

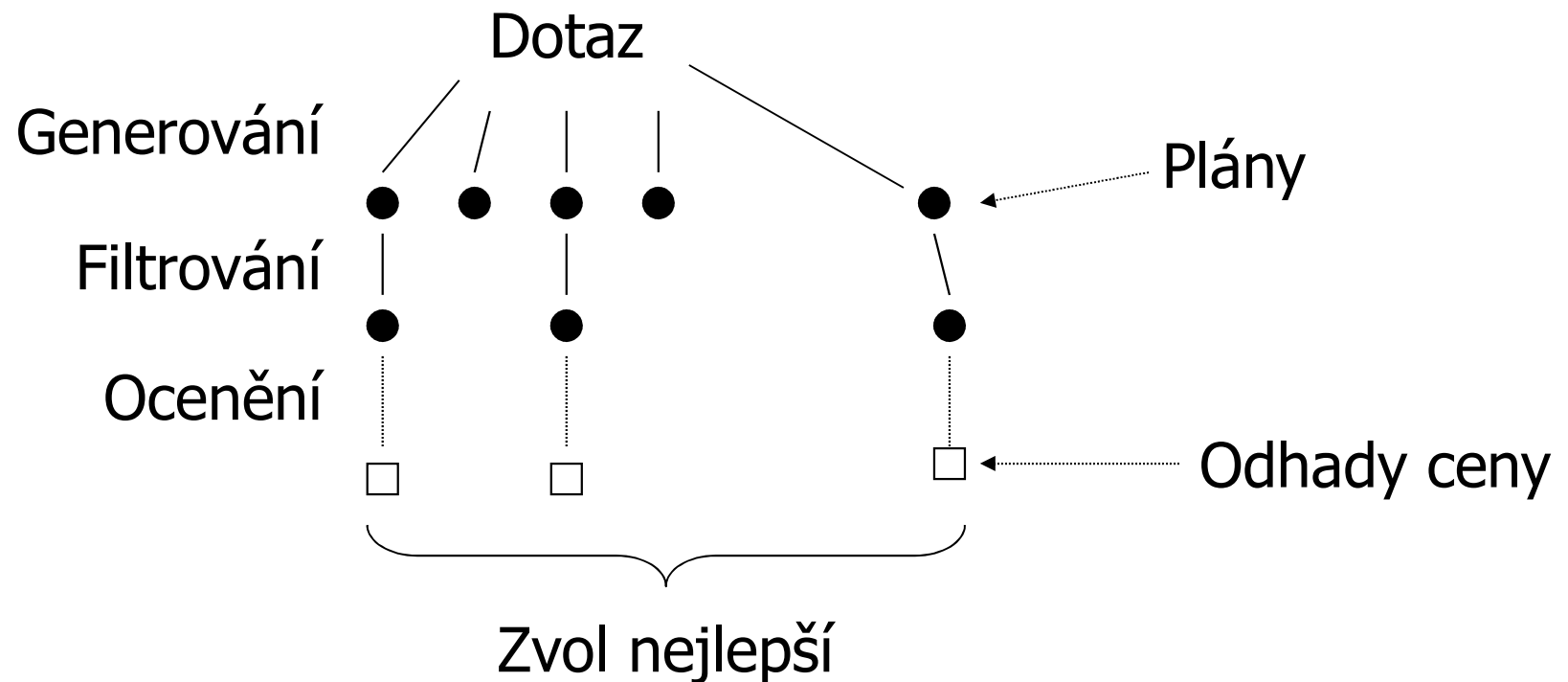


PA152: Efektivní využívání DB
8. Optimalizace dotazu

Vlastislav Dohnal

Optimalizace dotazu

- Generování a porovnávání plánů dotazu





Generování plánu dotazu

■ Zvážit používání:

- Transformační pravidla rel. algebry
- Implementace operací rel. algebry
- Použití existujících indexů
- Vytváření indexů a třídění podle potřeb



Odhad ceny plánu

- Závisí na ceně provedení každé operace
 - Tj. její implementaci
- Předpoklady ceny operace
 - Vstup se čte z disku
 - Výstup zůstává v operační paměti
 - Operace na CPU
 - CPU stačí počítat během čtení z disku
 - často zanedbány nebo zjednodušeny
 - Komunikace po síti
 - Počítat u distribuovaných databází
 - Ignorování vyrovnávacích pamětí mezi dotazy
- Odhad ceny operace
 - = počet čtení a zápisů z disku



Odhad ceny operace

■ Příklad nastavení PostgreSQL

<http://www.postgresql.org/docs/9.6/static/runtime-config-query.html#GUC-CPU-OPERATOR-COST>
<https://www.postgresql.org/docs/9.6/static/runtime-config-resource.html>

- seq_page_cost (1.0)
- random_page_cost (4.0)
- cpu_tuple_cost (0.01)
- cpu_index_tuple_cost (0.005)
- cpu_operator_cost (0.0025)

- shared_buffers (32MB) – ¼ RAM
- effective_cache_size (4GB) – ½ RAM
- work_mem (8MB)
 - Memory available to an operation



Odhad ceny operace

■ Parametry

- $B(R)$ – velikost relace R v blocích
- $f(R)$ – max. počet záznamů relace v bloku
- M – max. dostupná RAM v blocích (work_mem)

- $HT(i)$ – počet úrovní indexu i
- $LB(i)$ – celkový počet listových bloků indexu



Implementace operace

■ Použití konceptu **iterátor**

- *Open* – inicializace operace
 - příprava před vrácením řádků výsledku
- *GetNext* – vrácení dalšího řádku výsledku
- *Close* – ukončení operace
 - uvolnění dočasné paměti, ...

■ Výhody

- Výsledek nemusí být vygenerován „naráz“
 - Nezabírá paměť, nemusí být ukládán
- Operace lze řetězit (pipelining)

Operace čtení relace: **table scan**

■ Relace není prokládaná

R1 R2 R3 R4

R5 R6 R7 R8

...

- Čtení je $B(R)$
- TwoPhase-MergeSort = $3B(R)$ čtení a zápisů
 - Finální zápis ignorujeme

■ Relace je prokládaná

R1 R2 *S1 S2*

R3 R4 *S3 S4*

...

- Čtení je až $T(R)$ bloků!
- TwoPhase-MergeSort
 - $T(R) + 2B(R)$ čtení a zápisů

Operace čtení relace: **index scan**

■ Čtení relace s použitím indexu

- Procházíme index → čteme záznamy

- Čteme bloky indexu ($\ll B(R)$)

- Čteme záznamy relace

- Na libovolném atributu

- Max. náklady:

- (max. $B(R)$ až $T(R)$ čtení) + (až $m^{HT+1} - 1$)

- Where m is an index arity ($LB = m^{HT}$)

- Min. náklady:

- 0 čtení bloků relace + $1..HT$ bloků indexu

■ Výhoda

- Lze omezit pouze na interval záznamů

- „Covering“ index nevyžaduje čtení záznamů

Maximální počet uzlů
 m -árního stromu



Jednoprůchodové algoritmy

■ Implementace:

- Čtení relace → zpracování → výstupní paměť
- Zpracování záznam po záznamu

■ Operace

- Projekce, selekce, rušení duplicit (DISTINCT)
 - Náklady $B(R)$
- Agregáční funkce (GROUP BY)
 - Náklady $B(R)$
- Množinové operace, kartézský součin
 - Náklady $B(R) + B(S)$

Rušení duplicit – distinct

- Postup zpracování
 - Otestuj, zda je již záznam ve výstupu
 - Ne, přidej na výstup
- Testování existence ve výstupu
 - Pamatovat si v paměti již vypsané záznamy
 - Lze použít $M-2$ bloků
 - Testování sekvenčně je pomalé (n^2 porovnání)
 - Použití hašování
- Omezení: $B(R) < M-1$
- Lze realizovat pomocí iterátorů?

Distinct – example

■ Relation company(company_key,company_name)

```
# explain analyze SELECT DISTINCT company_name FROM provider.company;
HashAggregate (cost=438.68..554.67 rows=11600 width=20) (actual time=9.347..12.133 rows=11615 loops=1)
  Group Key: company_name
    -> Seq Scan on company (cost=0.00..407.94 rows=12294 width=20)
      (actual time=0.019..5.007 rows=12295 loops=1)
Planning time: 0.063 ms
Execution time: 12.799 ms
```

```
# explain analyze SELECT DISTINCT company_key FROM provider.company;
Unique (cost=0.29..359.43 rows=12294 width=8) (actual time=0.041..8.857 rows=12295 loops=1)
  -> Index Only Scan using company_pkey on company (cost=0.29..328.69 rows=12294 width=8)
    (actual time=0.039..5.686 rows=12295 loops=1)
    Heap Fetches: 4726
Planning time: 0.063 ms
Execution time: 9.645 ms
```

```
# explain analyze SELECT DISTINCT company_name FROM provider.company ORDER BY company_name;
Unique (cost=1243.05..1304.52 rows=11600 width=20) (actual time=53.468..59.072 rows=11615 loops=1)
  -> Sort (cost=1243.05..1273.79 rows=12294 width=20) (actual time=53.467..55.482 rows=12295 loops=1)
    Sort Key: company_name
    Sort Method: quicksort Memory: 1214kB
    -> Seq Scan on company (cost=0.00..407.94 rows=12294 width=20)
      (actual time=0.018..5.338 rows=12295 loops=1)
```



Agregační funkce (GROUP BY)

■ Postup zpracování

- Vytváření skupin pro group-by atributy
- Ukládání hodnot atributů pro agregační funkce

■ Interní struktura

- Organizace skupin – např. hašování
- Stav agregační funkce
 - MIN, MAX, COUNT, SUM – pouze jedno „číslo/hodnota“
 - AVG – dvě čísla (SUM a COUNT)
- Ukládaná informace je malá: $M-1$ bloků bývá dostatečné

■ Iterátory:

- Vše je vypočteno v *Open*
- Výhoda proudového zpracování mizí



Množinové operace

- Požadavek $\min(B(R), B(S)) < M-1$
 - Menší relace se načte celá
 - Větší se čte postupně
 - Množinové sjednocení a množinový rozdíl
 - Paměť může být větší: $B(R)+B(S) < M-1$
- Předpoklad
 - R je větší relace, tj. S je celá v paměti
- Implementace
 - Obvykle pomocná vyhledávací struktura
 - Např. hašování



Množinové sjednocení

- Pozor: *Ne multimnožinová verze,*
tj. bez ALL v SQL

- Načti S, vybuduj vyhledávací strukturu

- Eliminuj duplicitní řádky
- Unikátní řádky, hned vypisuj

- Při čtení R ověřuj přítomnost záznamu v S

- Je, pak nic.
- Není, pak vypiš a přidej do struktury

- Omezení

- $B(R)+B(S) < M-1$



Množinový průnik

Pozor: *Ne multimnožinová verze,*
tj. bez ALL v SQL

■ Načti S, vybuduj vyhledávací strukturu

Eliminuj duplicitní řádky

■ Při čtení R ověřuj přítomnost záznamu v S

Je, pak vypiš a smaž z interní struktury.

Není, pak nic.

■ Omezení

$\min(B(R), B(S)) < M-1$

Množinový rozdíl

■ R–S

- Načti S, vybuduj vyhledávací strukturu
 - Eliminuj případné duplicity v S
- Při čtení R ověřuj přítomnost záznamu v S
 - Pokud není, dej na výstup
 - také přidej do interní struktury
- $B(S) + B(R) < M-1$ (nejhorší případ)

■ S–R

- Načti S, vybuduj vyhledávací strukturu
 - Eliminuj duplicity
- Při čtení R ověřuj přítomnost záznamu v S
 - Pokud je, smaž záznam z interní struktury
- Nakonec vypiš zbylý obsah S
- $B(S) < M$

Multimnožinové operace

- Multimnožinové sjednocení $R \cup_B S$
 - Snadné cvičení...
- Multimnožinový průnik $R \cap_B S$
 - Načti S , vybuduj vyhledávací strukturu
 - Místo ukládání duplicitních řádků ukládej jejich počet
 - Při čtení R ověřuj přítomnost záznamu v S
 - Záznam byl nalezen, pak dej na výstup
 - A sniž počet záznamů!
 - Pokud je počet již nula, pak zruš z interní struktury.
 - Záznam nebyl nalezen, pak nic
 - $\min(B(R), B(S)) < M-1$



Multimnožinové operace

■ Multimnožinový rozdíl $S -_B R$

- Používá stejný trik
- Záznam z R byl nalezen, sniž počet záznamů
- Nakonec vypiš pouze ty záznamy z S
 - které mají kladný počet.

■ Multimnožinový rozdíl $R -_B S$

- Analogicky...
- Záznam z R nebyl v S nalezen → výstup
- Záznam z R byl v S nalezen
 - → pokud je počet nula, dej na výstup.
 - → jinak sniž počet a nic



Operace spojení - jednopřechodově

■ Kartézský součin

- Snadné cvičení...

■ Přirozené spojení (NATURAL JOIN v SQL)

- Předpoklad $R(X, Y)$, $S(Y, Z)$

- X – atributy unikátní v R , Z – atributy unikátní v S
- Y – atributy společné v R a S

- Načti S , vybuduj vyhledávací strukturu pro Y

- Pro záznam z R , najdi v S všechny odpovídající

- Na výstup dej jejich kombinace (eliminuj opakování Y)

■ Vnější spojení ?



Jednoprůchodové algoritmy

■ Shrnutí

- Unární operace: $op(R)$

- $B(R) \leq M-1$, 1 blok pro výstup; někdy i 1 pro vstup

- Binární operace: $R \ op \ S$

- $B(S) \leq M-2$, 1 blok pro R , 1 blok pro výstup

- U některých $B(R)+B(S) \leq M-2$

- Cena = $B(R) + B(S)$

■ Založeno na volné paměti M

- Je známo \rightarrow ok

- Není známo \rightarrow odhadnout

- Chyba \rightarrow swapování, výměna jednoprůchodového za dvouprůchodový algoritmus

Algoritmy pro spojení

- Relace se nevejdou do paměti
 - Tzv. „jeden a půl“ průchodové algoritmy
- Základ – vnořené cykly (*nested-loop join*)
 - **for** each s in S **do**
 - **for** each r in R **do**
 - **if** r a s se shodují **then** vypiš spojení r a s .
- Příklad
 - $T(R) = 10\ 000$ $T(S) = 5\ 000$ $M=2$
 - Náklady = $5\ 000 \cdot (1 + 10\ 000) = 50\ 005\ 000$ čtení

Čtení záznamu z S

Čtení celé R



Algoritmy pro spojení

- Relace přístupovány po blocích
- Blokované vnořené cykly
 - *block-based nested-loop join*
 - R – vnitřní relace, S – vnější relace
- Příklad:
 - $B(R) = 1000$ $B(S) = 500$ $M=3$
 - Náklady = $500 \cdot (1+1000) = 500\,500$ čtení

Algoritmy pro spojení

- Využití vyrovnávací paměti (M bloků)
 - Cached Block-based Nested-loop Join
 - Načti M-2 bloků relace S naráz
 - Načítej relaci R po 1 bloku
 - Spojuj záznamy
 - Náklady: $B(S)/(M-2) \cdot (M-2 + B(R))$ čtení
- Příklad $R \bowtie S$:
 - $M=102$
 - Náklady: $5 \cdot (100 + 1000) = 5\,500$ čtení
 - Změna pořadí relací
 - Náklady: $10 \cdot (100 + 500) = 6\,000$ čtení



Algoritmy pro spojení – hodnocení

- Vnořené cykly
 - Vždy blokovávaná varianta
 - Do paměti načítat dávky menší relace (pro $M > 3$)
- Způsob uložení relace
 - Důležité pro výslednou cenu
 - Prokládané → každý záznam jedno čtení
 - Neprokládané → každý záznam $B(R)/T(R)$ čtení
- Využitelné pro libovolnou podmínku spojení
 - tzv. theta-joins



Dvouprůchodové algoritmy

■ Princip:

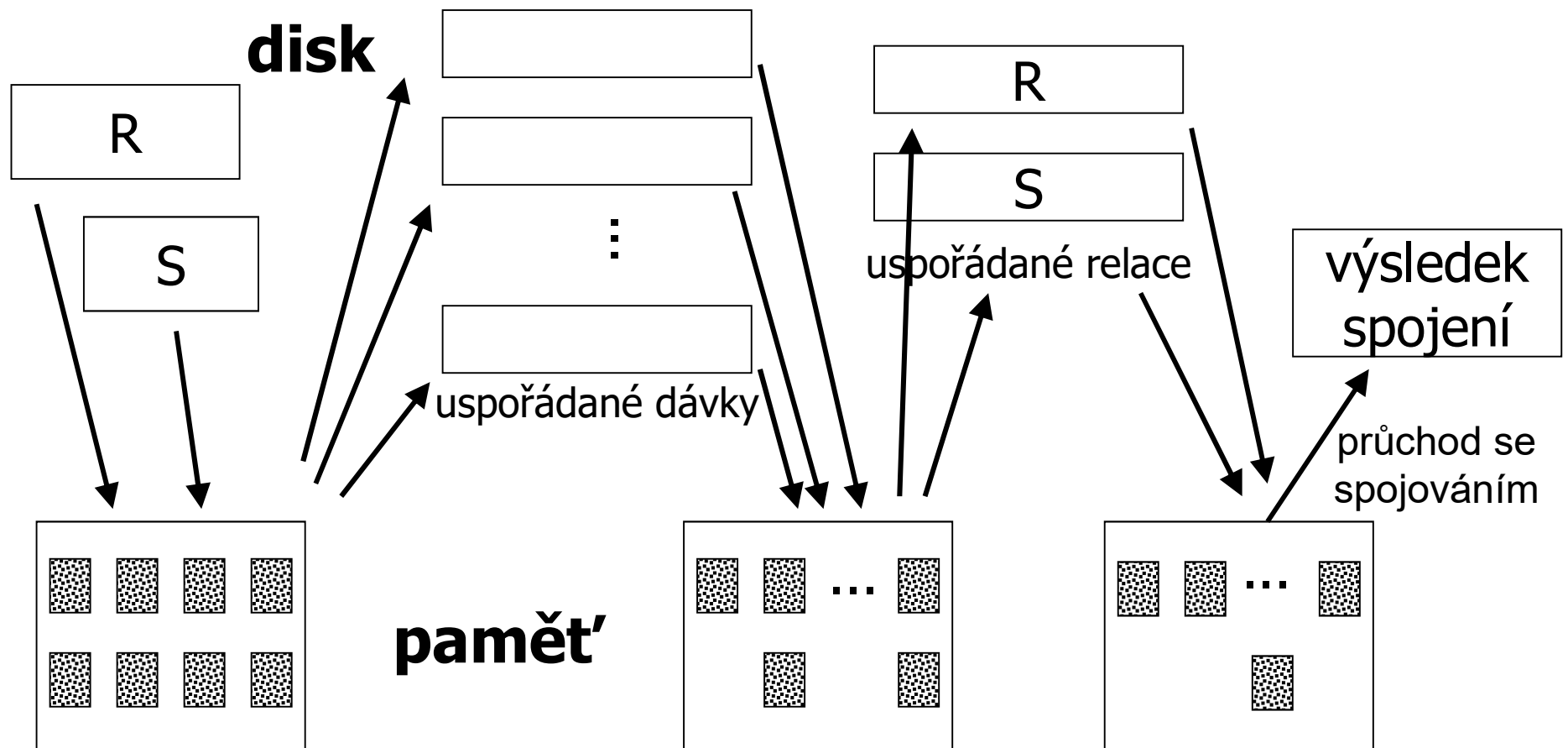
- Předzpracování vstupu → uložení
 - Třídění (vícecestný MergeSort)
 - Hašování
- Zpracování

■ Operace:

- Spojení relací
- Rušení duplicit (DISTINCT)
- Agregáční funkce (GROUP BY)
- Množinové operace

Algoritmy pro spojení – MergeJoin

■ $R \bowtie S$ $R(X,Y), S(Y,Z)$





Algoritmy pro spojení – MergeJoin

- $R \bowtie S$ $R(X,Y), S(Y,Z)$
- Algoritmus:
 - Setříd' R a S
 - $i = 1; j = 1;$
 - **while** $(i \leq T(R)) \wedge (j \leq T(S))$ **do**
 - **if** $R[i].Y = S[j].Y$ **then** doJoin()
 - **else if** $R[i].Y > S[j].Y$ **then** $j = j+1$
 - **else if** $R[i].Y < S[j].Y$ **then** $i = i+1$

Algoritmy pro spojení – MergeJoin

■ Funkce doJoin():

□ Proved' nested-loop join pro řádky se stejným Y

□ **while** $(R[i].Y = S[j].Y) \wedge (i \leq T(R))$ **do**

■ $j_2 = j$

■ **while** $(R[i].Y = S[j_2].Y) \wedge (j_2 \leq T(S))$ **do**

□ Vypiš spojení $R[i]$ a $S[j_2]$

□ $j_2 = j_2 + 1$

■ $i = i + 1$

□ $j = j_2$



Algoritmy pro spojení – MergeJoin

i	R[i].Y	S[j].Y	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7



Algoritmy pro spojení – MergeJoin

■ Cena

- MergeSort R a S $\rightarrow 4 \cdot (B(R) + B(S))$
- MergeJoin $\rightarrow B(R) + B(S)$

■ Příklad (M=102)

□ MergeJoin

- Uspořádání: $4 \cdot (1000 + 500) = 6000$ čtení/zápisů
 - Spojení: $1000 + 500 = 1500$ čtení
 - Celkem: 7500 čtení/zápisů
- ### □ Původní cached block-based nested-loop join
- 5500 čtení

Algoritmy pro spojení – MergeJoin

■ Jiný příklad

□ $B(R) = 10\ 000$

$B(S) = 5\ 000$

10x větší relace!!!

□ $M = 102$ bloků

□ Cached Block-based Nested-loop Join

■ $(5\ 000/100) \cdot (100 + 10\ 000) = 505\ 000$ čtení

□ MergeJoin

■ $5 \cdot (10\ 000 + 5\ 000) = 75\ 000$ čtení a zápisů



Algoritmy pro spojení – MergeJoin

■ MergeJoin

- Předzpracování je drahé

- Pokud jsou relace uspořádány podle Y, lze vynechat.

■ Náklady – analýza V/V operací

- MergeJoin

- lineární složitost

- Cached Block-based Nested-loop Join

- kvadratická složitost

- → od jisté velikosti relací je MergeJoin lepší

Algoritmy pro spojení – MergeJoin

■ Paměťové nároky

□ Omezení na $\max(B(R), B(S)) < M^2$

■ Optimální paměť

□ Používáme MergeSort na relaci R

■ Počet dávek = $B(R)/M$, Délka dávky = M

■ Omezení: počet dávek $\leq M - 1$

■ $B(R)/M < M \rightarrow B(R) < M^2 \rightarrow M > \sqrt{B(R)}$

■ Příklad

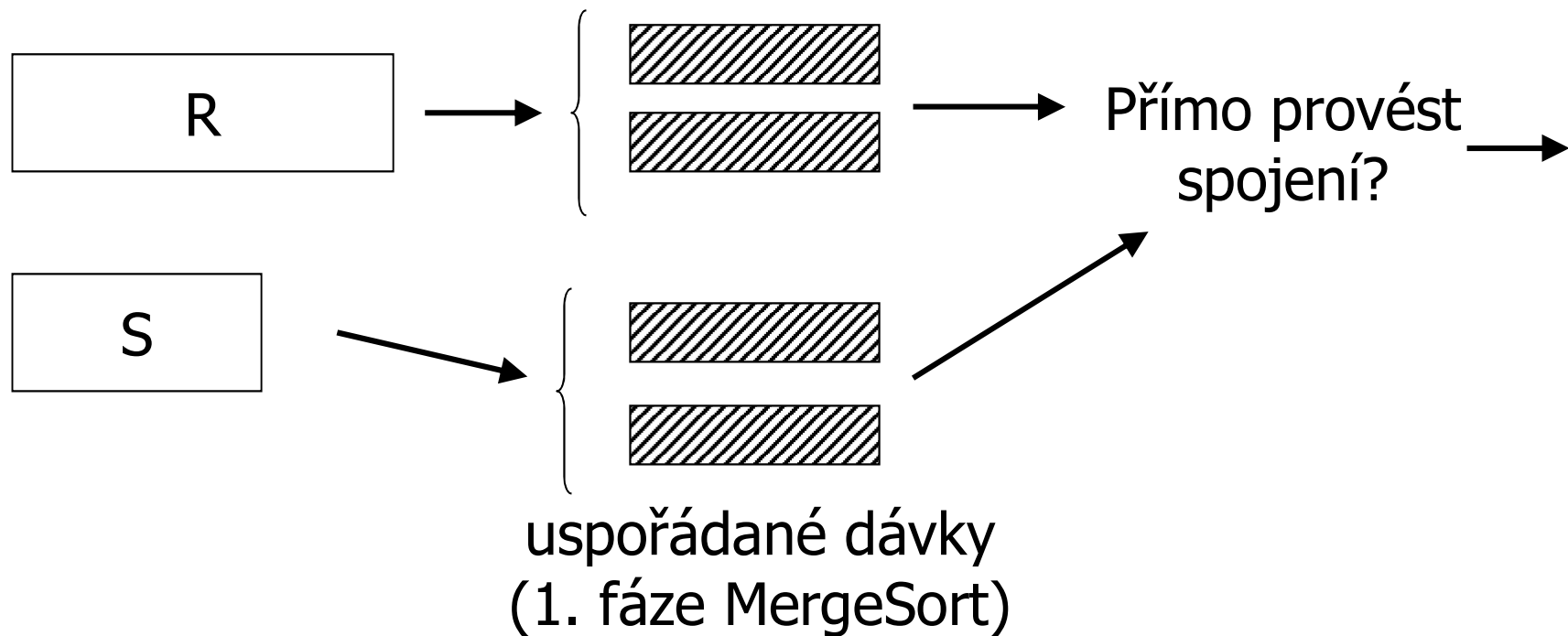
□ $B(R) = 1000 \rightarrow M > 31,62$

□ $B(S) = 500 \rightarrow M > 22,36$

Algoritmy pro spojení – MergeJoin

- Vylepšení:

- není potřeba mít relace zcela uspořádané





Algoritmy pro spojení – SortJoin

■ Vylepšení

- Vytvoř setříděné dávky R a S
- Načti první blok z každé dávky (R i S)
- Zjistí minimální hodnotu v Y
 - Najdi odpovídající záznamy z ostatních dávek
 - Proveď spojení

■ Pokud je hodně řádků se stejným Y

- Aplikuj block-nested-loop join ve zbytku paměti



Algoritmy pro spojení – SortJoin

■ Náklady

- Uspořádání dávek: $2 \cdot (B(R) + B(S))$
- Provedení spojení: $B(R) + B(S)$

■ Omezení

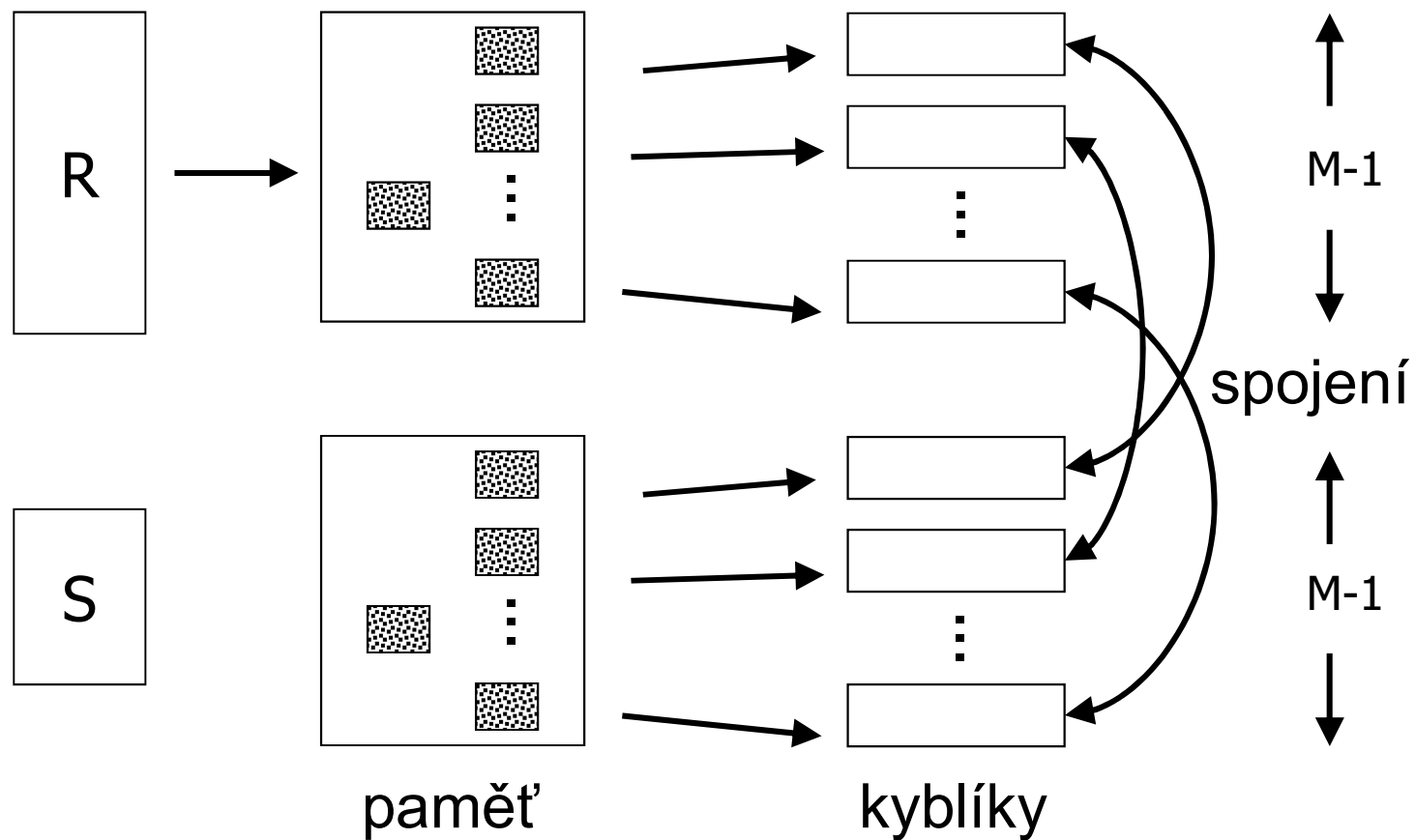
- Délka dávek M , počet dávek M
- $\rightarrow B(R) + B(S) < M^2$

■ Příklad ($M=102$)

- Uspořádání dávek: $2 \cdot (1000 + 500)$
- Spojení: $1000 + 500$
- Celkem: 4 500 čtení/zápisů
 - \rightarrow lepší než cached block-based nested-loop join

Algoritmy pro spojení – HashJoin

■ $R \bowtie S$ $R(X,Y), S(Y,Z)$





Algoritmy pro spojení – HashJoin

- $R \bowtie S$ $R(X,Y), S(Y,Z)$
 - Pro atributy Y vytvoř hašovací funkci
 - Vytvoř hašovaný index pro R i S
 - Počet kyblíků je $M-1$
 - Pro každé $i \in [0, M-2]$
 - Načti kyblík i pro S
 - Načítej kyblík i pro R a proved' vyhledání odpovídajících si záznamů a jejich spojení

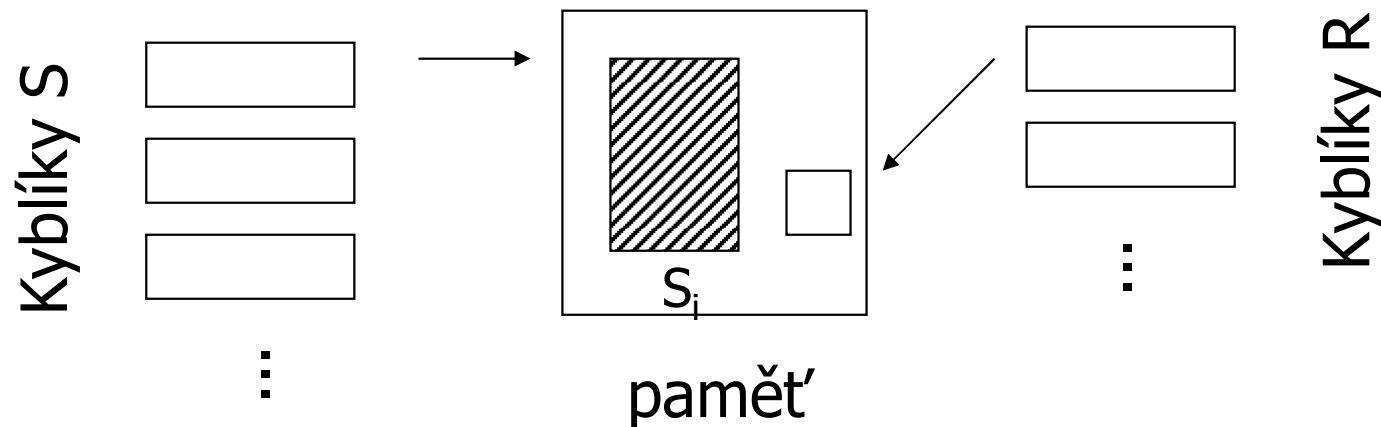
Algoritmy pro spojení – HashJoin

■ Spojování kyblíků

□ Kyblík S načti celý ($\leq M-2$)

■ Pro zrychlení si vytvoř paměťové hašování

□ Kyblík R čti po blocích





Algoritmy pro spojení – HashJoin

■ Náklady:

- Vytvoření hašovaného indexu: $2 \cdot (B(R) + B(S))$
- Provedení spojení: $B(R) + B(S)$

■ Omezení:

- Velikost každého kyblíku $S \leq M - 2$
 - Odhad: $\min(B(R), B(S)) < (M - 1) \cdot (M - 2)$

■ Příklad:

- Hašování: $2 \cdot (1000 + 500)$
- Spojení: $1000 + 500$
- Celkem: 4 500 čtení/zápisů

Algoritmy pro spojení – HashJoin

■ Minimální paměťové nároky

□ Hašování S , optimální naplnění kyblíků

■ Celkem máme M paměťových bloků

■ Velikost kyblíku = $B(S) / (M-1)$

□ Musí být menší než M (kvůli spojení)

□ $\rightarrow \lceil B(S)/(M-1) \rceil \leq M-2$

■ $\approx M-1 > \lceil \sqrt{B(S)} \rceil$



Algoritmy pro spojení – HashJoin

■ Optimalizace

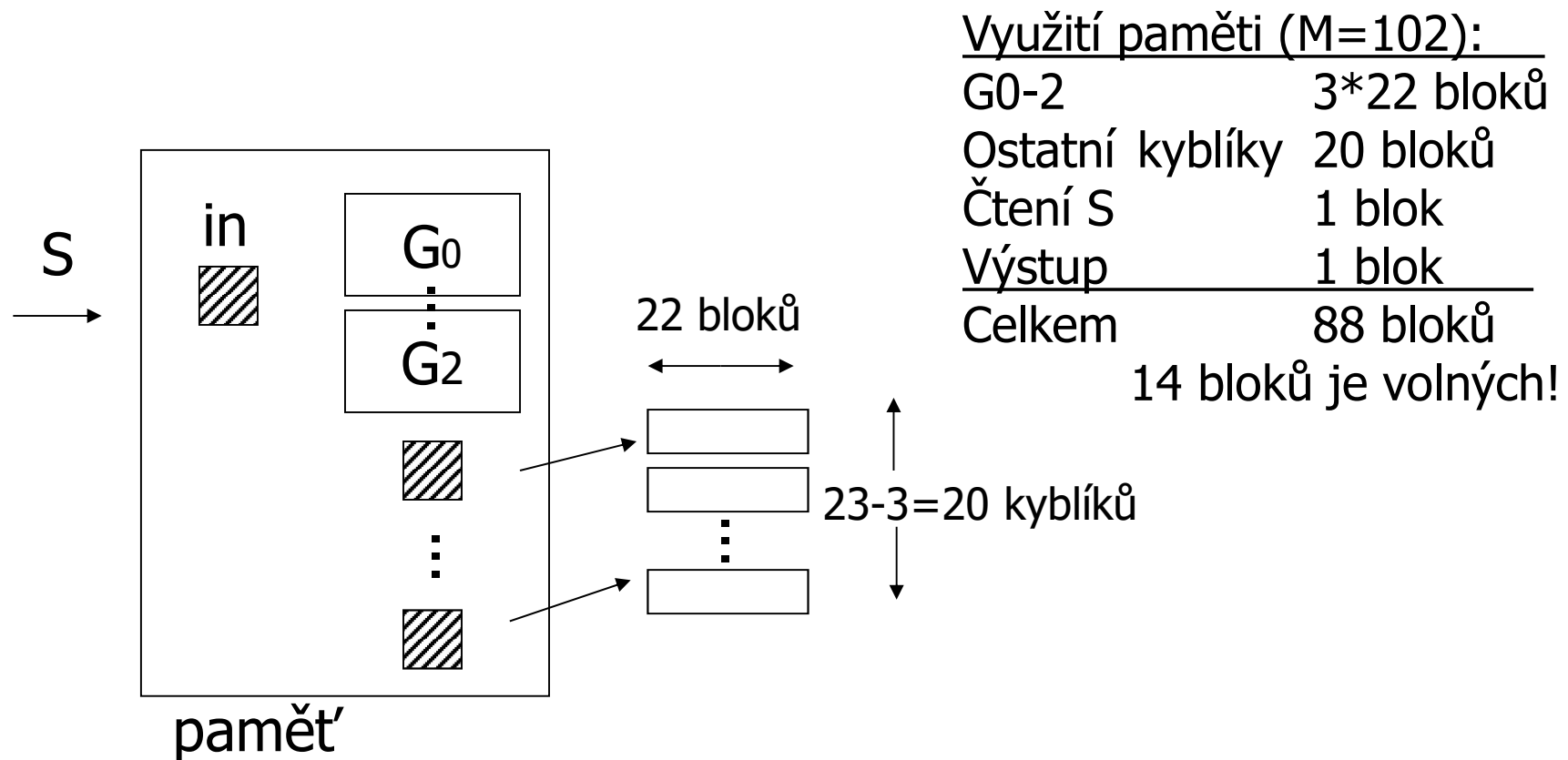
- ponech některé kyblíky v paměti
- Hybrid HashJoin

■ Optimum počtu kyblíků pro R

- $B(S)=500$
- $\sqrt{B(S)} \approx 23$
- Tj. 22 bloků má každý kyblík
- $M=102$
 - → ponechej 3 kyblíky v paměti (66 bloků)
 - → zbývá 36 bloků paměti

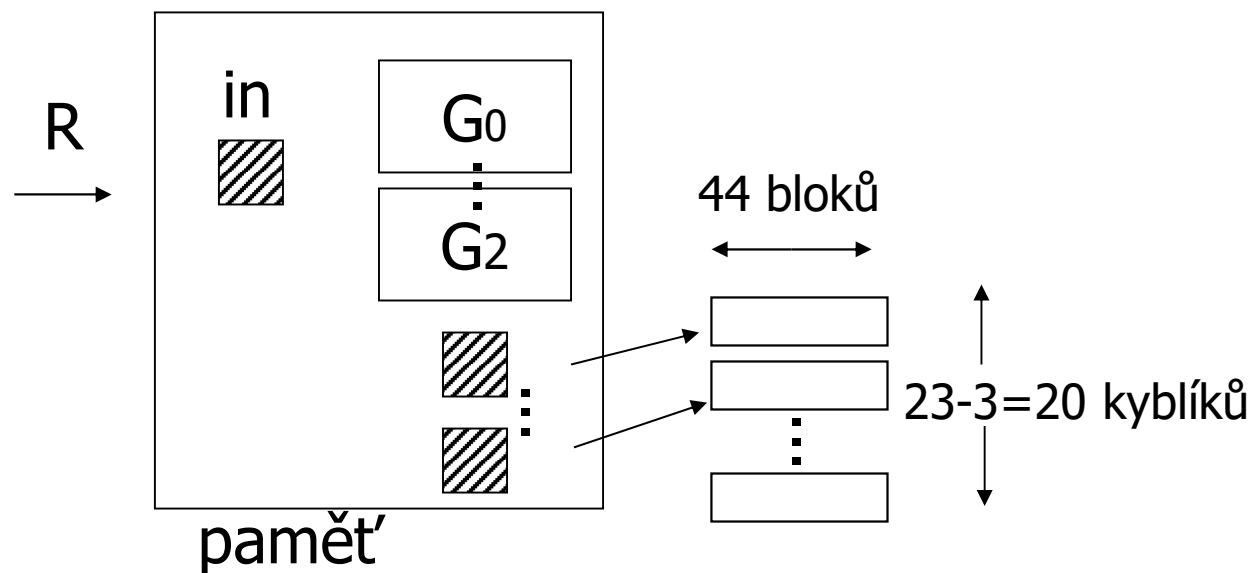
Alg. pro spojení – Hybrid HashJoin

■ Paměť pro vytvoření hašovaného indexu S



Alg. pro spojení – Hybrid HashJoin

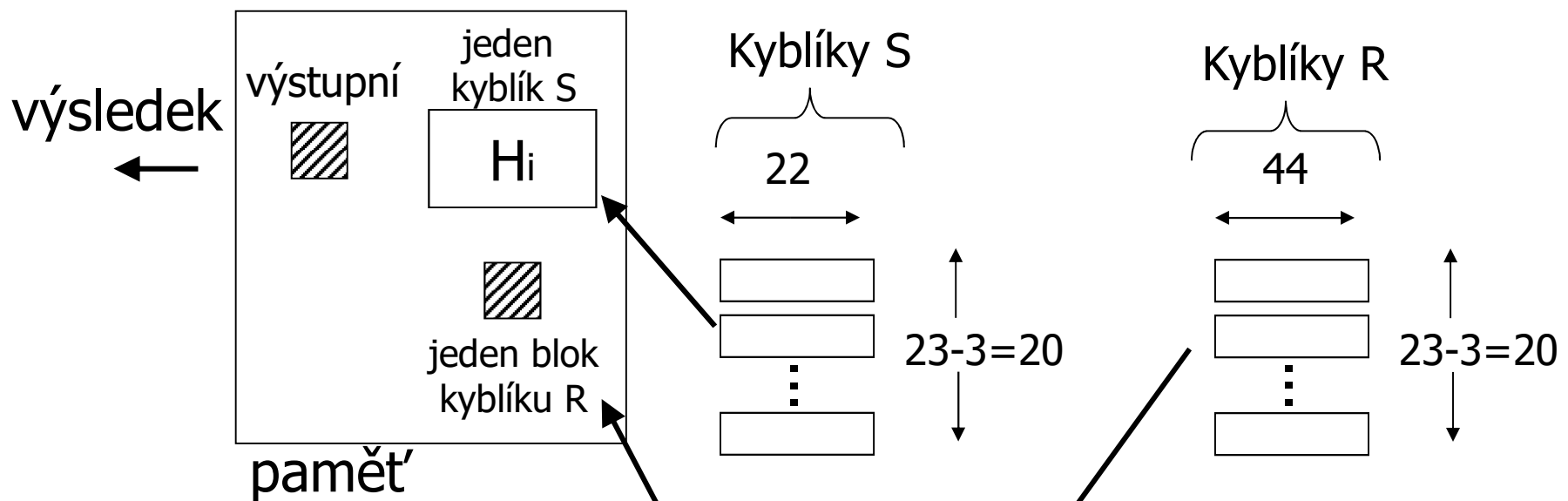
- Paměť pro vytvoření hašovaného indexu R
 - $1000/23 = 44$ bloků na kyblík
 - Záznamy hašované do kyblíků 0-2
 - Vyřešit hned (S_{0-2} jsou v paměti) → výstup




Alg. pro spojení – Hybrid HashJoin

■ Spojení kyblíků

- Pouze pro kyblíky s id 3-22
- Načti jeden kyblík celý do paměti, druhý procházej po blocích





Alg. pro spojení – Hybrid HashJoin

■ Náklady:

- Hašování S: $500 + 20 \cdot 22 = 940$ čtení/zápisů
- Hašování R: $1000 + 20 \cdot 44 = 1880$ čtení/zápisů
 - Pouze 20 kyblíků k zápisu!
- Spojení: $20 \cdot 44 + 20 \cdot 22 = 1320$ čtení
 - 3 kyblíky jsou již zpracovány
- Celkem: 4140 čtení/zápisů



Algoritmy pro spojení

■ Hybrid HashJoin

- Kolik kyblíků ponechat v paměti?

- Empiricky: 1 kyblík

■ Hašování ukazatelů

- Hašování ne záznamů, ale ukazatelů

- Do kyblíků ukládej dvojice [hodnota, ukazatel]

- Spojování

- Při shodě hodnot si musíme záznam načíst



Alg. pro spojení – Hašování ukazatelů

■ Příklad

- Do bloku se vejde 100 dvojic hodnota-klíč
- Odhadovaný výsledek je 100 záznamů
- Náklady:
 - Hašování S do paměti
 - 5000 záznamů \rightarrow $5000/100$ bloků = 50 bloků
 - Spojení – čti R a spojuj
 - Při shodě načti záznam S \rightarrow 100 čtení
 - Celkem: $500 + 1000 + 100 = 1600$ čtení



Algoritmy pro spojení – IndexJoin

- $R \bowtie S$ $R(X,Y), S(Y,Z)$

- Předpoklad:

- R má index nad atributy Y

- Postup:

- Pro každý záznam $s \in S$

- Prohledej index na shodu \rightarrow záznamy A

- Pro každý záznam $r \in A$

- Vypiš kombinaci r a s



Algoritmy pro spojení – IndexJoin

■ Příklad

□ Předpoklady

- Index na Y pro relaci R: $HT=2$, $LB=200$

■ Situace 1

□ Index se vejde do paměti

□ Náklady:

- Průchod S: 500 čtení ($B(S)=500$, $T(S)=5000$)
- Prohledání indexu: zdarma
 - Pokud shoda, načti záznam R → 1 čtení

Algoritmy pro spojení – IndexJoin

■ Náklady

□ Závisí na počtu shod v indexu

□ Případy:

■ A) Y je v R primární klíč, v S je cizí klíč → 1 záznam

Výsledek: $500 + 5000 \cdot 1 \cdot 1 = 5500$ čtení

■ B) $V(R, Y) = 5000$ $T(R) = 10\ 000$

rovnoměrné rozložení → 2 záznamy

Výsledek: $500 + 5000 \cdot 2 \cdot 1 = 10500$ čtení

■ C) $DOM(R, Y) = 1\ 000\ 000$ $T(R) = 10\ 000$

→ $10k/1m = 1/100$ záznamu

Výsledek: $500 + 5000 \cdot (1/100) \cdot 1 = 550$ čtení



Algoritmy pro spojení – IndexJoin

■ Situace 2

- Index se nevejde do paměti
- Index na Y pro R má 201 bloků
 - V paměti $M=102$ udržuj kořen a 99 listů
- Náklady pro vyhledání
 - $0 \cdot (99/200) + 1 \cdot (101/200) = 0.505$ čtení

Algoritmy pro spojení – IndexJoin

■ Situace 2

□ Náklady

- $B(S) + T(S) \cdot (\text{prohledání indexu} + \text{čtení záznamů})$

□ Případy:

- A) \rightarrow 1 záznam

Výsledek: $500 + 5000 \cdot (0.5 + 1) = 8000$ čtení

- B) \rightarrow 2 záznamy

Výsledek: $500 + 5000 \cdot (0.5 + 2) = 13000$ čtení

- C) \rightarrow 1/100 záznamu

Výsledek: $500 + 5000 \cdot (0.5 + 1/100) = 3050$ čtení

Algoritmy pro spojení – shrnutí

$R \bowtie S$
 $B(R) = 1000$
 $B(S) = 500$

Algorithm	Costs
Cached Block-based Nested-loop Join	5500
Merge Join (w/o sorting)	1500
Merge Join (with sorting)	7500
Sort Join	4500
Index Join (R.Y index)	8000 → 550
Hash Join	4500
Hybrid	4140
Pointers	1600

Join Algorithms – Summary

$R \bowtie S$

Assume $B(S) < B(R)$, Y are common attributes

Algorithm	Costs	Limits
Block-based Nested-loop	$B(S) \cdot (1+B(R))$	$M=3$
Cached version	$B(S)/(M-2) \cdot (M-2 + B(R))$	$M \geq 3$
Merge Join (w/o sorting)	$B(R) + B(S)$	$M=3$
Merge Join (with sorting)	$5 \cdot (B(R) + B(S))$	$M = \sqrt{B(R)}$
Sort Join	$3 \cdot (B(R) + B(S))$	$M = \sqrt{B(R)} + \sqrt{B(S)} + 1$
Index Join (R.Y index) (max costs)	$B(S) + T(S) \cdot (HT + \theta)$ e.g. $\theta = T(R)/V(R,Y)$	min. $M=4$
Hash Join	$3 \cdot (B(R) + B(S))$	$M = 2 + \sqrt{B(S)}$ max. $M-1$ buckets
Hybrid	$3(B(R) + B(S)) - \frac{2(B(R) + B(S))}{\lceil \sqrt{B(R)} \rceil}$	$M = \frac{B(R)}{\lceil \sqrt{B(R)} \rceil} + (\lceil \sqrt{B(R)} \rceil) + 1$
Pointers	$B(S)+B(R)+T(R) \cdot \theta$ e.g. $\theta = T(S)/V(S,Y)$	$M=B(\text{hash index on } S)+3$



Algoritmy pro spojení – doporučení

- **Cached Block-based Nested-loop Join**
 - Vhodné pro malé relace (vzhledem k paměti)
- **HashJoin**
 - Pro spojení na rovnost (*equi-join*)
 - Relace nejsou uspořádané a nejsou indexy
- **SortJoin**
 - Vhodný pro spojení s nerovností (*non-equi-join*)
 - Např. $R.Y > S.Y$
- **MergeJoin**
 - Pokud jsou relace již uspořádané
- **IndexJoin**
 - Pokud jsou indexy, může být vhodná volba
 - Závisí na velikosti odpovědi



Dvouprůchodové algoritmy

- Pomocí třídění
 - Rušení duplicit
 - Agregáčn  funkce (GROUP BY)
 - MnoŹinov  operace

Rušení duplicit

■ Postup zpracování

- Proved' první fázi MergeSort
 - → uspořádané dávky na disku
- Z každé dávky načítej postupně bloky
 - Vezmi nejmenší záznam a dej na výstup
 - Přeskoč všechny duplicitní záznamy

■ Vlastnosti

- Náklady: $3B(R)$
- Omezení: $B(R) \leq M^2$
 - Optimální $M \geq \sqrt{B(R)}$



Agregační funkce

- Postup (podobný předchozímu)
 - Uspořádej dávky R (podle group-by atributů)
 - Z každé dávky načítej postupně bloky
 - Vezmi nejmenší záznam \rightarrow nová skupina
 - Počítej agregační funkce pro všechny stejné záznamy
 - Žádný další není \rightarrow vypiš výsledky na výstup
- Vlastnosti
 - Náklady: $3B(R)$
 - Omezení: $B(R) \leq M^2$
 - Optimální $M \geq \sqrt{B(R)}$

Množinové sjednocení

- Pro multimnožiny není třeba dvou průchodů
- Množinové sjednocení
 - Proveď první fázi MergeSort pro R a S
 - → uspořádané dávky na disku
 - Z každé dávky (R i S) načítej postupně bloky
 - Vezmi nejmenší záznam a dej na výstup
 - Přeskoč všechny duplicitní záznamy (z R i S)
- Vlastnosti
 - Náklady: $3(B(R) + B(S))$
 - Omezení: $\sqrt{B(R) + B(S)} \leq M$
 - Potřebují jeden blok pro všechny dávky R i S

Množinový průnik a rozdíl

- $R \cap S$, $R - S$, $R \cap_B S$, $R -_B S$

- Postup

- Proveď první fázi MergeSort pro R a S
- Z každé dávky (R i S) načítej postupně bloky
 - Vezmi nejmenší záznam t
 - Spočítej jeho všechny výskyty v R a S (odděleně)
 - $\#_R$, $\#_S$
 - Vypiš na výstup (respektuj danou operaci)

Množinový průnik a rozdíl

■ Ad výpis

- $R \cap S$: vypiš t ,
 - pokud $\#_R > 0 \wedge \#_S > 0$
- $R \cap_B S$: vypiš t $\min(\#_R, \#_S)$ -krát
- $R - S$: vypiš t ,
 - pokud $\#_R > 0 \wedge \#_S = 0$
- $R -_B S$: vypiš t $\max(\#_R - \#_S, 0)$ -krát

■ Vlastnosti

- Náklady: $3(B(R) + B(S))$
- Omezení: $\sqrt{B(R) + B(S)} \leq M$
 - Potřebují jeden blok pro všechny dávky R i S



Dvouprůchodové algoritmy

- Pomocí hašování

- Rušení duplicit
- Agregáčn  funkce (GROUP BY)
- MnoŹinov  operace



Rušení duplicit

■ Postup zpracování

- Proved' hašování R do $M-1$ kyblíků

- → kyblíky ukládej na disk

- Pro každý kyblík

- Načti do paměti a zruš duplicity, dále zbytek na výstup

- Velikost kyblíku je max. $M-1$

■ Vlastnosti

- Náklady: $3B(R)$

- Omezení: $B(R) \leq (M-1)^2$

Agregační funkce

- Postup (podobný předchozímu)
 - Proved' hašování R do $M-1$ kyblíků
 - podle group-by atributů
 - Pro každý kyblík
 - Načítej po blocích postupně a
 - Vytvářej skupiny a počítej agregační funkce
 - Max. velikost kyblíku neomezená, ale skupiny se musí vejít do max. $M-1$.
 - Vypiš výsledky na výstup
- Vlastnosti
 - Náklady: $3B(R)$
 - Omezení: $B(R) \leq (M-1)^2$

Ize téměř zrušit

Množinové sjednocení, průnik, rozdíl

■ Postup

- Proveď hašování pro R a S (stejnou haš. funkcí!)
 - Vždy $M-1$ kyblíků
- Zpracuj vždy dvojici kyblíků R_i a S_i
 - Jeden z kyblíků načti do paměti
 - velikost kyblíku max. $M-2$
 - Druhý zpracuj postupně

■ Vlastnosti

- Náklady: $3(B(R) + B(S))$
- Omezení: $\min(B(R), B(S)) \leq (M-2)^2$



Summary

- Operations

- distinct, group by, set operations, joins

- Algorithm type

- one-pass, one-and-a-half pass, two-pass

- Implementation

- Sorting
- Hashing
- Exploiting indexes

- Costs

- blocks to read/write
- memory footprint