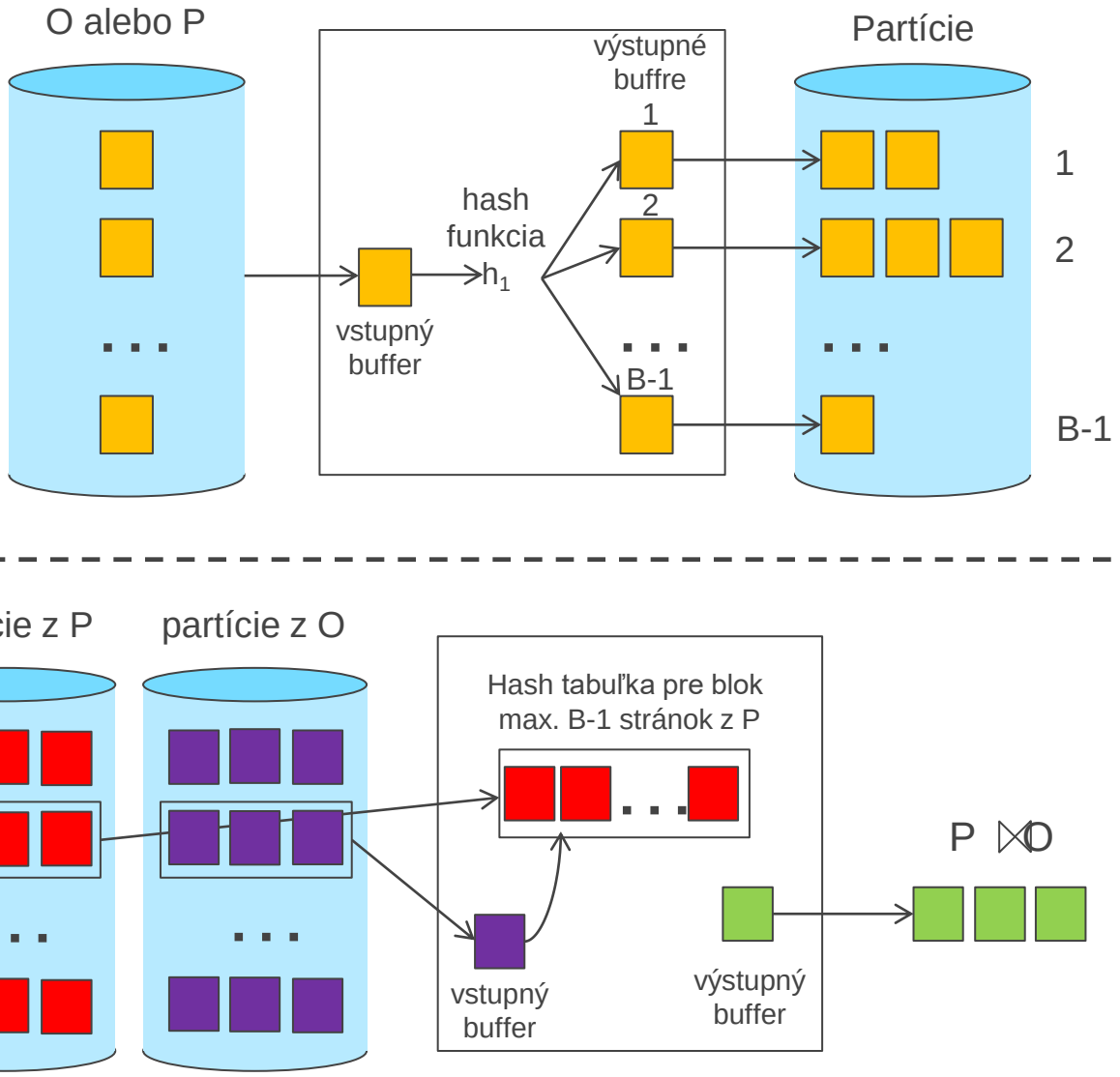
- B = 102
  - Triedenie O (2 behy): 1000 R + 1000 W + 1000 R + 1000 W = 4000 I/O
  - Triedenie P (2 behy) : 500 R + 500 W + 500 R + 500 W = 2000 I/O
  - O ⋈ P alebo P ⋈ O: 6000 + 1500 = 7500 I/O ≈ 75 sekúnd
- B = 37
  - Triedenie O (2 behy): 1000 R + 1000 W + 1000 R + 1000 W = 4000 I/O
  - Triedenie P (2 behy) : 500 R + 500 W + 500 R + 500 W = 2000 I/O
  - O ⋈ P alebo P ⋈ O: 6000 + 1500 = 7500 I/O ≈ 75 sekúnd

Vylepšený Sort-Merge join

- Ak $B \geq \lceil\sqrt{M}\rceil + \lceil\sqrt{N}\rceil$
- Najprv urobíme 0-tú fázu pre obe relácie, čím nám vznikne $\left\lceil\frac{M}{B}\right\rceil$ postupností pre reláciu O a $\left\lceil\frac{N}{B}\right\rceil$ postupností pre reláciu P.
- Z oboch skupín postupností generujeme utriedené prúdy dát, ktoré spojíme už rovnako ako klasický Sort-Merge join
- Cena:
  - 0-tý beh O : 1000 R + 1000 W = 2000 I/O
  - 0-tý beh P : 500 R + 500 W = 1000 I/O
  - O ⋈ P alebo P ⋈ O: 3000 + 1500 = 4500 I/O ≈ 45 sekúnd

Hash join

- Na vyrobenie partícií oboch relácií použijeme rovnakú hash funkciu h_1.
- Do partícii *i* sa dostanú iba tie riadky z O a P, ktoré môžu byť joinované.
- Ak máme dostatočne veľkú pamäť, tak O a P nám stačí prečítať iba raz.



Hash join

- Potrebná pamäť: $B \geq \sqrt{R * f}$, kde R je počet stránok vonkajšej relácie a *f* je koeficient zaplnenia oblastí (typicky 1,2)
  - B = 502
    - Delenie na partície O: 1000 R + 1000 W, R: 500 R + 500 W, dokopy = 4000 I/O
    - O ⋈ P: 4000 + 1500 = 6500 I/O ≈ 65 sekúnd
    - P ⋈ O: 500 + 1000 = 1500 I/O (optimum) ≈ 15 sekúnd (celé P vojde do pamäte)
  - B = 102
    - Delenie na partície rovnaké
    - O ⋈ P alebo P ⋈ O: 4000 + 1500 = 6500 I/O ≈ 65 sekúnd
  - B = 37 ($\sqrt{1000 * 1{,}2} = 34$)
    - Delenie na partície rovnaké
    - O P alebo P O: 4000 + 1500 = 6500 I/O ≈ 65 sekúnd

Hash join

- Počet particií k < B-1, a B-2 > veľkosť najväčšej particie vykonávaná v pamäti. Za predpokladu rovnomerne veľkých particií a maximálnom k, dostaneme
  - k= B-1, a M/(B-1) < B-2, tj., B musí byť >
- Ak využijeme v pamäti vstavanú hash tabuľku na zrýchlenie priraďovania riadkov, je potrebné využiť viac pamäte
- Ak hash funkcia nevytvára particie rovnomerne, jedna alebo viac R particií sa nezmestí do pamäte. Môžeme využiť hash-join techniku rekurzivne na vytvorenie joinu R particie s korešpondujúcou S particiou

Porovnanie join metód

metóda	B=1002	B=502	B=102	B=37	B=23
Index NL	48000	48000	48000	48000	48000
Block NL	?	1500	5500	15500	?
Sort-Merge	?	6500	7500	7500	?
Vylepšený SM	?	4500	4500	7500	?
Hash	?	1500	6500	6500	?

Všeobecné podmienky pre join

- Vyššie spomenuté algoritmy slúžili pre jednoduchý join (s jednou rovnosťou)
- Rovnosti pre viac atribútov (napr. *O.pid=P.pid* AND *O.omeno=P.pmeno*):
  - Pre *Index nested loop* môžeme vytvoriť index na <O.*pid, O.omeno*> a mať O ako vnútornú reláciu. Inou možnosťou je použiť existujúce indexy alebo *pid* alebo *pmeno.*
  - Pre Sort-Merge a Hash Join môžeme triediť/vytvoriť partície na kombinácii dvoch joinovaných stĺpcov.
  - Iné spomenuté algoritmy nie sú ovplyvnené.
- Podmienky nerovnosti (napr. *O.omeno < P.pmeno*):
  - Pre *Index nested loop* potrebujeme (klastrovaný!) B+strom na vnútornej relácii.
  - Hash Join, Sort Merge Join sú neaplikovateľné.
  - Iné spomenuté algoritmy nie sú ovplyvnené.

Množinové operácie

- Prienik – špeciálny prípad joinu (rovnosť na všetkých stĺpcoch)
- Kartéziansky súčin - špeciálny prípad joinu (bez podmienky pre join)
- Zjednotenie:
  - Triediaci prístup
    - Utriediť obe relácie na všetkých stĺpcoch.
    - Paralelne prechádzať R a S a spájať ich bez duplicít
  - Prístup založený na hash :
    - Vyrobíme partície pre obe relácie.
    - Vytvoríme v pamäti hash tabuľku.
    - Postupne čítame príslušnú partíciu z prvej relácie a posielame na výstup také riadky prvej relácie, ktoré nie sú v hash tabuľke.
- Rozdiel – Rovnaké možnosti ako pri zjednotení s menšou modifikáciou ak použijeme prístup založený na hash.

Agregačné operácie (avg, sum, min, count, ...)

- Výsledok je číslo, nie relácia
- Bez GROUP BY:
  - Niekedy máme odpoveď už v metadátach (počet riadkov, max. hodnota, min.hodnota)
  - Vo všeobecnosti vyžadujú prečítanie celej relácie a rátanie v extra pamäťových premenných
  - Ak máme index, ktorého kľuč obsahuje všetky atribúty v SELECTe alebo WHERE končtrukcii, môžeme robiť len index-only scan
- GROUP BY:
  - Zotriedime podľa GROUP BY atribútov, potom postupne čítam a rátam agregácie pre každú unikátnu hodnotu GROUP BY atribútov
  - Podobne môžem dať všetky riadky s rovnakou hodnotou GROUP BY atribútov dokopy pomocou vytvorenia partícií cez hash