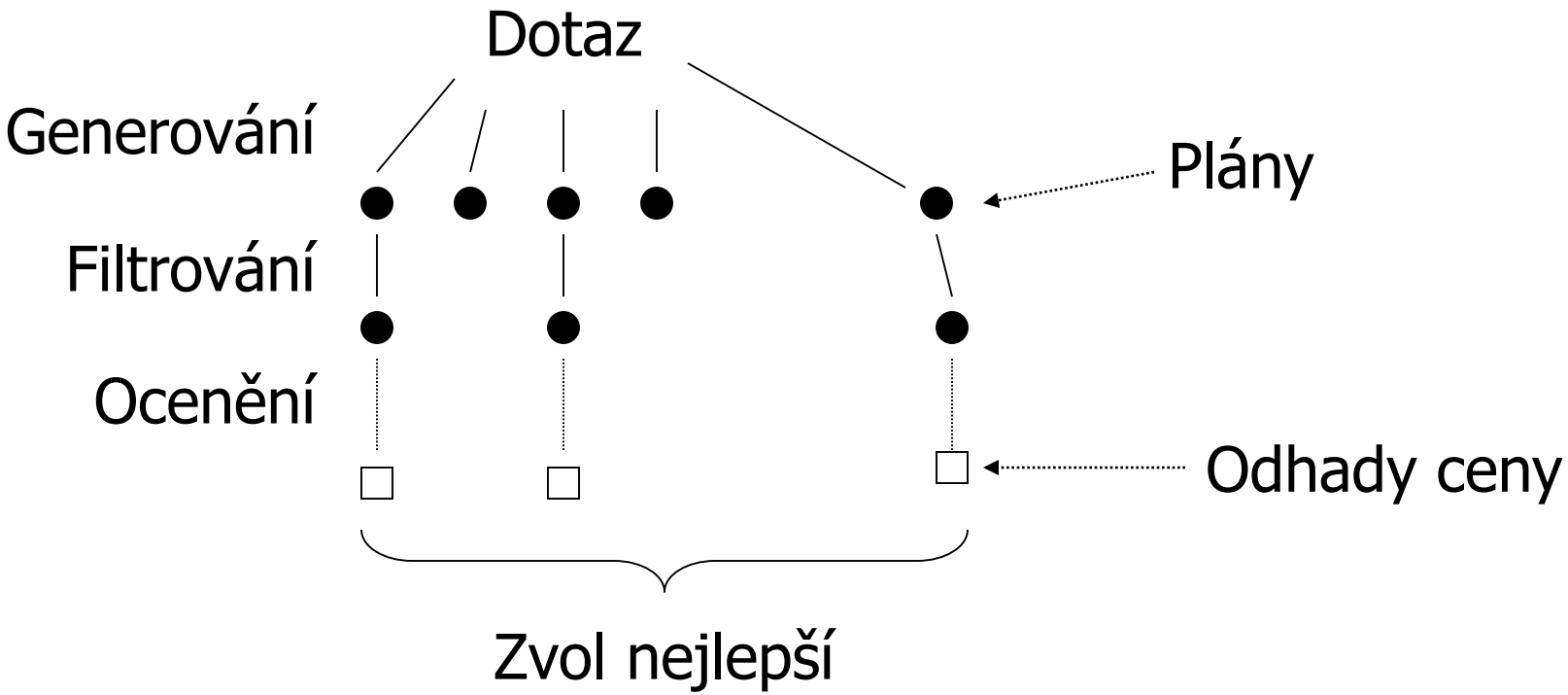
PA152: Efektivní využívání DB

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 plánu

- 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)
  - work_mem (4MB)
    - Memory available to an operation
  - effective_cache_size (4GB)

Odhad ceny plánu

- 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

Operace čtení relace: **table scan**

- Relace není prokládaná

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

  - 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 (<< B(R))
    - Čteme záznamy relace
  - Na libovolném atributu
  - Max. náklady:
    - (max. B(R) až T(R) čtení) + (max. $\frac{m^{HT}-1}{m-1}$)
      - Where *m* is an index arity ($\mathrm{LB} = m^{HT}$)
  - 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

Implementace operace

- Použití konceptu **iterátor**
  - *Open* – inicializace operace
    - příprava před vracením 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)

Jednoprůchodové algoritmy

- Implementace:
  - Čtení relace → zpracování → výstupní paměť
  - Zpracování záznam po záznamu
- Operace
  - Projekce, selekce, rušení duplicit (DISTINCT)
    - Náklady B(R)
  - Agregační 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-1* bloků
  - Testování sekvenčně je pomalé (n^2 porovnání)
  - Použití hašování
- Omezení: $B(R) < M$
- 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-2
  - Menší relace se načte celá
  - Větší se čte postupně
  - Množinové sjednocení a množinový rozdíl
    - Paměť může být větší: B(R)+B(S) < M-2
- 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-2

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-2

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

Zde pozor, omezení je B(R)+B(S) < M-2

- 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

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

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 $\rightarrow$ výstup
  - Záznam z R byl v S nalezen
    - $\rightarrow$ pokud je počet nula, dej na výstup.
    - $\rightarrow$ jinak sniž počet a nic

Operace spojení

- 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) ≤ M-1, 1 blok pro výstup
  - Binární operace: R *op* S
    - B(S) ≤ M-2, 1 blok pro R, 1 blok pro výstup
      - U některých B(R)+B(S) ≤ M-2
  - Cena = B(R) + B(S)
- Založeno na volné paměti M
  - Je známo → ok
  - Není známo → odhadnout
    - Chyba → 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·(1+10 000) = 50 005 000 čtení

Čtení záznamu z S

Čtení celé R

Algoritmy pro spojení

- Relace přistupovány 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·(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

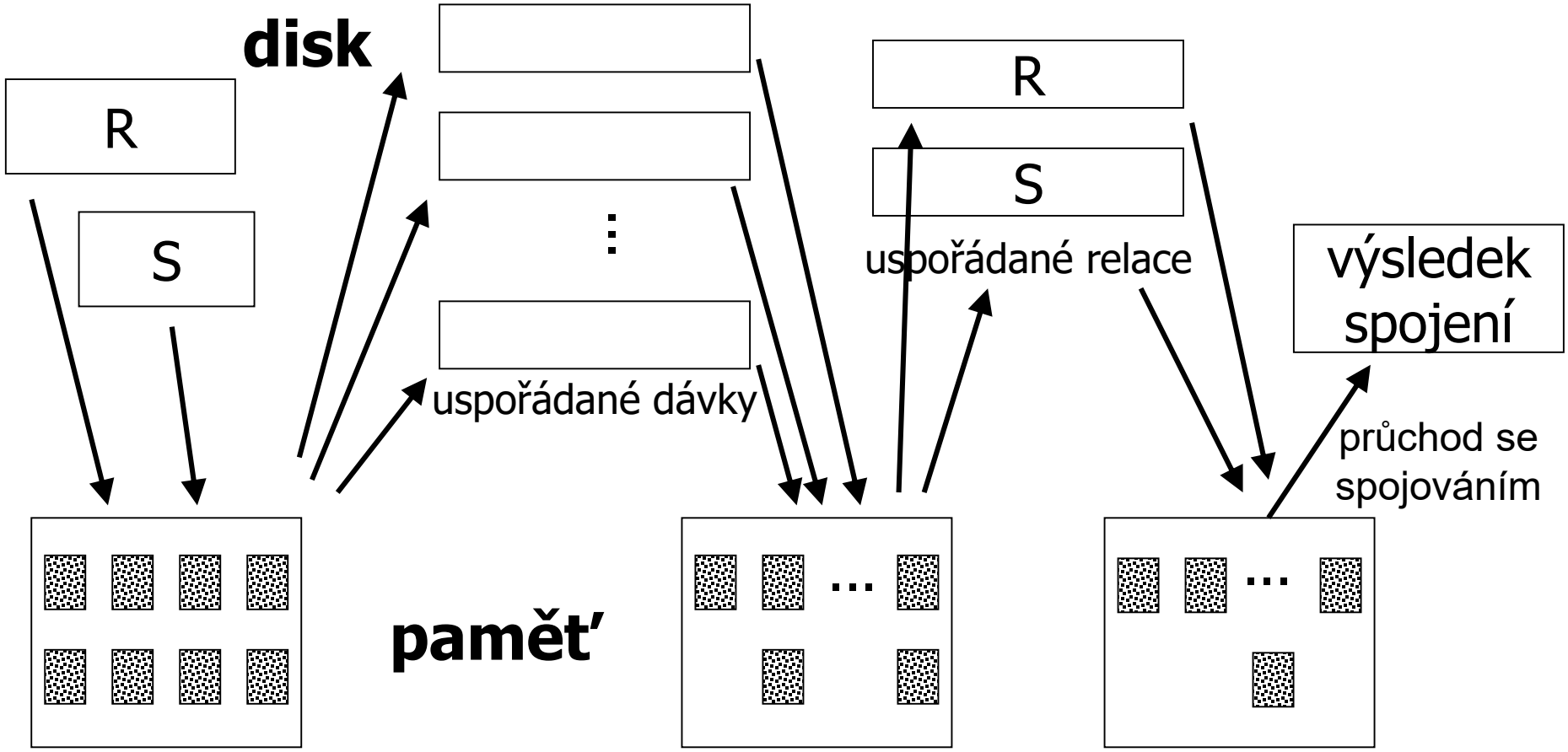
- Vnořené cykly
  - Vždy blokovaná varianta
  - Do paměti načítat dávky menší relace (pro M>3)
- Způsob uložení relace
  - Důležité pro výslednou cenu
    - Prokládané → každý záznam jedno čtení
    - Neprokládané → každý záznam B(R)/T(R) čtení
- Využitelné pro libovolnou podmínku spojení
  - tzv. theta-joins

Dvouprůchodové algoritmy

- Princip:
  - Předzpracování vstupu → uložení
    - Třídění (vícecestný MergeSort)
    - Hašování
  - Zpracování
- Operace:
  - Spojení relací
  - Rušení duplicit (DISTINCT)
  - Agregační funkce (GROUP BY)
  - Množinové operace

Algoritmy pro spojení – MergeJoin

- R⋈S R(X,Y), S(Y,Z)



Algoritmy pro spojení – MergeJoin

- $R \bowtie S$ R(X,Y), S(Y,Z)
- Algoritmus:
  - Setřiď R a S
  - i = 1; j = 1;
  - **while** (i ≤ T(R)) ∧ (j ≤ 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():
  - Proveď nested-loop join pro řádky se stejným Y
  - **while** (R[i].Y = S[j].Y) $\wedge$ (i ≤ T(R)) **do**
    - j2 = j
    - **while** (R[i].Y = S[j2].Y) $\wedge$ (j2 ≤ T(S)) **do**
      - Vypiš spojení R[i] a S[j2]
      - j2 = j2 + 1
    - i = i + 1
  - j = j2

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
  - M = 102 bloků
  - Cached Block-based Nested-loop Join
    - (5 000/100) · (100 + 10 000) = 505 000 čtení
  - MergeJoin
    - 5·(10 000 + 5 000) = 75 000 čtení a zápisů

10x větší relace!!!

Algoritmy pro spojení – MergeJoin

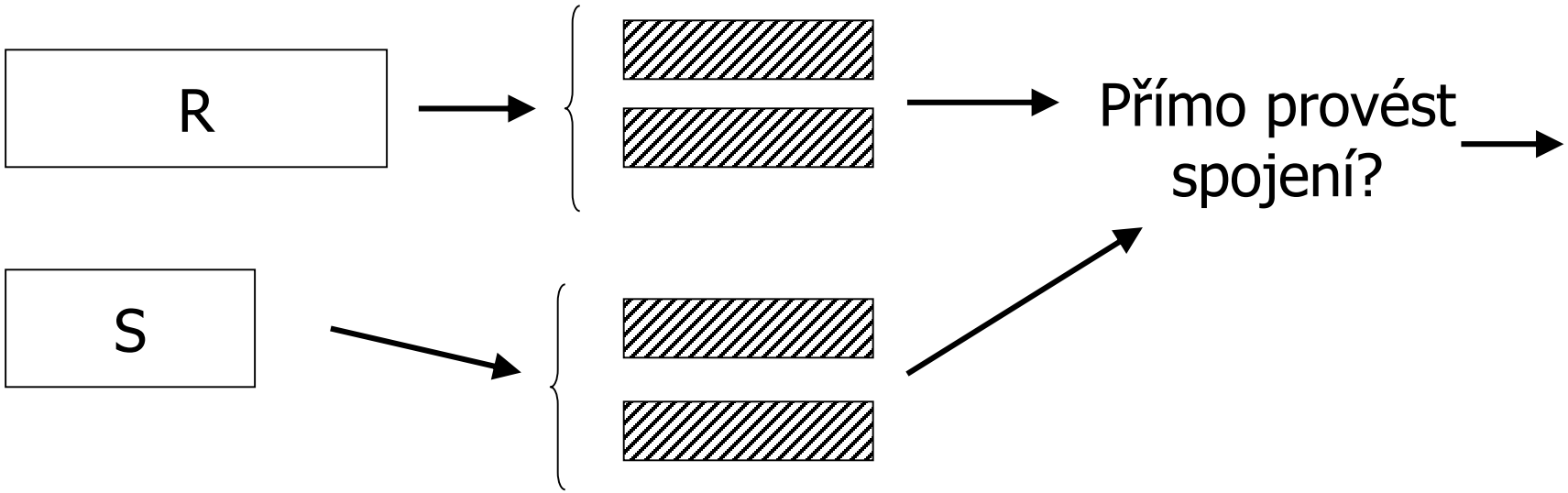
- MergeJoin
  - Předzpracování je drahé
    - Pokud jsou relace uspořádány podle Y, lze vynechat.
- Náklady – analýza
  - MergeJoin
    - lineární složitost
  - Cached Block-based Nested-loop Join
    - kvadratická složitost
  - $\rightarrow$ od jisté velikosti relací je MergeJoin lepší

Algoritmy pro spojení – MergeJoin

- Paměťové nároky
  - Omezení na $\max\big(B(R), B(S)\big) < M^2$
- Optimální paměť
  - Používáme MergeSort na relaci R
    - Počet dávek = $B(R)/M$, Délka dávky = M
    - Omezení: počet dávek $\leq M - 1$
    - $B(R)/M < M \quad \rightarrow \quad B(R) < M^2 \quad \rightarrow \quad M > \sqrt{B(R)}$
- 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

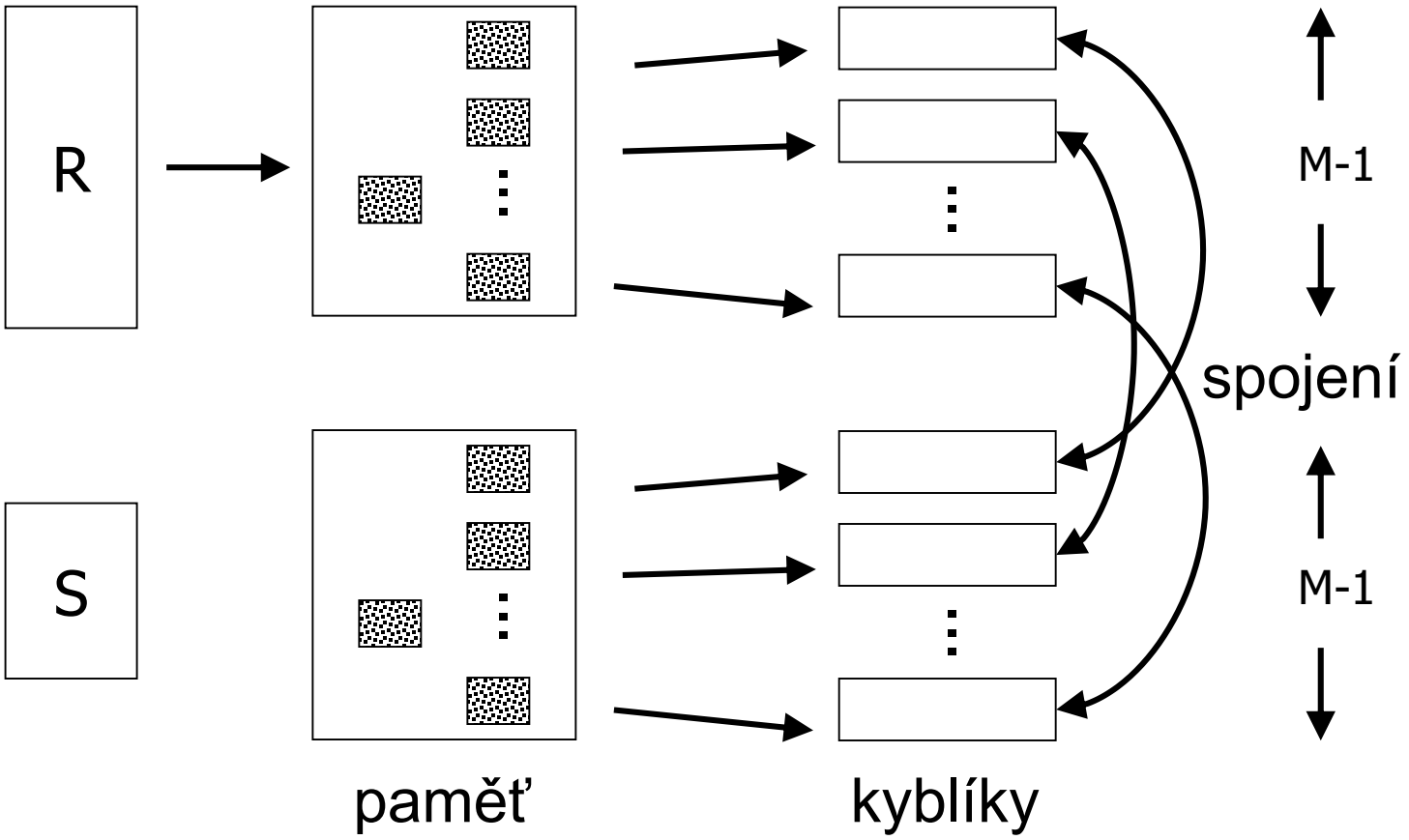
- Vylepšení
  - Vytvoř setříděné dávky R a S
  - Načti první blok z každé dávky (R i S)
  - Zjisti minimální hodnotu v Y
    - Najdi odpovídající záznamy z ostatních dávek
    - Proveď spojení
- Pokud je hodně řádků se stejným Y
  - Aplikuj block-nested-loop join ve zbytku paměti

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
  - $\rightarrow B(R) + B(S) \leq M^2$
- 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⋈S R(X,Y), S(Y,Z)

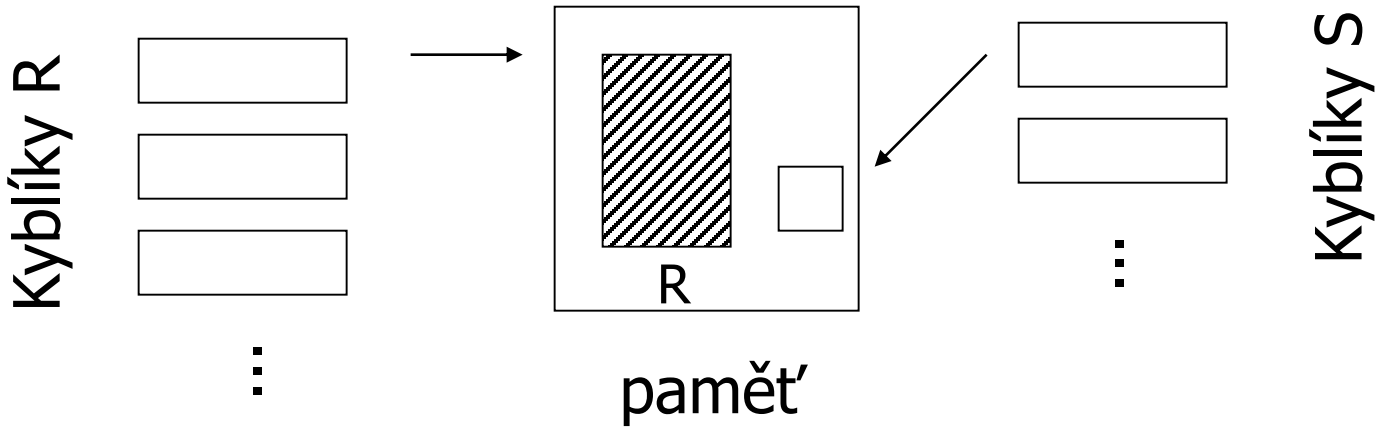


Algoritmy pro spojení – HashJoin

- $R \bowtie S$ 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 R a S
    - Proveď vyhledání odpovídajících si záznamů a jejich spojení

Algoritmy pro spojení – HashJoin

- Spojování kyblíků
  - Kyblík R načti celý (≤ M-2)
    - Pro zrychlení si vytvoř paměťové hašování
  - Kyblík S čti po blocích



Algoritmy pro spojení – HashJoin

- Náklady:
  - Vytvoření hašovaného indexu: 2·(B(R)+B(S))
  - Provedení spojení: B(R)+B(S)
- Omezení:
  - Velikost každého kyblíku R (nebo S) ≤ M-2
    - Odhad: $min\big(B(R), B(S)\big) < (M-1).(M-2)$
- Příklad:
  - Hašování: 2·(1000+500)
  - Spojení: 1000+500
  - Celkem: 4 500 čtení/zápisů

Algoritmy pro spojení – HashJoin

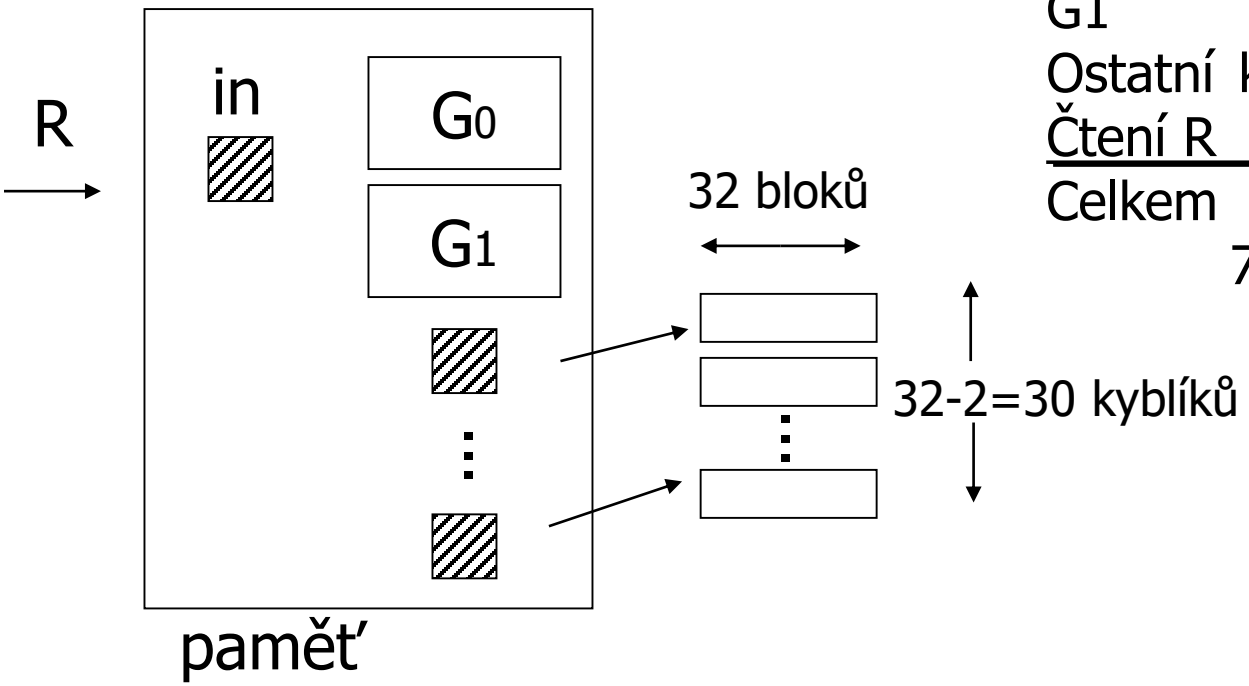
- Minimální paměťové nároky
  - Hašování R, optimální naplnění kyblíků
    - Celkem máme M paměťových bloků
    - Velikost kyblíku
      - B(R) / (M-1)
      - Musí být menší než M (kvůli spojení)
      - $\rightarrow \lceil B(R)/(M-1) \rceil \leq M-2$
    - $\rightarrow M - 1 > \left\lceil \sqrt{B(R)} \right\rceil$

Algoritmy pro spojení – HashJoin

- Optimalizace
  - ponech některé kyblíky v paměti
  - Hybrid HashJoin
- Optimum počtu kyblíků pro R
  - B(R)=1000
  - $\sqrt{B(R)} \approx 32$
  - Tj. 32 bloků má každý kyblík
  - M=102
    - $\rightarrow$ ponechej 2 kyblíky v paměti (64 bloků)
    - $\rightarrow$ zbývá 38 bloků paměti

Alg. pro spojení – Hybrid HashJoin

- Paměť pro vytvoření hašovaného indexu R

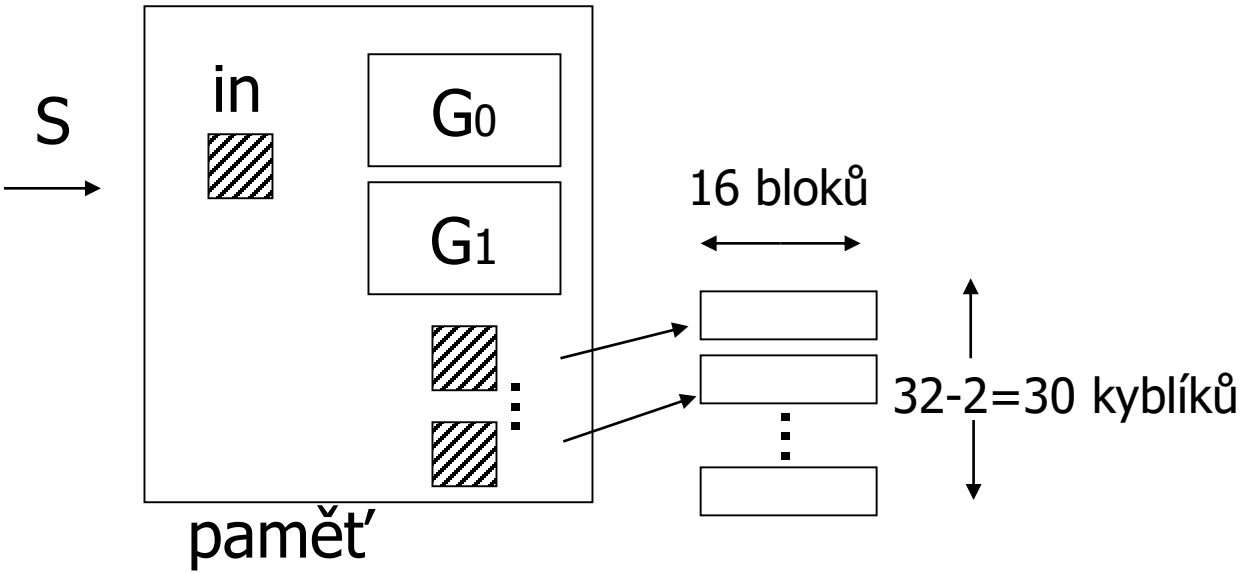


Využití paměti (M=102):	
G0	32 bloků
G1	32 bloků
Ostatní kyblíky	32-2 bloků
Čtení R	1 blok
Celkem	95 bloků

7 bloků je volných!

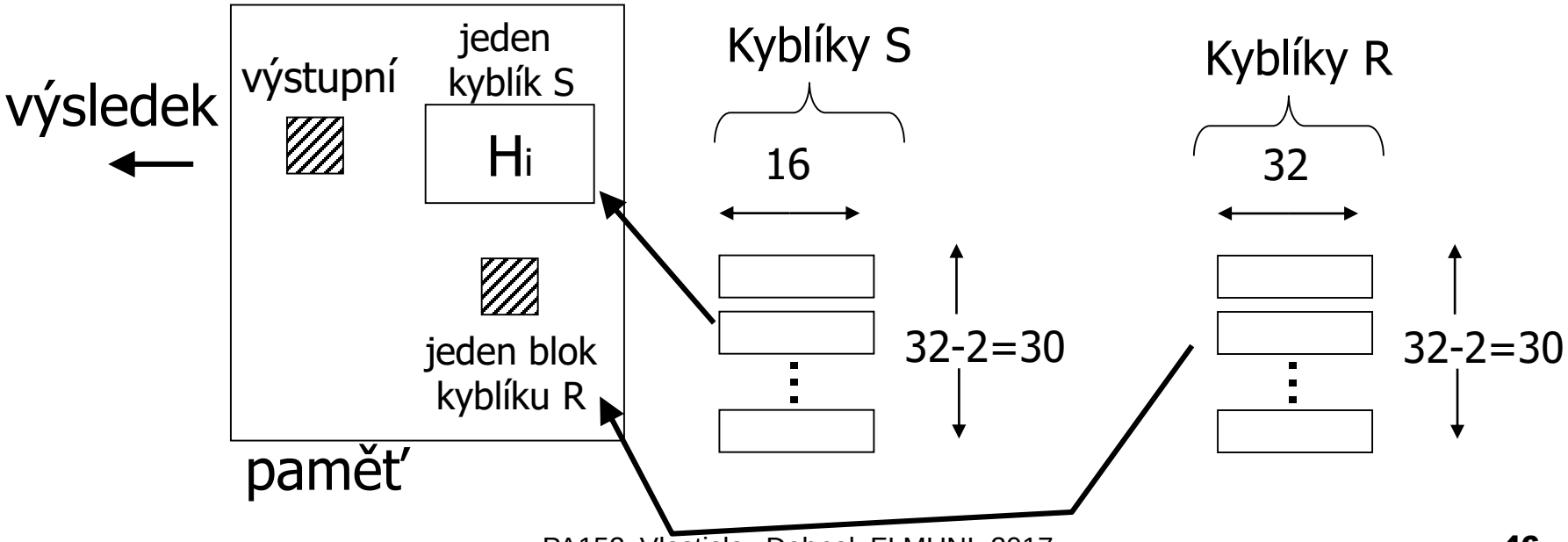
Alg. pro spojení – Hybrid HashJoin

- Paměť pro vytvoření hašovaného indexu S
  - 500/32 = 16 bloků na kyblík
  - Záznamy hašované do kyblíku 0 a 1
    - Vyřešit hned (R je v paměti) → výstup



Alg. pro spojení – Hybrid HashJoin

- Spojení kyblíků
  - Pouze pro kyblíky s id 2-31
  - Načti jeden kyblík celý do paměti, druhý procházej po blocích



Alg. pro spojení – Hybrid HashJoin

- Náklady:
  - Hašování R: 1000 + 30·32 = 1960 čtení/zápisů
  - Hašování S: 500 + 30·16 = 980 čtení/zápisů
    - Pouze 30 kyblíků k zápisu!
  - Spojení: 30·32 + 30·16 = 1440 čtení
    - 2 kyblíky jsou již zpracovány
  - Celkem: 4380 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 → 5000/100 bloků = 50 bloků
    - Spojení – čti R a spojuj
      - Při shodě načti záznam S → 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 $\rightarrow$ 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 + 5000·1·1 = 5500 čtení
    - B) V(R,Y) = 5000 T(R) = 10 000
      rovnoměrné rozložení → 2 záznamy
      Výsledek: 500 + 5000·2·1 = 10500 čtení
    - C) DOM(R,Y)=1 000 000 T(R) = 10 000
      → 10k/1m = 1/100 záznamu
      Výsledek: 500 + 5000·(1/100)·1 = 550 čtení

Algoritmy pro spojení – IndexJoin

- Situace 2
  - Index se nevejde do paměti
  - Index na Y pro R má 201 bloků
    - V paměti 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)·(prohledání indexu + čtení záznamů)
  - Případy:
    - A) → 1 záznam
      Výsledek: 500 + 5000·(0.5+1) = 8000 čtení
    - B) → 2 záznamy
      Výsledek: 500 + 5000·(0.5+2) = 13000 čtení
    - C) → 1/100 záznamu
      Výsledek: 500 + 5000·(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 $\rightarrow$ 550
Hash Join	4500
Hybrid	4380
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) · (1+B(R))	M=3
Cached version	B(S)/(M-2) · (M-2 + B(R))	M≥3
Merge Join (w/o sorting)	B(R) + B(S)	M=3
Merge Join (with sorting)	5 · (B(R) + B(S))	$M = \sqrt{B(R)}$
Sort Join	3 · (B(R) + B(S))	$M = \sqrt{B(R)} + \sqrt{B(S)} + 1$
Index Join (R.Y index) (max costs)	B(S) + T(S) · (HT + θ) e.g. θ = T(R)/V(R,Y)	min. M=4
Hash Join	3 · (B(R) + B(S))	$M = 2 + \sqrt{B(S)}$ max. M-1 buckets
Hybrid	$3(B(R) + B(S)) - \frac{2(B(R) + B(S))}{\lceil \sqrt{B(R)} \rceil}$	$M = \frac{B(R)}{\lceil \sqrt{B(R)} \rceil} + \left(\lceil \sqrt{B(R)} \rceil \right) + 1$
Pointers	B(S)+B(R)+T(R) · θ e.g. θ = T(S)/V(S,Y)	M=B(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
  - Agregační funkce (GROUP BY)
  - Množinové operace

Rušení duplicit

- Postup zpracování
  - Proveď první fázi MergeSort
    - $\rightarrow$ 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 $\mathrm{M} \geq \sqrt{B(R)}$

Agregační funkce

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

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: $\sqrt{B(R) + B(S)} \leq M$
    - Potřebuji 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∩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: $\sqrt{B(R) + B(S)} \leq M$
    - Potřebuji jeden blok pro všechny dávky R i S

Dvouprůchodové algoritmy

- Pomocí hašování
  - Rušení duplicit
  - Agregační funkce (GROUP BY)
  - Množinové operace

Rušení duplicit

- Postup zpracování
  - Proveď hašování *R* do M-1 kyblíků
    - → kyblíky ulož 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)
  - Proveď 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 se skupiny musí vejít do max. M-1.
    - Vypiš výsledky na výstup
- Vlastnosti
  - Náklady: $3B(R)$
  - Omezení: $B(R) \leq (M-1)^2$

lze 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