Definição do Plano de Execução

- Analisar alternativas de processamento
- Escolher a melhor alternativa
- Diversas medidas podem ser consideradas
  - tempo CPU, comunicação, acessos a disco
    - medida mais relevante ("gargalo"): **acessos a disco**
  - para avaliar o custo de uma alternativa
    - análise de estimativas sobre os dados
      - tamanho das tabelas, existência de índices, seletividade, ...
    - custo dos algoritmos de processamento de operações algébricas
      - supõe armazenamento clusterizado de dados e índices
      - supõe que o DD mantém localização física de arquivos de dados e índices

Estimativas sobre os Dados

n_R	número de tuplas na tabela R
t_R	tamanho (em bytes) de uma tupla de R
$t_R(a_i)$	tamanho (em bytes) do atributo a_i de R
f_R	fator de bloco de R (quantas tuplas de R cabem em um bloco *) * bloco: unidade de R / W em disco (medida básica de avaliação) $f_R = \lfloor t_{bloco} / t_R \rfloor$
$V_R(a_i)$	número de valores distintos do atributo a_i de R
$C_R(a_i)$	cardinalidade (estimada) do atributo a_i de R (tuplas de R que satisfazem um predicado de igualdade sobre a_i) (estimando distribuição uniforme: $C_R(a_i) = n_R / V_R(a_i)$)
$GS_R(a_i)$	grau de seletividade do do atributo a_i de R (estimando distribuição uniforme : $GS_R(a_i) = 1 / V_R(a_i)$)
b_R	número de blocos necessários para manter tuplas de R $b_R = \lceil n_R / f_R \rceil$

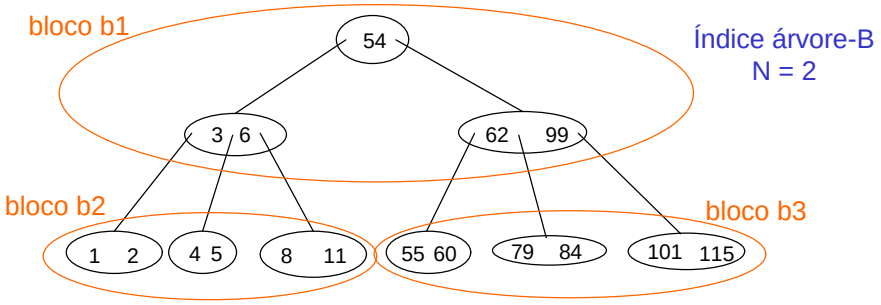
Exemplo de Estimativas de Tabela

- Existem 100 médicos cadastrados na tabela Médicos; cada tupla possui 60 bytes e 1 bloco lê/grava 1 kb
- Estimativas
  - $n_{Médicos}$ = 100 tuplas
  - $t_{Médicos}$ = 60 bytes
  - $f_{Médicos} = \lfloor 1024 / 60 \rfloor = 17$ tuplas
  - $b_{Médicos} = \lceil 100 / 18 \rceil = 6$ blocos

Estimativas sobre os Índices

f_i	fator de bloco do índice *i* (*fan-out* do índice)
h_i	número de níveis (de blocos) do índice para valores de um atributo a_i ("altura" do índice) (assume-se armazenamento clusterizado "em largura") $h_i = \lceil \log_{fi} \lceil V_R(a_i) / N \rceil \rceil$ (para índices árvore-B) (**N** é o número de valores que cabem em um nodo) $h_i = 1$ (para índices *hash*) (assume-se que tabelas *hash*, por não conterem muitos atributos, cabem inteiramente em um bloco)
bf_i	número de blocos de índice no nível mais baixo do índice (número blocos "folha")

Exemplo de Estimativas de Índice



- Estimativas
  - $f_{índice\text{-}CRM}$ = 3 nodos
  - $h_{índice\text{-}CRM} = \lceil \log_{fi} \lceil V_R(a_i) / N \rceil \rceil = \log_3 \lceil 17 / 2 \rceil = 2$
  - $bf_{índice\text{-}CRM} = 2$

Processamento de Seleções (σ)

- Alternativas e suas estimativas de custo
  - A1: pesquisa linear
  - A2: pesquisa binária
  - A3: índice primário para atributo chave
  - A4: índice primário para atributo não-chave
  - A5: índice secundário para atributo chave
  - A6: índice secundário para atributo não-chave
  - A7: desigualdade (>, >=) com índice primário
  - A8: desigualdade (<, =) com índice primário
  - A9: desigualdade com índice secundário

Pesquisa Linear (A1)

- Varre todo o arquivo para buscar os dados desejados
  - acessa todos os blocos do arquivo
- Em alguns casos, é a única alternativa possível
- Custo para uma tabela *R*
  - custo = b_R

Pesquisa Binária (A2)

- Aplicado sobre uma tabela *R* quando
  - dados estão ordenados pelo atributo de seleção a_i
  - há uma condição de igualdade sobre a_i
- Custo
  - custo para acessar o bloco da 1ª tupla: $\lceil \log_2 b_R \rceil$
  - custo para acessar os blocos das demais tuplas: $\lceil (C_R(a_i) / f_R) \rceil - 1$ → desconta-se o bloco da primeira tupla (já foi localizada)
  - custo = $\lceil \log_2 b_R \rceil + \lceil (C_R(a_i) / f_R) \rceil - 1$
  - se a_i é chave: custo = $\lceil \log_2 b_R \rceil$

Seleções Utilizando Índices

- Atributo a_i com índice primário
  - leitura do índice corresponde à leitura na ordem física do arquivo
    - arquivo fisicamente ordenado por valores de a_i
  - se a_i é chave (A3)
    - custo = $h_i + 1$ → acesso ao bloco onde está a tupla com o valor de a_i
  - se a_i é não-chave (A4)
    - custo = $h_i + \lceil (C_R(a_i) / f_R) \rceil$ → número de blocos contíguos acessados a partir do 1º bloco que contém o valor da chave

Seleções Utilizando Índices

- Atributo a_i com índice secundário
  - arquivo não está fisicamente ordenado por valores de a_i
  - se a_i é chave (A5)
    - custo = $h_i + 1$
  - se a_i é não-chave (A6)
    - supor que o bloco folha do índice aponta para uma lista de apontadores para as tuplas desejadas
      - estimar que esta lista cabe em um bloco
    - custo = $h_i + 1 + C_R(a_i)$ → **pior caso:** cada tupla com o valor desejado está em um bloco ≠

acesso adicional à lista de apontadores

Exercício 1

- Dado Pac(codp, nome, idade, cidade, doença) e as seguintes estimativas: n_{Pac} = 1000 tuplas; t_{Pac} = 100 bytes; V_{Pac}(codp) = 1000; V_{Pac}(doença) = 80; V_{Pac}(idade) = 700; um índice primário árvore-B para codp (I1) com N = 5; f_{I1} = 10; um índice secundário árvore-B para doença (I2) com N = 3; f_{I2} = 5; e 1 bloco = 2 kb
- Supondo a seguinte consulta:

$$\sigma_{\text{doença = 'câncer'}}(\text{Pac})$$

a) qual a melhor estratégia de processamento para σ ?

b) se agora 1 bloco = 8 kb, a estratégia escolhida no item anterior continua sendo a melhor?

Comparação por Desigualdade

- Supõe-se que aproximadamente metade das tuplas satisfazem a condição
  - $a_i <= x \Rightarrow$ número de tuplas $\approx \lceil n_R / 2 \rceil$
- DD mantém valores mínimo/máximo de a_i
  - $a_i <= x$
    - número de tuplas = 0, se x < MIN(a_i)
    - número de tuplas = n_R, se x >= MAX(a_i)
  - $a_i >= x$
    - número de tuplas = 0, se x > MAX(a_i)
    - número de tuplas = n_R, se x <= MIN(a_i)

Desigualdade e Índices

- Atributo a_i com índice primário
  - comparações do tipo $a_i > x$ ou $a_i >= x$ (A7)
    - custo para buscar $a_i = x$ através do índice: h_i
    - custo (médio) para varredura do arquivo: $\lceil b_R / 2 \rceil$
    - custo = $h_i + \lceil b_R / 2 \rceil$
  - comparações do tipo $a_i < x$ ou $a_i <= x$ (A8)
    - varre o arquivo até $a_i = x$
    - custo (médio) = $\lceil b_R / 2 \rceil$

Desigualdade e Índices

- Atributo a_i com índice secundário (A9)
  - custo para buscar $a_i = x$ através do índice: h_i
  - custo para varredura dos blocos folha do arquivo de índice (em média, metade dos blocos é acessado): $\lceil bf_i / 2 \rceil$
  - custo para varredura das listas de apontadores em cada bloco folha: $\lceil bf_i / 2 \rceil * f_i * N$
  - custo para acesso a blocos de dados: $\lceil n_R / 2 \rceil$
  - custo = $h_i + \lceil bf_i / 2 \rceil + \lceil bf_i / 2 \rceil * f_i * N + \lceil n_R / 2 \rceil$

cada bloco possui $\mathbf{f_i}$ nodos e cada nodo com **N** listas de apontadores

pior caso: cada tupla em um bloco ≠ e, em média, metade dos dados atende a condição

Exercício 2

- Considere a relação Pac e as estimativas dadas no exercício 1
- Dada a consulta

$$\sigma_{codp > 10000 \wedge cidade = \text{'Florianópolis'}} (Pac)$$

a) qual a melhor estratégia de processamento para σ ?

b) supondo agora a existência de um índice secundário árvore-B para cidade (I3) com $N = 3$, $f_{I3} = 5$, $bf_{I3} = 10$ e $V_{Pac}(cidade) = 100$, qual a melhor estratégia de processamento para σ ?

Conjunções – Estimativa de Tamanho

- Dada uma seleção $\sigma_{c1 \wedge c2 \wedge ... \wedge cn} (R)$
  - estima-se a cardinalidade de cada condição c_i
    - $C(c_i)$
  - tamanho da relação resultante é dado por
    - $\lceil n_R . (C(c_1). C(c_2). ... . C(c_n)) / (n_R)^n \rceil$
- Exemplo

  R(a, b, c) $n_R = 100$ tuplas $V_R(a) = 100$ $V_R(b) = 20$

  Dado $\sigma_{a > 5 \wedge b = 10}$, temos:

  $C(a>5) = \lceil n_R / 2 \rceil = 50$ tuplas

  $C(b=10) = \lceil n_R / V_R(b) \rceil = 5$ tuplas

  Estimativa tamanho $= \lceil 100 (50.5) / 100^2 \rceil = 3$ tuplas

Disjunções – Estimativa de Tamanho

- Dada uma seleção $\sigma_{c1 \vee c2 \vee ... \vee cp}$
  - tamanho da relação resultante é dado por

    $\lceil n_R .(1 – (1 - C(c_1) / n_R).(1 - C(c_2) / n_R). ... .(1 - C(c_p) / n_R)) \rceil$
- Exemplo

  R(a, b, c) n_R = 100 tuplas $V_R(a) = 100$ $V_R(b) = 20$

  Dado $\sigma_{a > 5 \vee b = 10}$, temos:

  $C(a>5) = \lceil n_R / 2 \rceil = 50$ tuplas

  $C(b=10) = \lceil n_R / V_R(b) \rceil = 5$ tuplas

  Estimativa tamanho = $100.(1 – (1 – 50/100).(1 – 5/100))$

  $= 53$ tuplas

Negações – Estimativa de Tamanho

- Dada uma seleção $\sigma_{\neg\theta}$
  - tamanho da relação resultante é dado por

    n_R – estimativaTamanho(σ_θ)
- Exemplo

  R(a, b, c) n_R = 100 tuplas $V_R(a) = 100$ $V_R(b) = 20$

  Dado $\sigma_{\neg(a > 5 \vee b = 10)}$, temos:

  Estimativa tamanho($\sigma_{a > 5 \vee b = 10}$) = 53 tuplas

  Estimativa tamanho($\sigma_{\neg(a > 5 \vee b = 10)}$) = 100 – 53 = 47 tuplas

Processamento de Produtos ("X")

- Estimativa de tamanho (R "X" S)
  - produto cartesiano (R X S)
    - tamanho = $n_R * n_S$
  - junção por igualdade (*"equi-join"* – natural ou theta)
    - junção natural sem atributo em comum
      - tamanho = $n_R * n_S$
    - junção por referência (*fk(R) = pk(S)*)
      - tamanho estimado <= n_R
    - junção entre chaves candidatas (atributos *unique*)
      - tamanho <= MIN (n_R , n_S)

Processamento de Produtos ("X")

- Estimativa de tamanho (R "X" S)
  - junção por igualdade (*"equi-join"* – natural ou theta)
    - junção entre atributos não-chave ($a_i(R) = a_j(S)$)
      - cada tupla de *R* associa-se com $C_S(a_j)$
      - se tenho n_R tuplas $\Rightarrow \lceil n_R * C_S(a_j) \rceil$
      - idem para as tuplas de S: $\lceil n_S * C_R(a_i) \rceil$
      - tamanho estimado = MIN($\lceil n_R * C_S(a_j) \rceil$, $\lceil n_S * C_R(a_i) \rceil$)
        - » menor estimativa geralmente é mais precisa
  - junção theta por desigualdade ($a_i(R) > a_j(S)$)
    - estimativa: cada tupla de R > $\lceil n_s / 2 \rceil$ tuplas de S e vice-versa
    - tamanho estimado = MAX($n_R * \lceil n_s / 2 \rceil$, $n_S * \lceil n_R / 2 \rceil$) (pior caso)

Processamento de Produtos (“X”)

- Alternativas e suas estimativas de custo
  - A1: laço aninhado *(“nested-loop”)*
  - A2: laço aninhado com índice
  - A3: merge-junção *(“balanced-line”)*
  - A4: hash-junção

Laço Aninhado (A1)

- Dois laços para varredura de blocos das relações a serem combinadas

```
para cada bloco B_R de R faça
        para cada bloco B_S de S faça
        início
                se uma tupla t_R ∈ B_R satisfaz a condição de
                junção com uma tupla t_S ∈ B_S então
                        adicione t_R * t_S ao resultado
        fim
```

Laço Aninhado - Custo

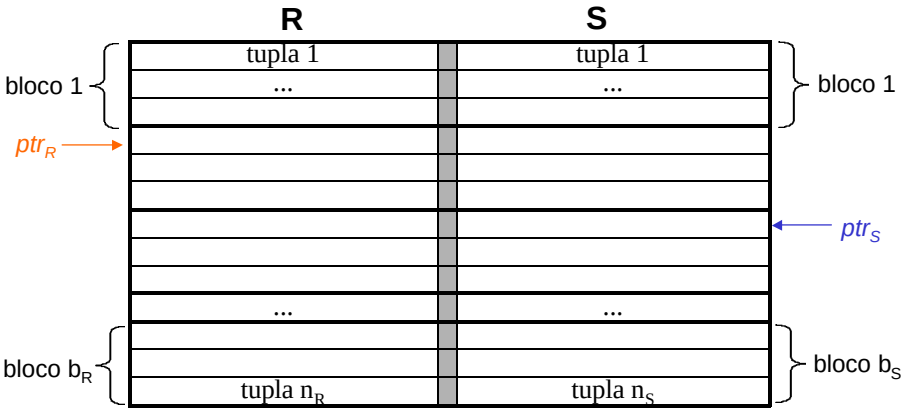
- Melhor caso
  - os blocos de R e S cabem todos na memória
  - custo = $b_R + b_S$
- Pior caso
  - apenas um bloco de cada relação pode ser lido por vez
  - custo = MIN($b_R + b_R * b_S$, $b_S + b_S * b_R$)

Laço Aninhado com Índice (A2)

- Aplicada se existir um índice para o atributo de junção do laço interno
- Custo
  - para cada tupla externa de R, pesquisa-se o índice para buscar a tupla de S
  - custo diretamente associado ao tipo de índice
  - exemplo com índice primário árvore-B para atributo chave em S (I_S)
    - custo = $\mathbf{b_R + n_R * (h_{Is} + 1)}$

Merge-Junção (A3)

- Aplicada se R e S estiverem fisicamente ordenadas pelos atributos de junção



Merge-Junção - Custo

- Pressupõe que pelo menos um bloco de cada relação cabe na memória
  - geralmente isso é possível
  - exige uma única leitura de cada relação
  - $custo_{M\text{-}J} = b_R + b_S$
- Se R e/ou S não estiverem ordenadas, elas podem ser ordenadas
  - custo = custo ordenação R e/ou S + $custo_{M\text{-}J}$

Exercício 3

- Proponha um algoritmo de alto nível para executar a alternativa merge-junção

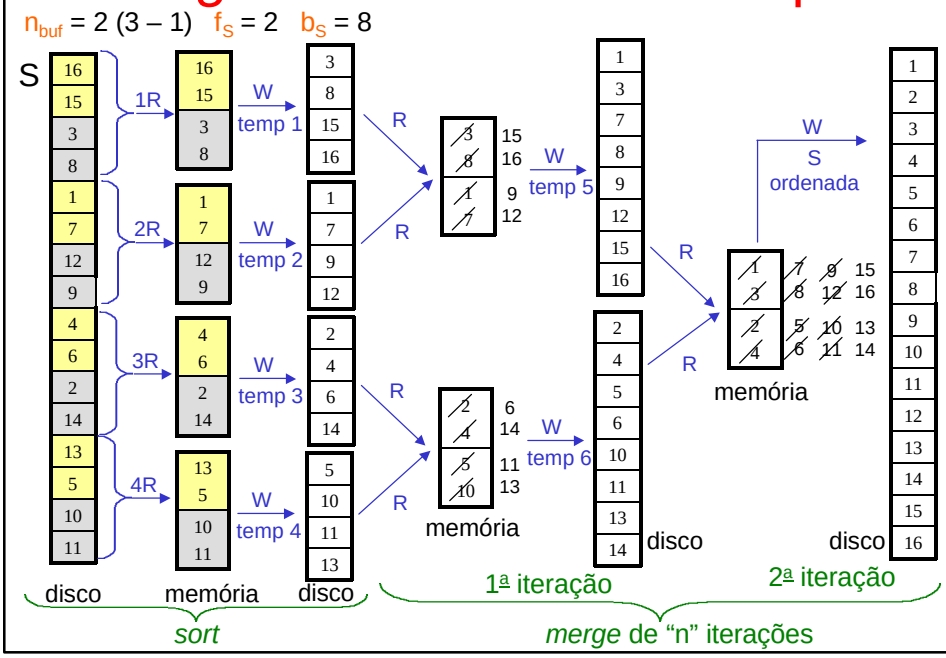
Ordenação Externa

- Ordenação interna
  - ordenação feita totalmente em memória
- Ordenação externa
  - ordenação na qual os dados não cabem inteiramente na memória
  - útil no processamento de consultas
    - exibição ordenada de dados (ORDER BY)
    - avaliação de planos de execução
  - técnica mais utilizada para ordenação de relações
    - ***MergeSort* Externo**

MergeSort Externo

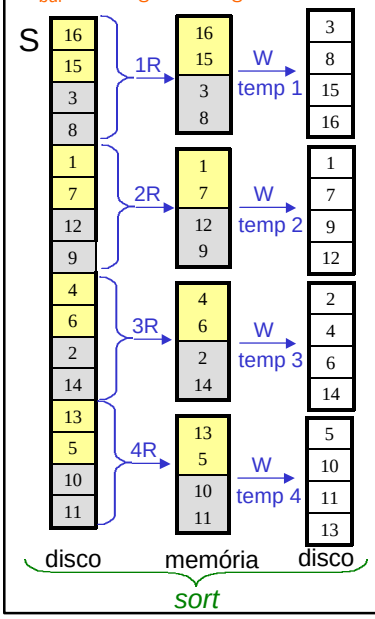
- Executa em 2 etapas
- Etapa 1 – *Sort*
  - ordena partições da relação em memória
    - tamanho da partição depende da disponibilidade de *buffers* em memória (n_{buf} = nº de *buffers* disponíveis)
    - gera um arquivo temporário ordenado para cada partição
- Etapa 2 – *Merge* de "n" iterações
  - ordena um conjunto de temporários a cada iteração
    - gera um novo temporário resultante da ordenação
    - ordenação termina quando existir somente um temporário que mantém a relação inteira ordenada

MergeSort Externo - Exemplo



MergeSort Externo - Custo

$n_{buf} = 2$ $f_S = 2$ $b_S = 8$



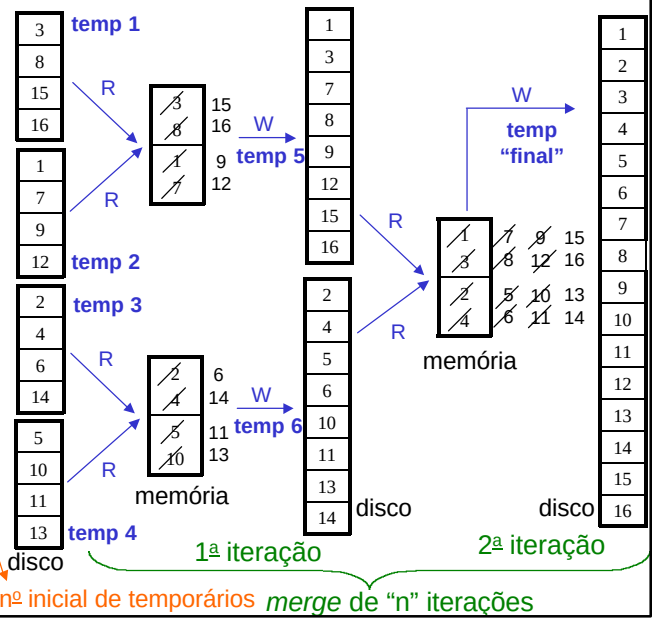
- 1 R + 1 W de todos os blocos da relação S
- custo = **$2 * b_S$**

MergeSort Externo - Exemplo

$n_{buf} = 2$ $f_S = 2$ $b_S = 8$

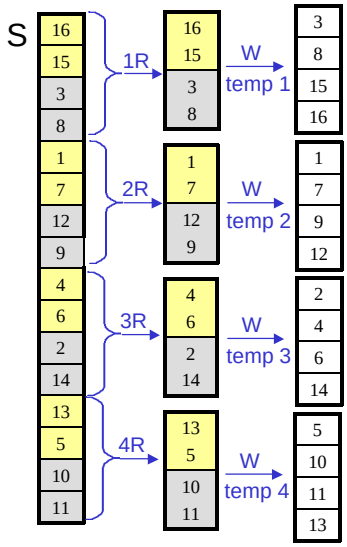
S ordenada

- Nº de iterações é dependente do nº de temporários a ordenar
- A cada iteração, o nº de temporários se reduz a um fator de n_{buf}
  - reserva-se 1 buffer para cada temporário. Logo ordena-se n_{buf} temporários a cada iteração
  - nº iterações: $\log_{n_{buf}} (b_S / n_{buf})$
  - 1 R + 1 W a cada iteração: $2 * b_S$
- custo = **$2 * b_S * \log_{n_{buf}} (b_S / n_{buf})$**



MergeSort Externo - Custo

$n_{buf} = 2$ $f_S = 2$ $b_S = 8$



Custo total = $2 * b_S +$
$2 * b_S * \log n_{buf} (b_S / n_{buf})$
= **$2 * b_S$ $(\log n_{buf} (b_S / n_{buf}) + 1)$**

Exemplo = $2 . 8 (\log_2 (8 / 2) + 1)$
$= 16 (2 + 1) = 48$ acessos

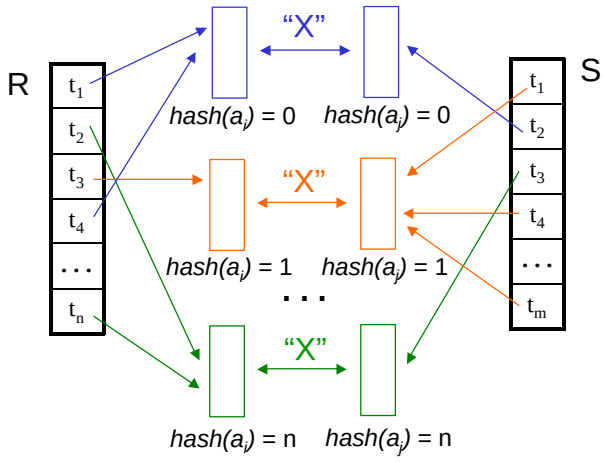
Merge-Junção - Custo

- Se ambas as relações (R e S) estão ordenadas
  - custo = $b_R + b_S$
- Se uma delas (R) não está ordenada
  - custo = $2 * b_R$ $(\log n_{buf} (b_R / n_{buf}) + 1) + b_R + b_S$
- Se ambas as relações não estão ordenadas
  - custo = $2 * b_R$ $(\log n_{buf} (b_R / n_{buf}) + 1) +$
    $2 * b_S$ $(\log n_{buf} (b_S / n_{buf}) + 1) +$
    $b_R + b_S$

Hash-Junção

- Aplicada se existir um índice *hash* com a mesma função definido para os atributos de junção de R e S
- Executa em 2 etapas
  1. Particionamento
     - separa em partições as tuplas de R e S que possuem o mesmo valor para a função de *hash*
  2. Junção
     - analisa e combina as tuplas de uma mesma partição

Hash-Junção - Funcionamento



Hash-Junção - Custo

- Fase de Particionamento
  - lê R e S e as reescreve, organizadas em partições
    - sempre que um conjunto de tuplas com o mesmo valor de *hash* adquire o tamanho de um bloco, este bloco é anexado a um arquivo temporário para a partição
    - considera-se geralmente um melhor caso
      - função *hash* distribui uniformemente os valores das tuplas
        - » evita escrita de muitas pequenas partições. Assim, assume-se custo "W" = custo "R" e não custo "W" > custo "R"
    - custo = $2 * b_R + 2 * b_S$ = **$2 * (b_R + b_S)$**

Hash-Junção - Custo

- Fase de Junção
  - lê as partições de mesmo *hash* e combina as tuplas
    - equivale aproximadamente a uma nova leitura de todos os blocos de R e S
    - custo = **$(b_R + b_S)$**
- Custo Total
  - custo = $2 * (b_R + b_S) + (b_R + b_S) = 3 * (b_R + b_S)$

Escrita ("W") do Resultado

- Qualquer alternativa de processamento deve considerar este custo
  - b_{res} = número de blocos de resultado a ser "W"
- Exemplo: estimativa de "W" do resultado de um produto
  - $b_{res} = \lceil tamanhoProduto / f_{res} \rceil$
  - estimativa do fator de bloco do resultado (f_{res})
    - $f_{res} = \lfloor tamanhoBloco / (t_R + t_S) \rfloor$

arredonda "para baixo" pois uma tupla do resultado não pode estar parcialmente escrita em um bloco

Exemplo

Med(<u>CRM</u>, nome, ...) com $n_{Med} = 50$ e $t_{Med} = 50$ b

Cons(<u>CRM, codp</u>, ...) com $n_{Cons} = 500$ e

$t_{Cons} = 20$ b e 1 bloco = 2 kb

Dado Med X $\theta = \sigma_{Med.CRM = Cons.CRM}$ Cons, temos:

- junção por referência (*fk(Cons)* = *pk(Med)*)
  - tamanho resultado = $n_{Cons} = 500$ tuplas
- $f_{res} = \lfloor tamanhoBloco / (t_R + t_S) \rfloor$
  - $f_{res} = \lfloor 2048 / (50 + 20) \rfloor = 29$ tuplas
- $b_{res} = \lceil tamanhoResultado / f_{res} \rceil$
  - $b_{res} = \lceil 500 / 29 \rceil = 18$ blocos

Tamanho de Buffer

- Influencia o custo
  - quanto maior o número de *buffers* (n_{buf}) para blocos, melhor!
- Exemplos de custos de produtos
  - se $n_{buf} >= (b_R + b_S + b_{res}) \Rightarrow$ custo = $b_R + b_S$
    (não é preciso "W" o resultado)
  - se n_{buf} é capaz de manter R e S, mas apenas 1 bloco p/ o resultado $\Rightarrow$ custo = $b_R + b_S + (b_{res} - 1)$

Exemplo

Med(CRM, nome, ...) Cons(CRM, codp, ...)

$b_{Med} = 10$; $b_{Cons} = 20$; $n_{buf} = 5$

Dado Med X $\theta = \sigma_{Med.CRM = Cons.CRM}$ Cons, temos:

$b_{res} = 18$ blocos

um bloco do resultado pode ficar na memória

- Custo do laço aninhado (s/ considerar *buffers*)

  custo = $b_{Med} + b_{Med} * b_{Cons} + (b_{res} - 1) = 10 + 10*20 + 17 = 227$

- Custo do laço aninhado (considerando 3 *buffers* p/ Med, 1 *buffer* p/ Cons e 1 *buffer* para o resultado)
  - melhor manter em memória + blocos da relação menor

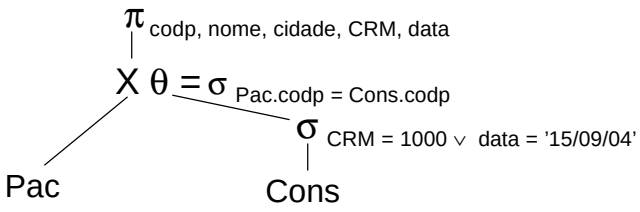
  custo = $b_{Med} + \lceil b_{Med} / 3 \rceil * b_{Cons} + (b_{res} - 1) = 10 + 4*20 + 17 = 107$

reduz em 1/3 o número de acessos a blocos da tabela Cons

Exercício 4

Dado **Pac(<u>codp</u>, nome, idade, cidade, doença)** e **Cons(<u>CRM, codp, data</u>, hora)** e as seguintes estimativas: n_{Pac} = 500 tuplas; t_{Pac} = 50 bytes; $t_{Pac}(codp)$ = 5 bytes; $t_{Pac}(nome)$ = 15 bytes; $t_{Pac}(cidade)$ = 15 bytes; n_{Cons} = 1000 tuplas; t_{Cons} = 20 bytes; $t_{Cons}(CRM)$ = 5 bytes; $t_{Cons}(data)$ = 10 bytes; $V_{Cons}(data)$ = 50; $V_{Cons}(codp)$ = 500; $V_{Cons}(CRM)$ = 200; um índice primário árvore-B para *codp* (I1) em **Pac** com N = 10 e f_{I1} = 10; um índice secundário hash para *codp* (I2) em **Cons**; um índice secundário hash para *CRM* (I3) em **Cons**; **Pac** está ordenada pelo *codp*; **Cons** está ordenada pela *data*; 1 bloco = 1 kb e n_{buf} = 3

Dada a seguinte árvore algébrica de consulta:



Estime custos e o tamanho do resultado desta consulta.

Produtos Complexos - Custo

- Dada uma operação produtória complexa conjuntiva R "X"$_{\theta = \sigma\ c1 \wedge c2 \wedge \ldots \wedge cn}$ S
  - estima-se o custo de cada condição c_i
    - R "X"$_{\theta = \sigma\ ci}$ S
  - escolhe-se a condição c_i de menor custo para ser implementada
    - as demais condições $c_1, c_2, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n$ são verificadas a medida que as tuplas de R "X"$_{\theta = \sigma\ ci}$ S são geradas

Produtos Complexos - Custo

- Dada uma operação produtória complexa disjuntiva R "X"$_{\theta = \sigma\ c1 \vee c2 \vee ... \vee cn}$ S, tem-se as seguintes alternativas
  - aplica-se o algoritmo de laço aninhado
    - mais simples e independente de condição de junção
  - aplica-se (R "X"$_{\theta = \sigma\ c1}$S) $\cup$ (R "X"$_{\theta = \sigma\ c2}$S) $\cup$... $\cup$ (R "X"$_{\theta = \sigma\ cn}$S)
    - custo total é a soma dos menores custos de cada junção individual

Processamento de Projeções (π)

- Custo (na teoria) de $\pi_{a1, a2, ..., an}$ (R)
  - custo = (1) varredura de R + (2) eliminação de duplicatas
    - custo de (1) = b_R (gera b_{Res} blocos de resultado)
    - custo de (2) = custo de classificar o resultado pelos atributos da projeção = $2 * b_{Res}\ (\log n_{buf} (b_{Res} / n_{buf}) + 1)$
      - tuplas iguais estarão adjacentes e apenas uma delas é mantida (deve-se ainda varrer o resultado ordenado)
    - **custo = $b_R + 2 * b_{Res}\ (\log n_{buf} (b_{Res} / n_{buf}) + 1) + b_{Res}$**
- Custo (na prática) de $\pi_{a1, a2, ..., an}$ (R)
  - custo = b_R
    - SQL não faz eliminação de duplicatas

Processamento de Projeções (π)

- Tamanho de $\pi_{a1, a2, ..., an}(R)$ (na prática)
  - tamanho = $n_R * (t_R(a_1) + ... + t_R(a_n))$
- Na teoria, é difícil estimar o tamanho do resultado pois é difícil estimar quantas duplicatas serão eliminadas
  - o que é possível estimar?
    - se a projeção é apenas da chave primária (*pk(R)*)
      - tamanho = $n_R * t_R(pk(R))$
    - se a projeção é de um único atributo a_i
      - tamanho = $V_R(a_i) * t_R(a_i)$

Processamento de Operações de Conjunto ($\cup$, — e $\cap$)

- Aplica-se uma estratégia *merge-junção*
  - (1) classificação de R e S
    - facilita a verificação de tuplas iguais em R e S
  - (2) varredura de R e S para obtenção do resultado
  - custo (pior caso) = $2 * b_R (\log_{n_{buf}} (b_R / n_{buf}) + 1) + 2 * b_S (\log_{n_{buf}} (b_S / n_{buf}) + 1) + b_R + b_S$

Processamento de Operações de Conjunto ($\cup$, — e $\cap$)

- Estimativas de tamanho
  - pior caso
    - tamanho ($R \cup S$) = $n_R + n_S$
    - tamanho ($R - S$) = n_R
    - tamanho ($R \cap S$) = MIN(n_R , n_S)
  - melhor caso
    - tamanho ($R \cup S$) = MAX(n_R , n_S)
    - tamanho ($R - S$) = 0
    - tamanho ($R \cap S$) = 0
  - caso médio
    - média aritmética do melhor e pior casos

Funções de Agregação e *Group By*

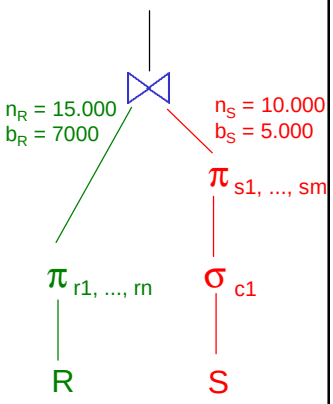
- Função de agregação (*count*, *max*, *sum*, ...)
  - custo da varredura da relação R = b_R
  - tamanho = *lenght* (*int* ou *float*)
- *Group By* + Função de Agregação
  - processamento: ordenação de R pelos atributos de agrupamento + varredura de R ordenada para definir grupos e aplicar função
    - custo = $2 * b_R$ (log n_{buf} (b_R / n_{buf}) + 1) + b_R
  - tamanho de *group by* $a_1, ..., a_n$
    - número de grupos * ($t_R(a_1)$ + ... + $t_R(a_n)$) + *lenght* (*int* ou *float*)

Índice Temporário

- Um índice temporário pode ser criado para o processamento de uma operação algébrica op_x
- Objetivo
  - gerar um custo menor que outras alternativas de processamento de op_x
  - este custo envolve
    - "W" total ou parcial dos blocos do índice no disco
    - acesso a ele durante o processamento de op_x
  - estes custos devem ser estimados antes da criação do índice, para decidir por criá-lo ou não

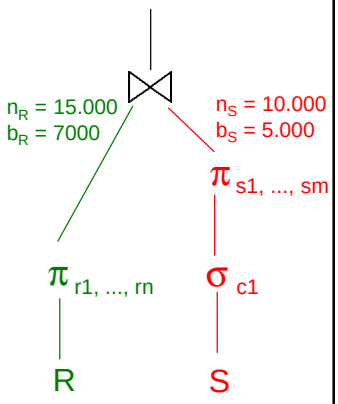
Índice Temporário - Motivação

- Processamento da junção
  - A1: laço aninhado
    - custo = $b_S + b_S * b_R$
      = 5 + 5 * 7 = 40 mil acessos
  - A3: merge-junção ($n_{buf} = 3$)
    - custo = ordenação de R + ordenação de S + $b_R + b_S$
      = 126 + 80 + 7 + 5 = 218 mil acessos
  - e se houvesse um índice *Ix* sobre o resultado de R? Poderíamos estimar A2: laço aninhado indexado
    - custo = $b_S + n_s * (h_{Ix} + 1)$
    - se *Ix* tiver $h_{Ix} < 3$, A2 será a alternativa de menor custo! Exemplo: $h_{Ix} = 2$:
      - custo = 5 + 10 * (2+1) = 35 mil acessos



Índice Temporário - Exemplo

- Avaliando custo de criação de índice árvore-B sobre o resultado de R
  - supondo que o atributo de junção em R é chave, deve-se indexar 15.000 dados
  - supondo que se consegue um máximo de f_I = 55 nodos, com N = 50 valores, temos:
    - nível 0 ⇒ indexa 50 valores
    - nível 1 ⇒ indexa 51x50 = 2.550 valores
    - nível 2 ⇒ indexa 51x51x50 = 130.050 valores (máximo 3 níveis na árvore-B)
    - se f_I = 55, o primeiro nível (1 nodo) e o segundo nível (51 nodos) da árvore podem ficar em um bloco e os restantes em outros blocos. Logo, teremos no máximo 2 acessos (h_I = 2)! Vale a pena criar o índice!
    - custo total de A2 = 35 mil + "W" do índice
    - custo "W" do índice = 15.000 valores / N = 300 nodos / f_I = "W" de 6 blocos de índice (pior caso – o índice não cabe na memória)

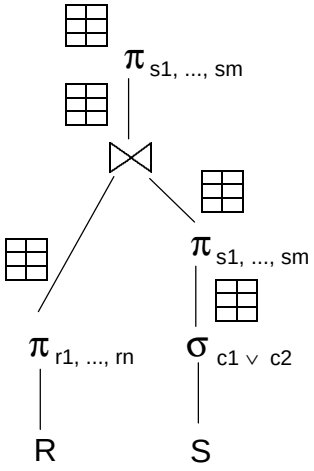


Materialização X *Pipeline*

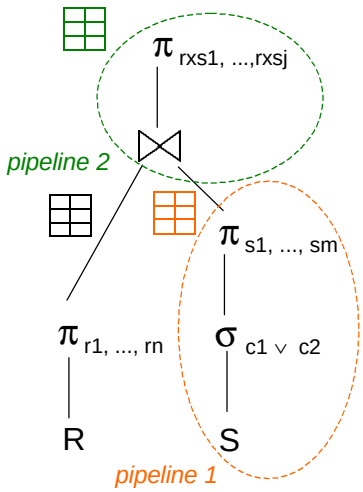
- Materialização
  - cada operação da álgebra é materializada em uma relação temporária (se necessário) e utilizada como entrada para a próxima operação
  - situação *default* no processamento de consultas
- *Pipeline*
  - uma seqüência de operações algébricas é executada em um único passo
    - cada tupla gerada por uma operação é passada para a operação seguinte
      - cada tupla passa por um canal (*pipe*) de operações
    - somente o resultado ao final do *pipeline* é materializado (se necessário)

Materialização X *Pipeline*

Materialização



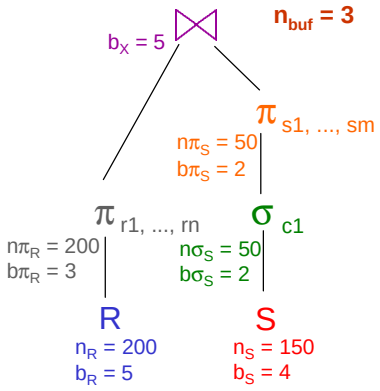
Definição de *Pipelines*



Pipeline de Operações

- \+ : evita a materialização de todos os resultados intermediários no processamento de uma consulta
- \- : resultado não é passado de forma completa para uma próxima operação dentro do *pipeline*
  - algoritmos de processamento das operações algébricas deve ser modificados para invocar outras operações para cada tupla gerada
    - algoritmos "dinâmicos"
  - algumas alternativas não podem ser estimadas
    - exemplos: merge-junção; operações de conjunto
      - exigem um resultado completo e ordenado para processar

Exemplo: um Produto sem *Pipeline*

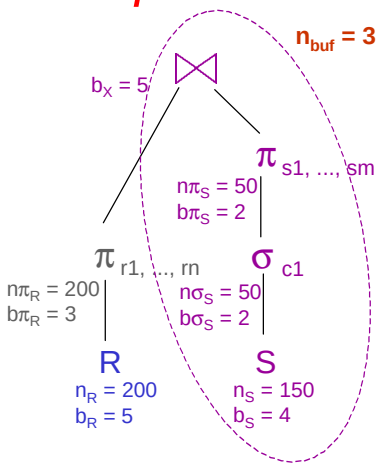


1) custo σ (pior caso) = 4 acessos
   (não é preciso "W" resultado)
2) custo π = 0 (tudo em memória)
   custo "W" resultado = 1 acesso
   (reserva apenas 1 buffer para os dados que vêm de S)
3) custo π = 5 acessos
   custo "W" resultado = 2 acessos
   (reserva apenas 1 buffer para os dados que vêm de R)
4) custo ⋈ (pior caso – laço aninhado)
   $= (b\pi_S - 1) + b\pi_S * b\pi_R = 1 + 2*3 = 7$ acessos
   custo "W" resultado = $b_x - 1 = 4$ acessos

CUSTO TOTAL = 4+1+5+2+7+4 = 23 acessos

Produto dentro de um *Pipeline*

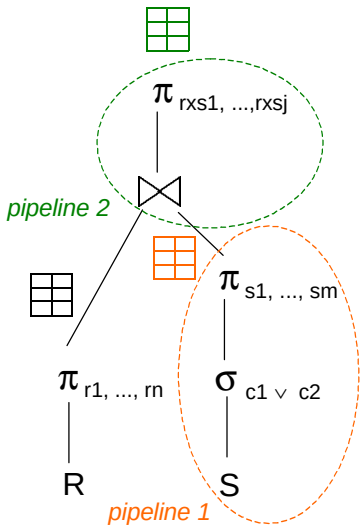
- o produto vai recebendo uma a uma as tuplas filtradas de S
- as tuplas de S não são recebidas ordenadas pelo atributo de junção
  - não dá para usar merge-junção
- custo ⋈ (pior caso - laço aninhado):
  $= b_S + n\pi_S * b\pi_R$
  $= 4 + 50 * 3$
  $= 154 + 4$ "W" = 158 acessos
- custo π = 5 acessos
  custo "W" resultado = 2 acessos
  (reserva apenas 1 buffer para os dados que vêm de R)



CUSTO TOTAL = 158+5+2 = 165 acessos

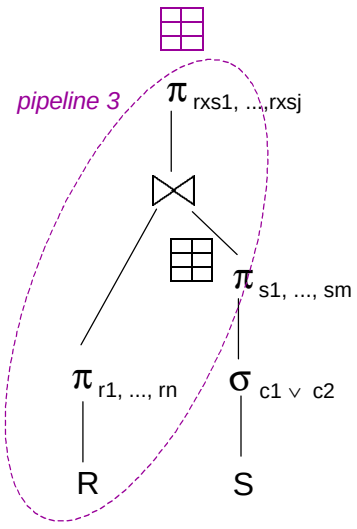
Uso mais Comum de *Pipelines*

- Em uma seqüência de operações que inicia em um nodo folha ou uma operação binária e termina ou no resultado da consulta ou em uma operação binária ob_x, sem incluir ob_x



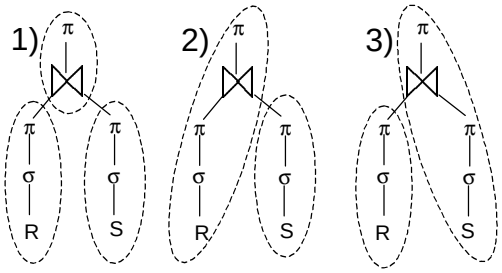
Uso mais Comum de *Pipelines*

- Em uma seqüência composta apenas por operações π e operações produtórias, a partir de um nodo folha ou uma operação binária ob_x, incluindo ob_x
  - considera que o tamanho dos resultados intermediários das operações π são muito grandes para serem materializadas
    - mesmo assim, avaliar se o custo das operações produtórias não aumenta com o *pipeline*...



Exercício 5

a) qual alternativa de *pipeline* para a árvore ao lado possui o menor custo de pior caso?



b) se $b\pi_R = 1$, a resposta do item anterior é diferente?