GBC053–Gerenciamento de Banco de Dados
Processamento de Consultas

Ilmério Reis da Silva
ilmerio@facom.ufu.br
www.facom.ufu.br/~ilmerio/gbd
UFU/FACOM/BCC

ROTEIRO

- *Introdução*
- *Técnicas de Processamento e Métodos de Acesso*
- *Implementação de Operadores da Álgebra Relacional*
- *Otimização de Consultas*

Introdução

PLANO DE CONSULTA

Def.: um plano de consulta é uma ávore de operações da álgebra relacional com a indicação de um algoritmo para cada operação.

Consulta SQL

→

(Álgebra Relacional)

→

Árvore de Consulta

→

Plano de Consulta

→

Estimativa de Custo

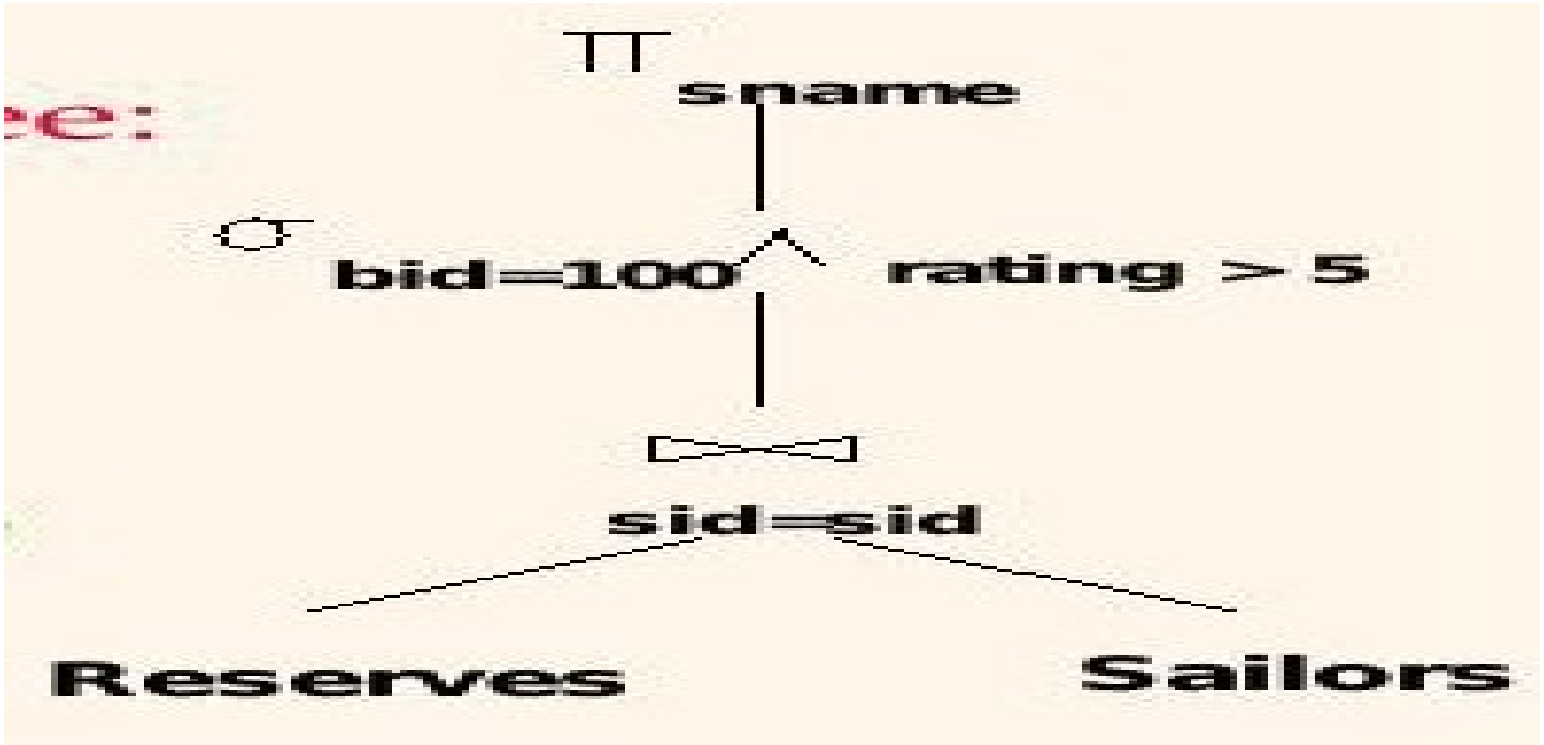
EXEMPLO DE CONSULTA SQL

```
SELECT   S.sname
FROM     Reserves R, Sailors S
WHERE    R.sid=S.sid
AND      R.bid=100 AND S.rating > 5
```

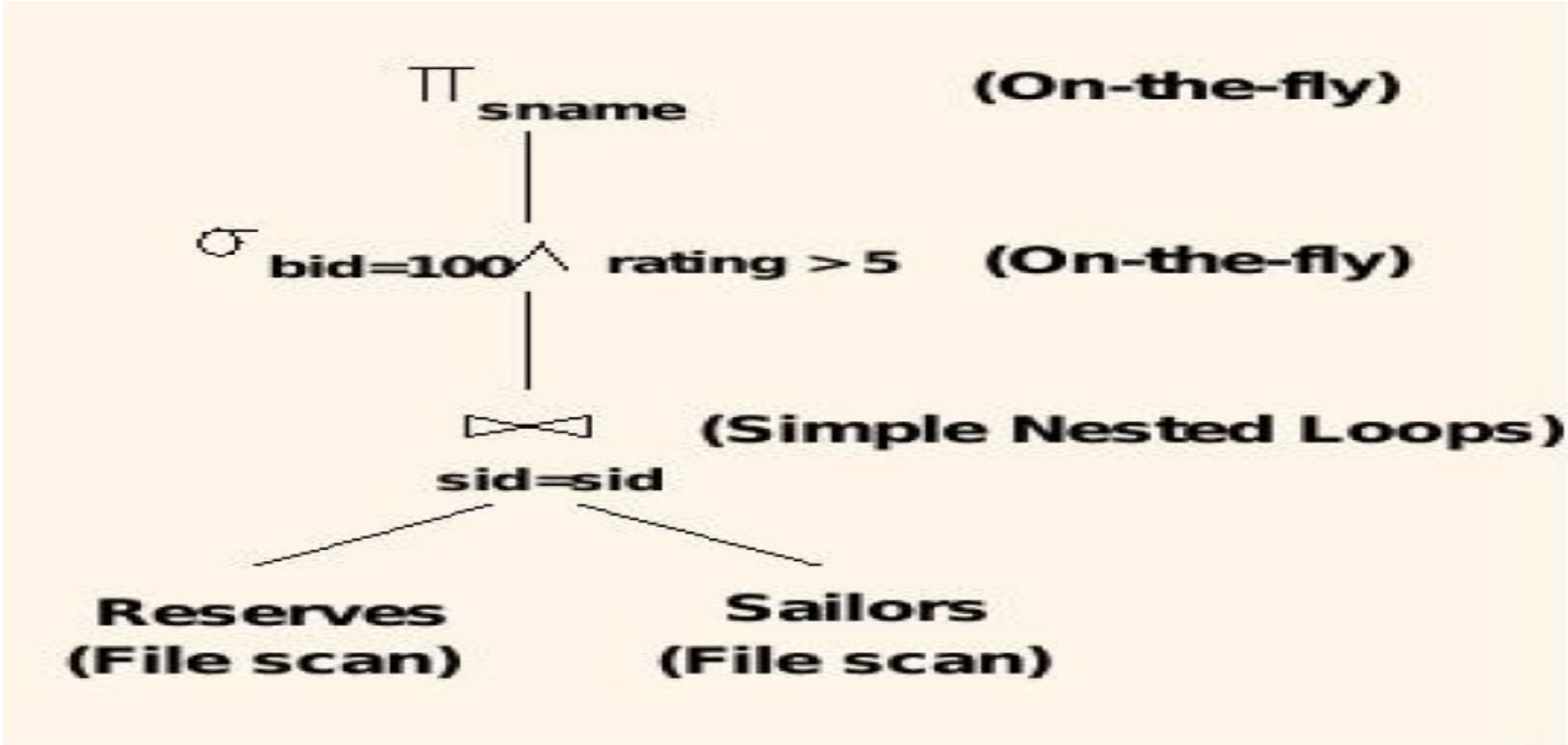
EXEMPLO DE EXPRESSÃO DA ÁLGEBRA RELACIONAL

$$\pi_{sname}\left(\sigma_{bid=100 \wedge rating>5}\left(Reserves \bowtie_{sid=sid} Sailors\right)\right)$$

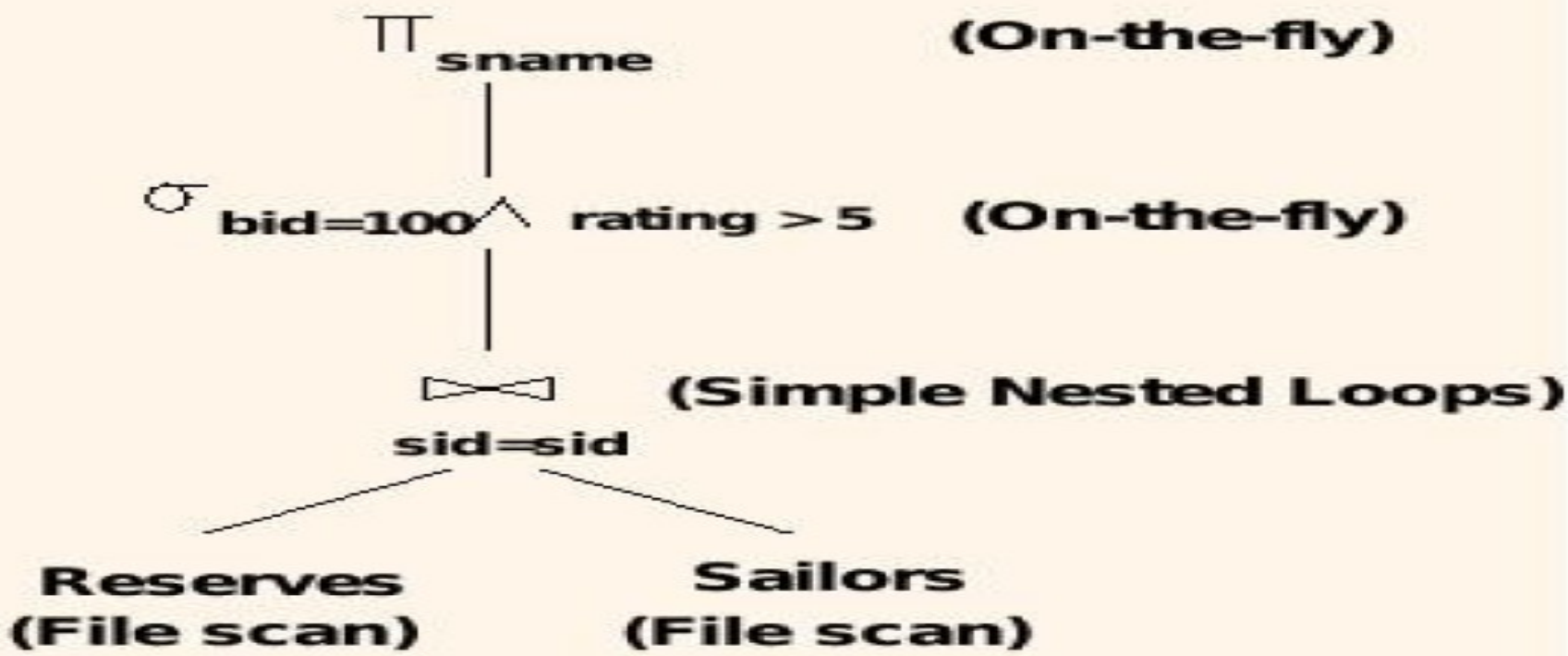
EXEMPLO DE ÁRVORE DE CONSULTA



EXEMPLO DE PLANO DE CONSULTA



CUSTO DE UM PLANO DE CONSULTA



Custo = 1000 + 1000 x 500 = 501.000

OBS: on-the-fly tem custo 0.

Principais problemas

- *Explorar espaço de possíveis planos*
- *Estimar custo de cada plano*
- *Escolher o melhor*

ou

- *Evitar planos ruins*

Relações utilizadas em alguns exemplos

- *Sailors (sid:int, sname:string, rating:int, age:real)*

50 bytes, 40.000 tuplas, 80 tuplas/página, 500 páginas

- *Reserves (sid:int, bid:int, day:date, rname:string)*

40 bytes, 100.000 tuplas, 100 tuplas/página, 1000 páginas

Dados do catálogo

- *Tamanho do bufferpool*
- *Tamanho da página*
- *Dados de tabelas e índices:*
  - Nome
  - Arquivo
  - Organização
  - Atributos
  - Restrições
  - Chaves

Dados do catálogo(cont...)

- *Dados quantitativos, sendo R uma tabela e I um índice*
  - *tuplas(R):* número de tuplas da tabela *R*
  - *pages(R):* número de páginas do arquivo associado a *R*
  - *pages(I):* número de páginas do índice *I*
  - *keys(I):* número de chaves distintas do índice *I*

Dados do catálogo(cont...)

- *Outros dados:*
  - *low(I):* menor valor da chave do índice *I*
  - *high(I):* maior valor da chave do índice *I*
  - Histograma de valores
  - Informações de segurança
  - etc.
- *Dados estatísticos são atualizados periodicamente*

Técnicas de processamento de operadores e métodos de acesso de acordo com condições na cláusula WHERE

Três técnicas comuns

- *Índice: quando a cláusula where recupera pequeno número de tuplas (seleções ou junções)*
- *Iteração: varredura de arquivo examinando sequência de tuplas ou de entradas em um índice*
- *Particionamento: usando hashing ou sorting decompõe operação custosa em similares de menor custo*

OBS: uma dessas técnicas estará presente em cada algoritmo que será estudado.

Escolha do método de acesso

Varredura ou índice? Que tipo de índice?

Inicialmente em condições conjuntivas

- *Árvore: se há uma conjunção de termos presentes no prefixo da chave de busca da árvore*
  - Exemplo: chave=<a,b,c,> e
    - ✓ WHERE a=5 AND b=3
    - ✓ WHERE a=5 AND b>3

- *Hash: se há uma conjunção de igualdades com todos os termos da chave*
  - Exemplo: chave=<a,b,c,> e
    - ✓ WHERE a=5 AND b=3 AND c=5
    - ✓ Não é útil para prefixo ou desigualdade

Condições conjuntivas

- *Aplicar condições dos índices*
- *Aplicar demais condições nas tuplas selecionadas*

Exemplo:

(day < 8/9/2002 AND bid = 5 AND sid = 3),

considere Arvore B+ com chave=<day> e Hash em chave'=<sid>

- ✓ Aplicar Arvore B+ em *day*
- ✓ Aplicar Hash em sid
- ✓ Ordenar resposta por rids e obter interseção
- ✓ Aplicar bid = 5

Condições complexas

Exemplo:

(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3

Converter para Forma normal conjuntiva

(day<8/9/94 OR bid=5 OR sid=3) AND

(rname='Paul' OR bid=5 OR sid=3)

- *Nas disjunções:*

SE houver índice em todas as condições

ENTÃO buscar rids e fazer união

SENÃO varredura;

- *Nas conjunções: fazer intersecção de rids.*

Condições com disjunção Exemplo 1

Não havendo índice em um dos predicados, use varredura(SCAN), verificando todas condições em cada tupla

Exemplo:

(day < 8/9/2002 OR rname ='Joe')

mesmo havendo um hash com chave=<rname>, varre-se a tabela, resolvendo as duas condições

Condições com disjunção Exemplo 2

FNC incluindo índice em um termo da conjunção

Exemplo:

(day < 8/9/2002 OR rname ='Joe') AND sid = 3
com Hash chave=<sid> e Hash chave'=<rname>

- Busca sid no hash
- Varre resultado anterior para resolver
  (day < 8/9/2002 OR rname ='Joe')

Condições com disjunção Exemplo 3

Índice em todos os predicados da disjunção

Exemplo:

(day < 8/9/2002 OR rname ='Joe')
com árvore em chave=<day> e Hash chave'=<rname>

- Busca resultados da árvore *day < 8/9/2002*
- Busca resultados no hash *rname ='Joe'*
- União dos resultados, baseada em *<rid> se alternativa 2 ou 3*

Seletividade das condições de acordo com método de acesso

Def. Seletividade é o número de páginas de dados e de índice necessárias para recuperar todas as tuplas que satisfazem a condição

Def. Fator de redução é a fração de tuplas da tabela que satisfazem a condição

OBS: Assumindo independência entre os predicados de conjunções, os fatores de redução podem ser multiplicados

Exemplo 1: Seletividade das condições de acordo com método de acesso

- *Hash H de Reserves com chave=< rname, bid, sid >*
- *condição: rname ='Joe' AND bid = 5 AND sid = 3*
- *Aproximação para o número de páginas que satisfazem a condição (estimativa da seletividade):*

$$\frac{pages(Reserves)}{keys(H)}$$

Exemplo 2: Fator de Redução das condições de acordo com o método de acesso

- *Índice de Reserves com chave=<bid, sid >*
- *condição: (bid = 5 AND sid = 3)*
- *Conhecendo valores distintos de bid (values(bid)), ou estimativa de 1/10*
- *Conhecendo valores distintos de sid (values(sid)), ou estimativa de 1/10*
- *Estima-se fator de redução pelo produto dos dois, assumindo independência*

SE índice agrupado
ENTÃO esta é a fração de páginas recuperadas
SENÃO cada tupla será uma página recuperada

Exemplo 3: Fator de Redução das condições de acordo com o método de acesso

- *Árvore B+ de Reserves com chave=<day>*
- *condição: (day > 8/9/2008)*
- *Considera-se distribuição uniforme*
- *Estima-se fator de redução :*

$$\frac{High(day) - 8/9/2008}{High(day) - Low(day)}$$

Implementação de Operadores
da Álgebra Relacional

Introdução

- *Operadores individuais:*
  - Unários: seleção, projeção
  - Binários: junção, produto cartesiano, intersecção, união e diferença
  - Funções de agregação
- *Ignorar custos para gravar resultado (pois é comum a todas as soluções)*
- *Notação:*
  - *M e N* são números de páginas das tabelas
  - *B* é o número de páginas do *bufferpool*
  - Usaremos notação *O(F(N)): Custo(N) é O(F(N)) se existem duas constantes C e Y tais que:*

    $\text{Custo}(N) \leq C \cdot F(N)$ para todo $N \geq Y$

SELEÇÃO

```
SELECT *
FROM    Reserve R
WHERE R.rname = 'Joe'
```

$$\sigma_{(R.rname = 'Joe')} Reserves$$

Custo(Scan(R)) = M Ios, neste caso M=1000

Alternativas para SELEÇÃO

- *Se dados ordenados: busca binária*

Custo(Busca(R)) = O(log_2 M)+ paginas_qualificadas

- *Se Árvore B+:*

SE AGRUPADO

Custo = Custo(busca) + paginas_qualificadas

SENAO

Custo = Custo(busca) + tuplas_qualificadas

- *Considerações sobre índices não agrupados*
  - *ordenação de entradas qualificadas por rid.page evita um IO por tupla*
  - *Uso somente se fator de reduçao < 5%*

SELEÇÃO - Exemplo

*SELECT **

FROM Reserve R

WHERE R.rname < 'C%'

$$\sigma_{rname<'C\%'}(Reserves)$$

R tem 100.000 tuplas e 100 tuplas/página, logo M=1000

considerando fator de redução de 10% a seletividade=100 páginas

serão 10.000 tuplas qualificadas

Custo(busca): percorrer a árvore para encontrar início

- *Custo(Arvore B+ agrupada)=Custo(busca) + 100 IOs*
- *Custo(Arvore B+ nao agrupada)=Custo(busca)+10.000IOs*
- *Custo(SCAN)= 1000 Ios*

SELEÇÃO usando Hash

```
SELECT *
FROM    Reserve R
WHERE R.rname = 'Joe'
```

$$\sigma_{(R.rname = 'Joe')} Reserves$$

- *Um ou dois IOs para chegar ao bucket*
- *Recuperar tuplas qualificadas*
  - Supondo 100 reservas para Joe, serão mais 100 IOs, se alternativas 2 ou 3

PROJEÇÃO

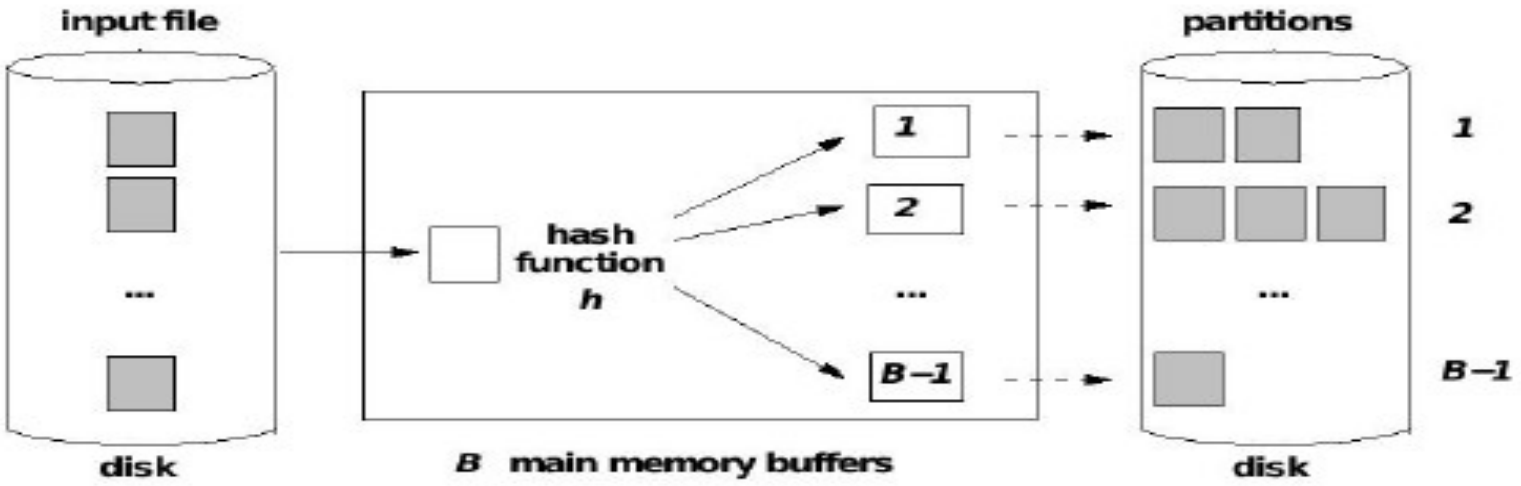
SELECT DISTINCT sid, bid

FROM Reserves R

- *Problemas*
  - Eliminar atributos
  - *Se DISTINCT* eliminar tuplas duplicadas
- *Alternativas*
  - Pode-se usar processamento *index-only* caso a chave contenha todos os atributos
  - Índice pode ser usado para eliminar tuplas duplicadas
  - Se não há índice apropriado, use ordenação*(sorting)* ou particionamento por meio de *hashing*

PROJEÇÃO - Particionamento por Hash

- *Fase 1: Particionamento*
  - Ler Reserves descartando atributos não desejados
  - Aplique uma função hash h_1, distribuindo tuplas em B-1 buckets, com base nos atributos da projeção
  - Resultado são B-1 partições com tuplas iguais na mesma partição. Cada partição deve caber no buffer para Fase 2



PROJEÇÃO - Particionamento por Hash

- *Fase 2: Eliminação de tuplas duplicadas*
  - para cada partição
    - ✓aplique nova função hash h_2 diferente de h_1
    - ✓compare tuplas com mesmo valor de h_2
    - ✓descarte as tuplas iguais
    - ✓grave partição sem duplicadas

- *A partição deve caber na memória RAM*

PROJEÇÃO - Particionamento por Hash Análise

- *Considere distribuição uniforme entre partições*
- *Seja T o nro de páginas antes da eliminação das duplicadas*
- *Então*
  - *T/(B-1)* é o número de páginas de cada partição
  - $f \times T/(B-1)$ é o número de páginas considerando f como fator de correção para o *hash*
  - Para evitar *overflow* $B > f \times T/(B-1)$
  - Logo $B > sqrt(f \times T)$
- *Custo*
  - Particionamento: le M e grava T, logo, M + T
  - Eliminação: le T
  - Custo = M + 2 × T IOs

PROJEÇÃO - Hash x Sorting

- *Distribuição não uniforme prejudica Hash*
- *Além disso, saída do sorting já é ordenada*
- *Se B > sqrt(T) então sorting terá :*
  - passo 0 lendo M páginas e gravando T (M + T) IO, e
  - passo 1, lendo T e gravando o resultado
  - ignorando esta última gravação: (M + 2T) IO igual ao Hash

- *A ordenação é o padrão para eliminação de duplicatas em projeção, pois:*
  - Facilita quando há necessidade de saída ordenada (order by);
  - É melhor quando não houver distribuição uniforme das “tuplas”;
  - O algoritmo é mais simples e de menor custo de CPU;

- *OBS: ambos os algoritmos, Hash e Sorting, são implementado na maioria dos gerenciadores comerciais*

JUNÇÃO

- Operação custosa e comum
- Exemplo:

$$Reserves \bowtie_{sid} Sailors$$

Principais algoritmos:

- Laços Aninhados Simples, Paginados e Blocados
- Laços Aninhados Indexado
- *Sort Merge Join e Hash Join*

Considerações:

- R com M páginas com p_r tuplas por pagina
- S com N páginas e p_s tuplas por página.

Laços Aninhados Simples

para cada $r \in R$

para cada $s \in S$

SE $r_i = s_j$ *adicione* $< r, s >$ *à resposta*

- Custo = $M + p_r \times M \times N\ Ios$

- Exemplo : $Reserves \bowtie_{sid} Sailors$

Custo = 1000 +100×1000×500

= 50.001.000 IOs (+/- 140 horas ou 6 dias)

Laços Aninhados Paginados

para cada Rpage

para cada Spage

para cada r ϵ Rpage

para cada s ϵ Spage

SE $r_i = s_j$ adicione < r, s > à resposta

- Custo = M + M× N IOs
- Exemplo ·

$$Reserves \bowtie_{sid} Sailors$$

Custo = 1000 + 1000 × 500

= 501.000 IOs (+/- 1h30m)

Laços Aninhados Blocados

Duas situações:

- A menor relação cabe em B-2 páginas do buffer
- A menor relação não cabe no buffer

Laços Aninhados Blocados (R < B-2)

A menor relação, por exemplo R, cabe em B-2 páginas do buffer

ler R para o buffer

para cada Spage

para cada r ϵ R

para cada s ϵ Spage

SE $r_i = s_j$ adicione < r, s > à resposta

Custo = M + N IOs

$$Reserves \bowtie_{sid} Sailors$$

Custo = 1000 + 500 IOs (+/- 15 segundos)

Laços Aninhados Blocados (R > B-2)

A menor relação, por exemplo R, nao cabe no buffer

divida R em partições com B-2 páginas

para cada Rblock com B-2 páginas

para cada Spage

para cada r ϵ Rblock

para cada s ϵ Spage

SE $r_i = s_j$ adicione < r, s > à resposta

Custo = M + N x M/(B-2) IOs

$Reserves \bowtie_{sid} Sailors$

Sendo B=102,Custo = 500 + 1000 500/100=5500 IOs (+/- 55s)

Considerações sobre Laços Aninhados Blocados

Outras Considerações sobre Laços Blocados

- Um *hash* para R no *buffer* facilita a avaliação do predicado de igualdade $r_i = s_j$
- O uso de bloco diminui o custo de IO
- Pode-se usar *double buffering*, aproveitando melhor a CPU durante IO dos blocos

Laços Aninhados Indexado

para cada tupla $r \in R$

faça busca no índice de S usando cada chave r_j *e*

se encontrar $s_j = r_i$ *adicione* $< r, s >$ *à resposta*

$Custo = M + p_r \times M \times Custo(busca)$

Custo(busca) : Arvore B+: 2 a 4 Ios e Hash 1 a 2 Ios

Se alternativa diferente de 1, acrescentar um IO

Exemplos com alternativa 2:

- *Hash em Sailors e 20% de overflow e PK(sailors)=sid*

  $Custo = 1000 + 1000 \times 100 \times (1,2 + 1) = 221.000$ *Ios*

- *Hash em Reserves (agrupado e não agrupado)*

  $Custo = 500 + 500 \times 80 \times (1,2 + 1) = 88.500$ *Ios*

  $Custo = 500 + 500 \times 80 \times (1,2 + 2,5) = 128.500$ *Ios*

Sort Merge Join

```
proc smjoin(R, S,Ri, Sj)
if not sorted(R,Ri) then sort(R,Ri);
if not sorted(S, Sj) then sort(S, Sj);
r = first(R); s = first(S);
g = s;                                  % grupo(partição) corrente de S
while (r <> eof) AND (g <> eof) {
        while (r.i < g.j) r = next(R);  % percorrendo R;
        while (r.i > g.j) g = next(S);  % percorrendo S;
        s = g;                          % necessário se r.i <> g.j
        while (r.i == g.j) {
                s = g;                  % retorna busca na partição g de S
                while (r.i == s.j) {
                        adicione < r, s > à resposta;
                        s = next(S);    % percorrendo S;
                }
                r= next(R);             % percorrendo R;
        }
        g= s;                           % próxima partição S;
}
```

Custo do Sort Merge Join

- *Custo = Custo(sorting) + Custo(merging)*
- *Custo(sorting)= O(MlogM) + O(NlogN)*
- *Custo(merge(R,S,Ri,Sj)= M + N caso não existam repetições em S*
- *Exemplo Reserves x Sailors*

$$Custo(sortmergejoin) = 2 \times 2 \times 1000 + 2 \times 2 \times 500 + 1000 + 500 = 7500 \text{ Ios.}$$

- *Pior caso: leitura de S para cada tupla de R, quando todos os atributos são iguais (produto cartesiano). Custo = M × N*
- *Comum: relacionamento por chave estrangeira, com relações previamente ordenadas, portanto, uma varredura por relação*

Melhoria no Sort Merge Join

- *Considerando:*
  - *duas relaçõe não ordenadas*
  - *Passo 0 do Merge Sort Externo para R e S;*
  - *merge de todos os subarquivos-ordenados*
  - *O buffer deve ser suficiente para conter uma página de cada subarquivo-ordenado*
- *Custo = 3 × (M + N) IOs*

  % Passo 0: leitura/gravação.

  % Merge: leitura.
- *Seja L tamanho da maior relação*

  número de subarquivos-ordenados = teto(L/B)

  então B > L/B, logo B > 2 × sqtr(L)
- *SE Replacement sort, o Passo 0 gera subarquivos de L/(2×B) então basta que B > sqtr(L). No exemplo: Custo = 4.500 IOs*

Hash Join

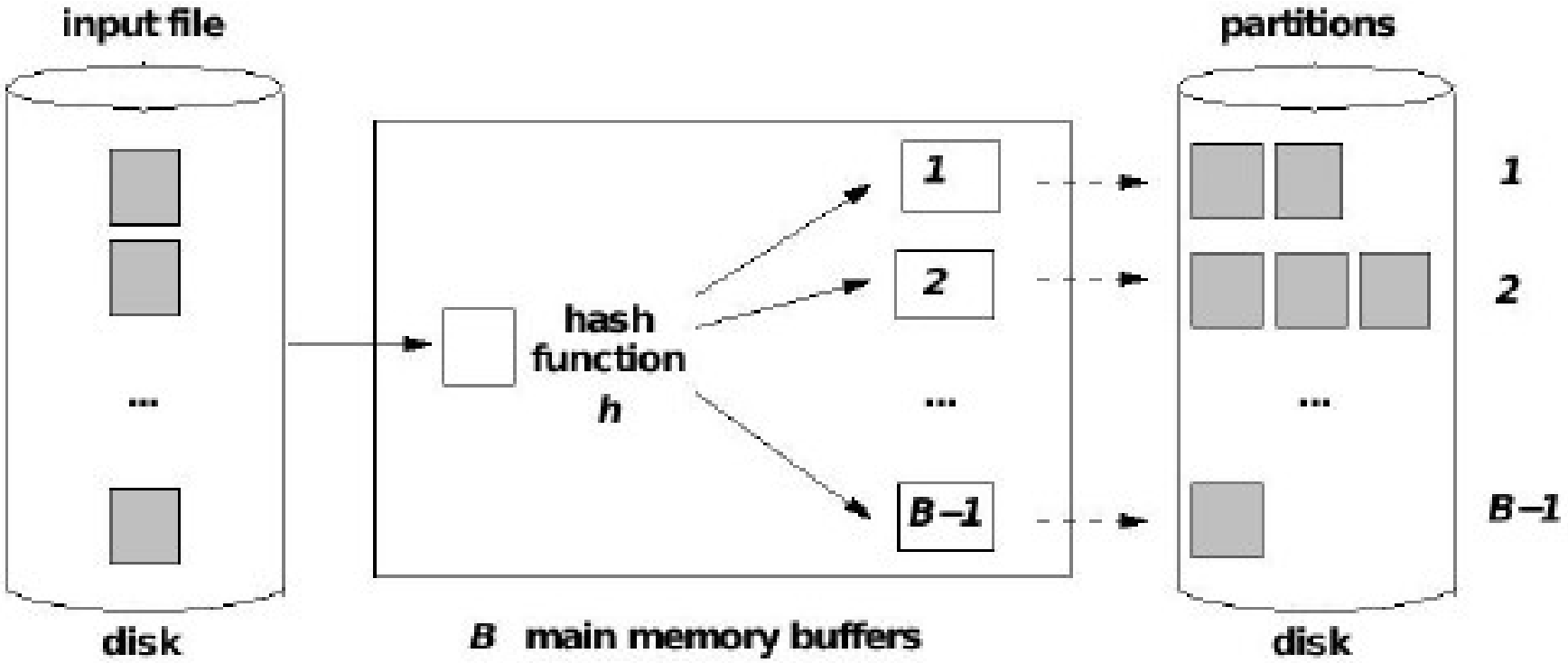
Duas fases

Particionamento – idêntico à projeção, gerando partições de cada tabela R e S com base em uma mesma função hash h_1

Probing – compara partições R_x *e* S_x *usando uma segunda função hash* h_2

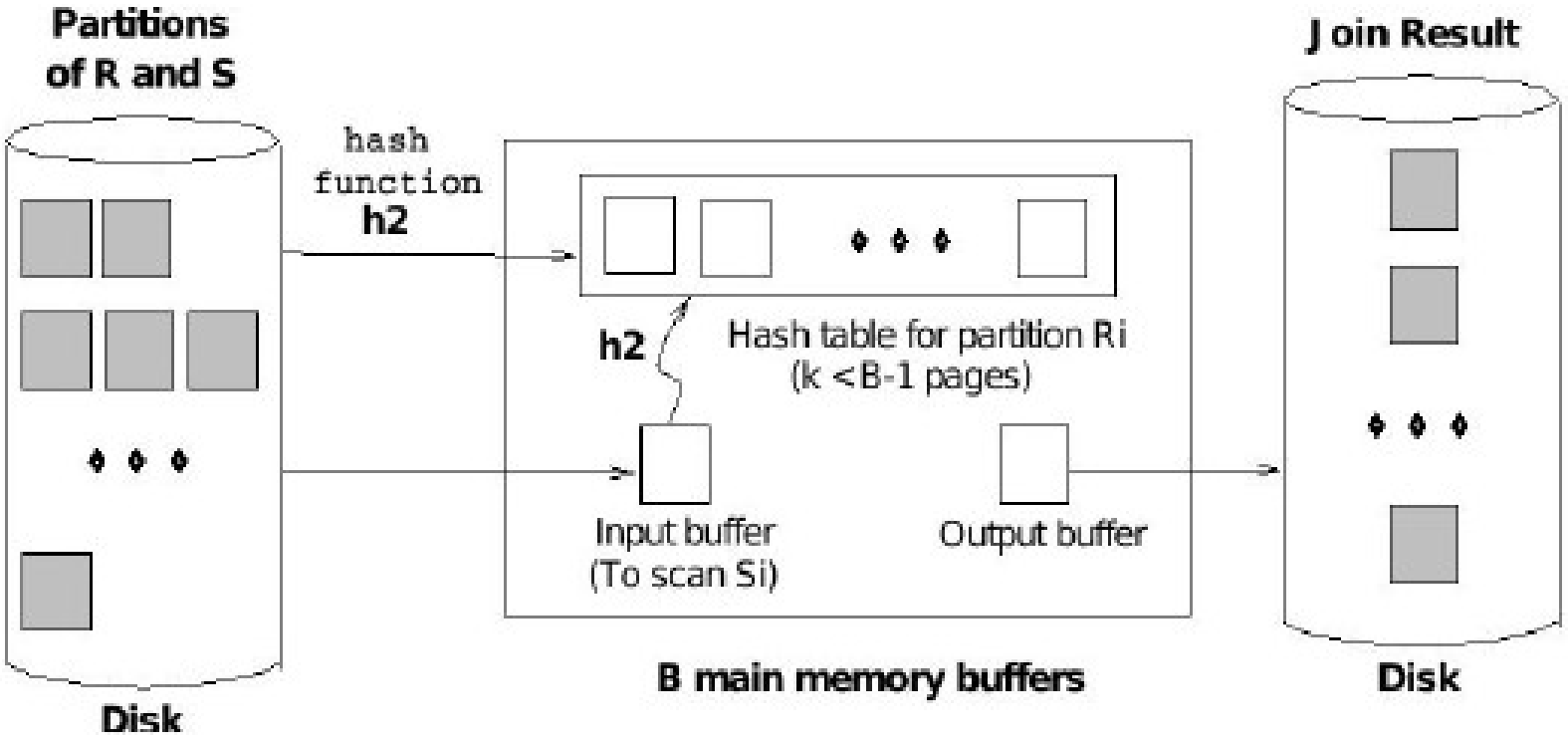
Hash Join

Fase 1 : Particionamento – gera partições de cada tabela R e S com base em uma mesma função hash h_1



Hash Join

Fase 2 : Probing – compara partições R_x e S_x usando uma segunda função hash h_2



Algoritmo Hash Join

```
% Partition
foreach r ϵ R add r to buffer h(r.i); % flush quando necessário
foreach s ϵ S add s to buffer h(s.j); % flush quando necessário
% Probing
for l = 1 · · · k {            % percorre partições de R e de S
    foreach r ϵ R_l add r to page h_2(r.i);   % novo hash
    foreach s ϵ S_l {                        % percorre partição S_l
        compute h_2(s.i);
        forall (r ϵ R_l | r.j == s.i)        % verifica matchings em R_l
                output <r, s>
    }
    clear hash table;
}
```

Custo e Requisitos de memória do Hash Join

Custo

- *Custo = Custo(particionamento) + Custo(probing)*
- *Custo(particionamento) = 2 × (M + N)*
- *Custo(probing) = M + N*
- *Custo = 3(M + N)*

Requisitos de memória

- *maior número de partições é B − 1*
- *tamanho médio de cada partição M/(B−1)*
- *páginas p/ hash f × M/(B−1) | f é um fator de correção*
- *segunda fase B > f × M/(B−1) + 2, (hash+inp+out)*
- *aproximação B > sqrt(f × M), onde M é o número de páginas da menor tabela*

OBS: perda de desempenho se distribuição não uniforme

Hash Join Híbrido

Supondo que $B > f \times M/k$, k inteiro $< B-1$

- *R é dividida em k partições de tamanho M/k*
- *Constroí-se um hash para cada partição (R e S)*
- *Particiona-se usando k buffers de output e um de input*
- *Sobram B -(k+1) páginas no buffer, que pode caber uma partição de R, por exemplo R_1, que será permanente na memória e cuja verificação com S_1 será feita na fase de particionamento.*
- *As demais partições são processadas normalmente.*

Hash Join Híbrido – Exemplo de custo

Exemplo: M=500, N=1000, B=300 k = 2

R_1 = 250 pg é lida e mantida na memória, Custo(R_1)=250 IOs

R_2 = 250 pg é lida e gravada em disco, Custo(Fase1,R_2) = 500 IOs

S_1 = 500 pg é lida e comparada com R_1 Custo(S_1) = 500 IOs

S_2 = 500 pg é lida e gravada, Custo(Fase1, S_2) = 1000 IOs

Na segunda fase somente R_2 e S_2 são lidas,

Custo(Fase 2) = 250 + 500 = 750

Custo Total = 250 + 500 + 500 + 1000 + 750 = 3000 Ios

OBS: se uma das duas cabem na memória, a outra também poderá ser lida somente uma vez. Custo = 1500 Ios (ótimo)

Comparando Hash × Laços Blocados

Se uma relação cabe no buffer,

ambos são ótimos (M+N)

Caso contrário,

laços blocadas lerá várias vezes uma das relações,

logo, Hash Join é melhor.

Comparando Hash × Merge Join

Seja M o tamanho da menor relação

Se B > sqrt(M)

Custo(Hash Join) = 3 × (M + N)

Logo usa-se Sort Merge Join se:

- *Partições não são de tamanho uniforme*
- *Saída deve ser ordenada*
- *B < sqrt(M)*
- *Ou SE B > sqtr(L) > sqtr(M), sendo L o tamanho da maior relação. Usando replacement sort teremos o custo do sort merge join = 3 × (M + N)*

Junção com condições genéricas

Conjunção de várias igualdades

Exemplo: R.sid = S.sid AND R.rname = S.sname

- *Laços Aninhados Indexados*
  Construir índice com chave =< S.sid, S.sname > e usar como tabela "inner"

- *Sort Merge (ou Hash)*
  Sort em R e S pela chave <sid, name>

Junção com condições genéricas

Desigualdades

Exemplo: (R.rname < S.sname)

Laços Aninhados Indexados usando Arvore B+

Varrer índice na tabela inner

Haverá maior número de matchings

Laços Aninhados com blocos pode ser melhor estratégia

Sort e Hash não se aplicam

Operações com conjuntos

Intersecção e produto cartesiano, $R \cap S$ e $R \times S$, são casos especiais da junção

Em União, $R \cup S$, o principal problema é a eliminação de duplicidades da cláusula DISTINCT, idêntico a projeção

Diferença, $R - S$, o problema é similar à união e projeção, remover duplicidades

União baseada em sorting

sort R e S usando tupla como chave

varrer R e S eliminando duplicadas

Alternativa:

Passo 0 para R e S

merge subarquivos de R e S, eliminando duplicadas

União baseada em hashing

Fase 1 - Particionamento: usando h_1 idêntico à projeção

Fase 2 - para cada partição R_l comparar com S_l

construir hash em memória para R_l usando $h_2 <> h_1$,

varrer S_l, calcular h_2 e

adicionar não repetidas à resposta

•

Diferença baseada em hashing

Semelhante à União

Fase 1 - Particionamento: usando h_1 idêntico à projeção

Fase 2 - para cada partição R_l comparar com S_l

construir hash em memória para R_l com $h_2 <> h_1$

varrer S_l,

calcular h_2

remover tuplas repetidas do hash de R_l

adicionar tuplas restantes em R_l à resposta

Funções de agregação

Sem Group By :

- *varrer tabela acumulando valores em memória ou*
- *contar entradas no índice se houver*

Com Group By:

- *Sorting com chave = atributos do Group By*
- *varrer com acumuladores por partição*

Custo = O(MlogM)

Agregação index-only

Índice não é usado para selecionar subconjuntos, mas para evitar acesso ao arquivo quando inclui os atributos da agregação

Em Group By, se atributos são prefixo do índice, evita-se sorting

Hashing somente se chave = atributos do Group By

Dados do tipo: sum(chave); count(chave);

ou count(chave=valor)

Impactos do Bufferpool

- *Operações concorrentes diminuem disponibilidade*
  - *Política de substituição depende de padrões de acesso*
  - *Exemplos em operações de Junção:*
    - *Laços Aninhada Simples: Em LRU, há Sequential Flooding; Em MRU, (B − 2) pgs da tabela inner ficam no buffer*
    - *Laços Aninhada Blocados: para cada bloco da tabela outer lê-se toda a tabela inner não há diferença entre as políticas*
    - *Laços Aninhada Indexados: para cada tupla da tabela outer lê-se, via índice, tuplas da tabela inner. Se várias tuplas da tabela outer tem mesmo valor, sua ordenação otimiza uso do buffer*

Sobre os operadores isolados

- *Um fator positivo é que o conjunto de operações do modelo relacional é formado por poucas operações*
- *Entretanto são várias alternativas e não há uma melhor em todos os casos*
- *Estatísticas do BD, armazenadas no catálogo, auxiliam a escolha de uma boa estratégia*
- *Otimização inclui esta escolha e a combinação de operações, que será nosso próximo assunto*

Tabela de custos

Otimização de Consultas

Otimização de Consultas

Fundamentos:

- *Comandos SQL têm várias formas de implementação*
- *Plano de Consulta: árvore de operadores (seleção, projeção e junção) incluindo algoritmos de implementação*
- *Principais problemas:*
  - ✓ espaço de planos alternativos
  - ✓ estimativa de custo
  - ✓ encontrar o de menor custo ou encontrar uma boa estratégia

Otimização de Consultas

Relembrando as relações exemplo

Sailors (sid:int, sname:string, rating:int, age:real)

50 bytes, 40.000 tuplas, 80 tuplas/pagina, 500 páginas

Reserves (sid:int, bid:int, day:date, rname:string)

40 bytes, 100.000 tuplas, 100 tuplas/página, 1000 páginas

Boats (bid:int, bname:string, color:string)

Otimização de Consultas

Exemplo SQL e expressão da Álgebra Relacional:

```
SELECT   S.sname
FROM     Reserves R, Sailors S
WHERE    R.sid=S.sid
AND      R.bid=100 AND S.rating > 5
```

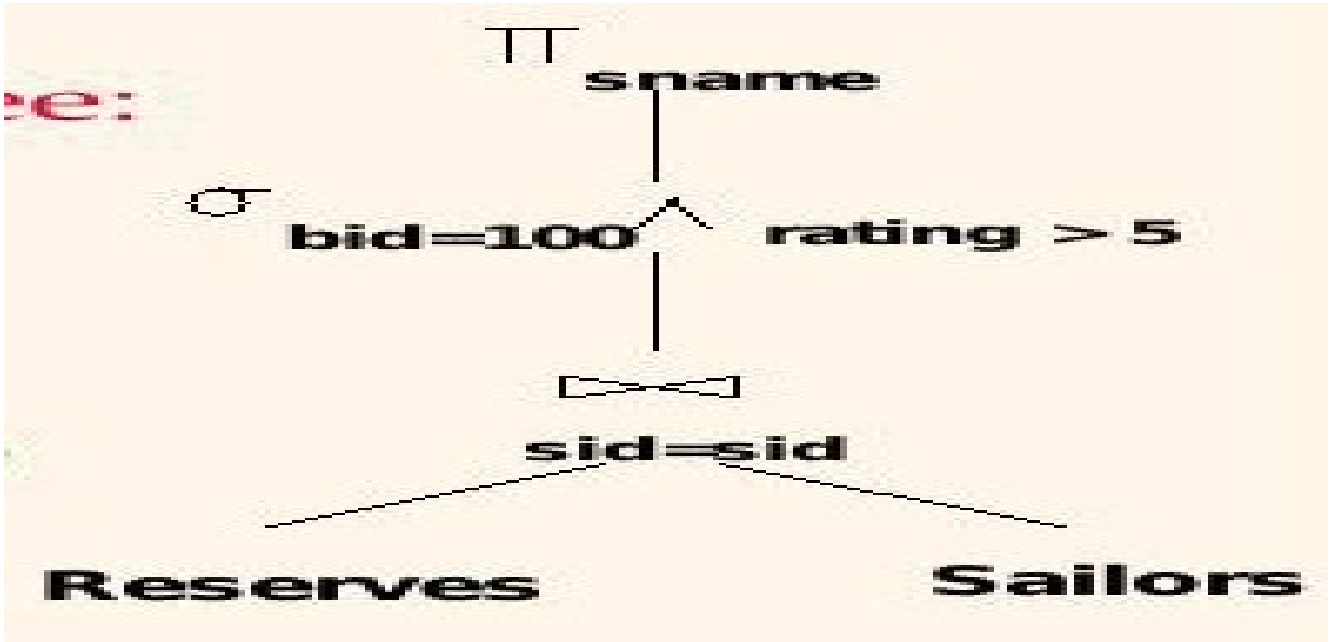
Expressão da álgebra:

$$\pi_{sname}\left(\sigma_{bid=100 \wedge rating>5}\left(Reserves \bowtie_{sid=sid} Sailors\right)\right)$$

Otimização de Consultas

Exemplo Expressão da Álgebra e Árvore de Consulta:

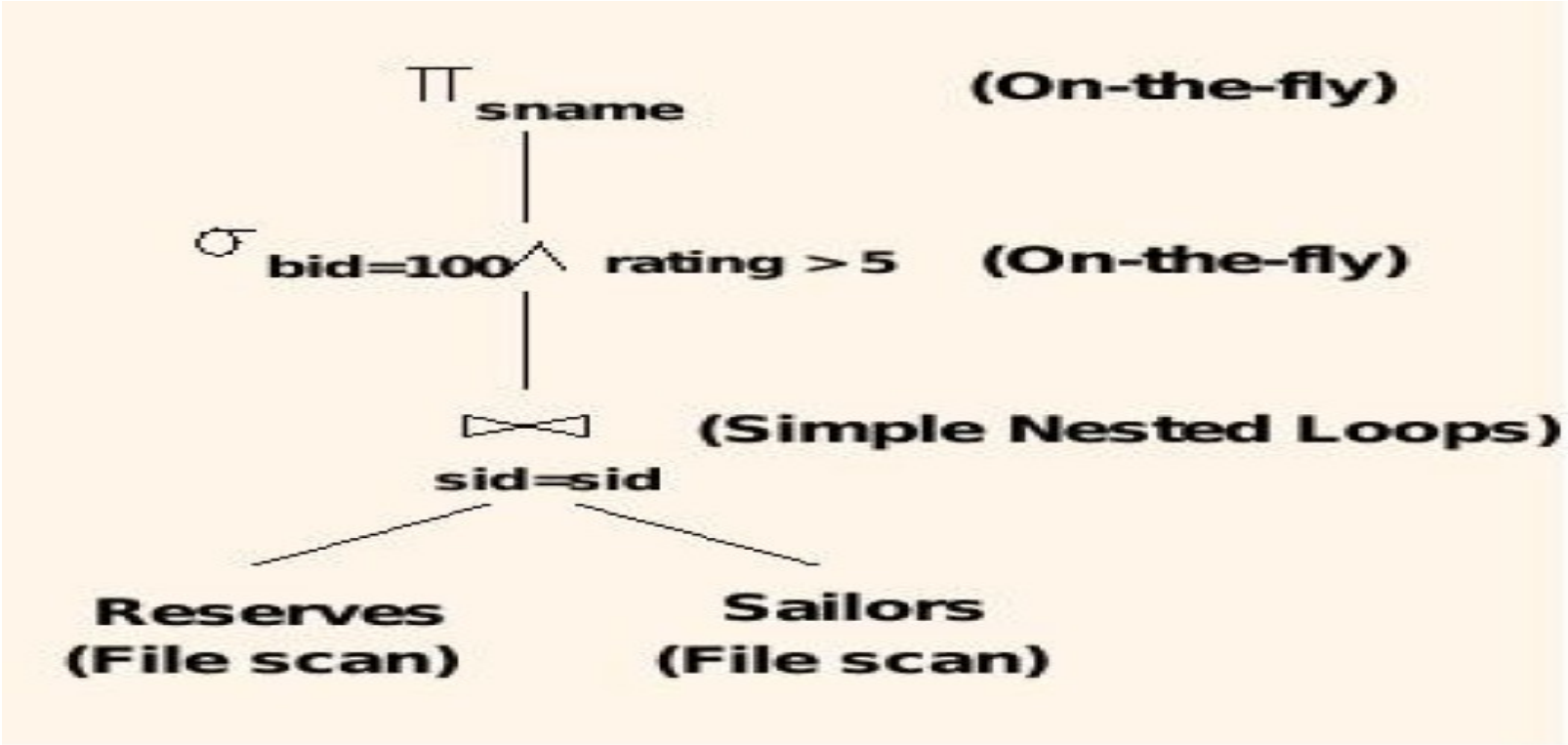
$$\pi_{sname}(\sigma_{bid=100 \wedge rating>5}(Reserves \bowtie_{sid=sid} Sailors))$$



Otimização de Consultas

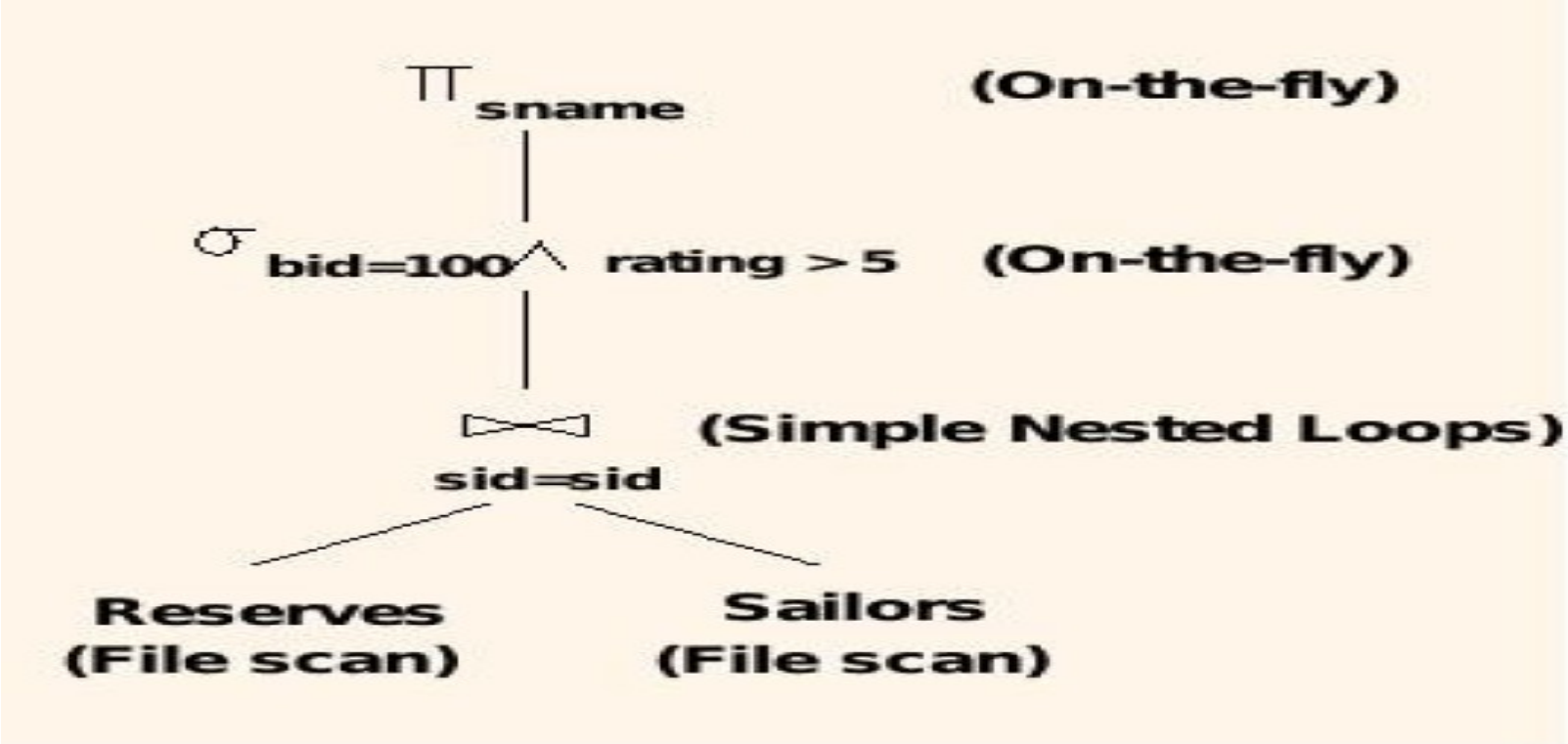
Exemplo Expressão da Álgebra e Plano de Consulta:

$$\pi_{sname}\left(\sigma_{bid=100 \wedge rating>5}\left(Reserves \bowtie_{sid=sid} Sailors\right)\right)$$



Custos - Otimização de Consultas

Custo Plano de Consulta:

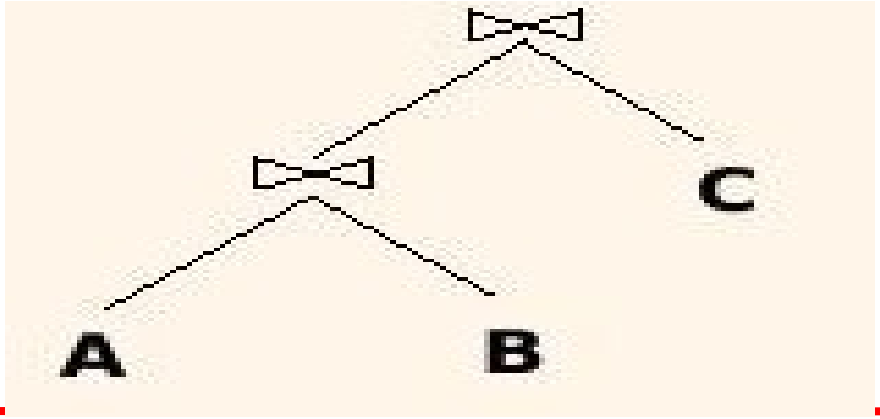


Custo = 1000 + 1000 x 500 = 501.000

OBS: on-the-fly tem custo 0.

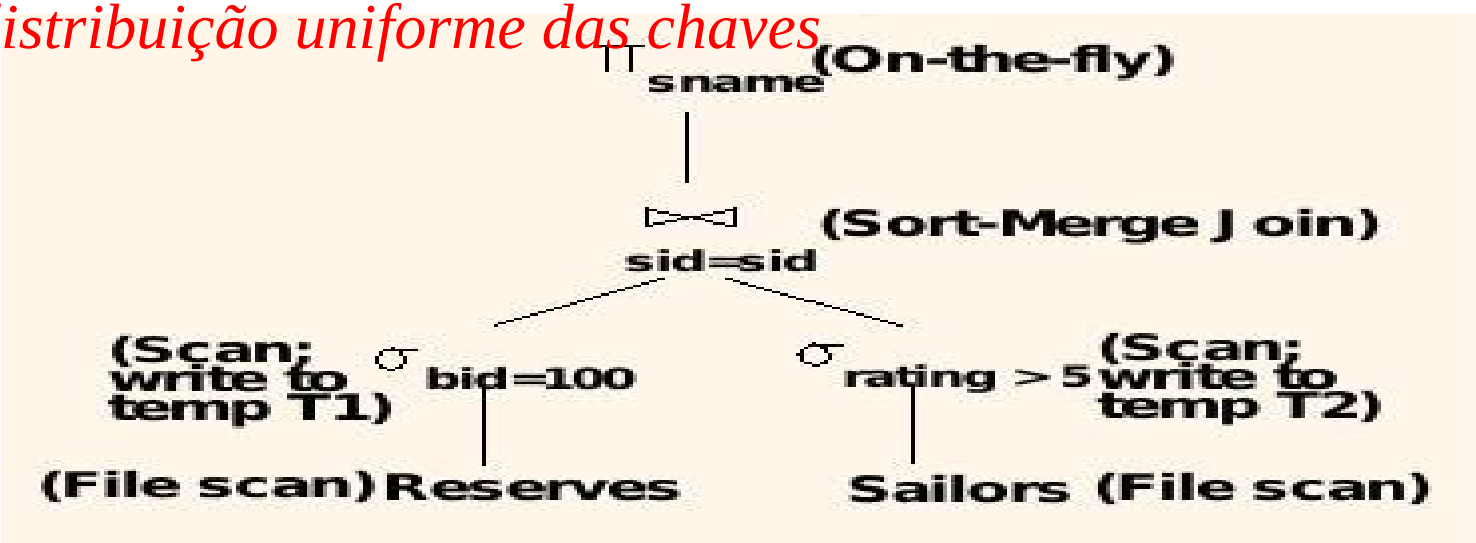
Pipeline x Materialização

- *divisão de condições em grupos, priorizando índices;*
- *materializacao grava resultado de primeiras condições e aplica outras no resultado*
- *pipeline aplica segundo grupo de condições nas tuplas que passaram pelo primeiro, evitando gravação de tabelas intermediárias, também chamado (on-the-fly)*
- *Exemplo de pipeline: com junção "Laços Aninhados" busca-se tuplas em C para cada tupla da junção de A com B*



Plano Alternativo

- *Custo Plano de Consulta Antecipando seleção, assumindo distribuição uniforme das chaves*



- *Selecao(bid=100) = 1000 + 10, scan/gravação T1(1%)*
- *Selecao(rating>5)= 500 + 250, scan/gravação T2(50%)*
- *Sorting(T1) = 2 x 2 x 10 = 40 (2 passos, B=5)*
- *Sorting(T2) = 2 x 4 x 250 = 2000 (4 passos)*
- *MergeJoin(T1, T2) = 10 + 250, scan nas duas*
- *Total = 1010 + 750 + 40 + 2000 + 260 = 4060 IOs*

Plano Alternativo

- *Custo Plano de Consulta Antecipando seleção, usando Laços Blocados ao inves de Sort-Merge*

para cada bloco de 3 pgs de T1

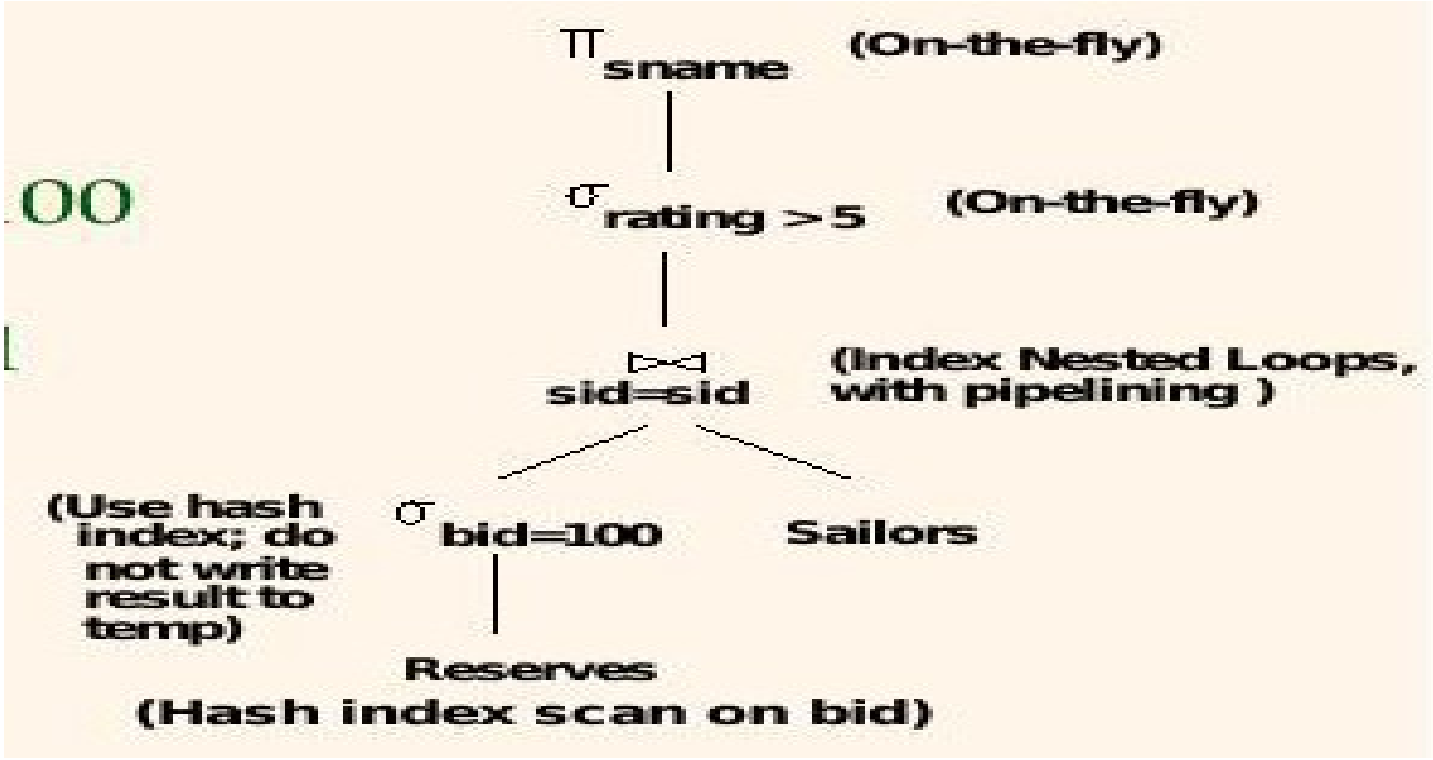
varrer T2

- *Selecao(bid=100) = 1000 + 10, scan/gravação T1(1%)*
- *Selecao(rating>5)= 500 + 250, scan/gravação T2(50%)*
- *JunçãoBlocada(T1, T2) = 10 + 4 x 250*
- *Total = 1010 + 750 + 1010 = 2770 Ios*
- *Se anteciparmos projeção T1'=sid e T2'=(sid, sname), T1' caberá no buffer: CustoJunção<250*
- *Logo Custo = 2010 IOs*

Plano Alternativo usando índice

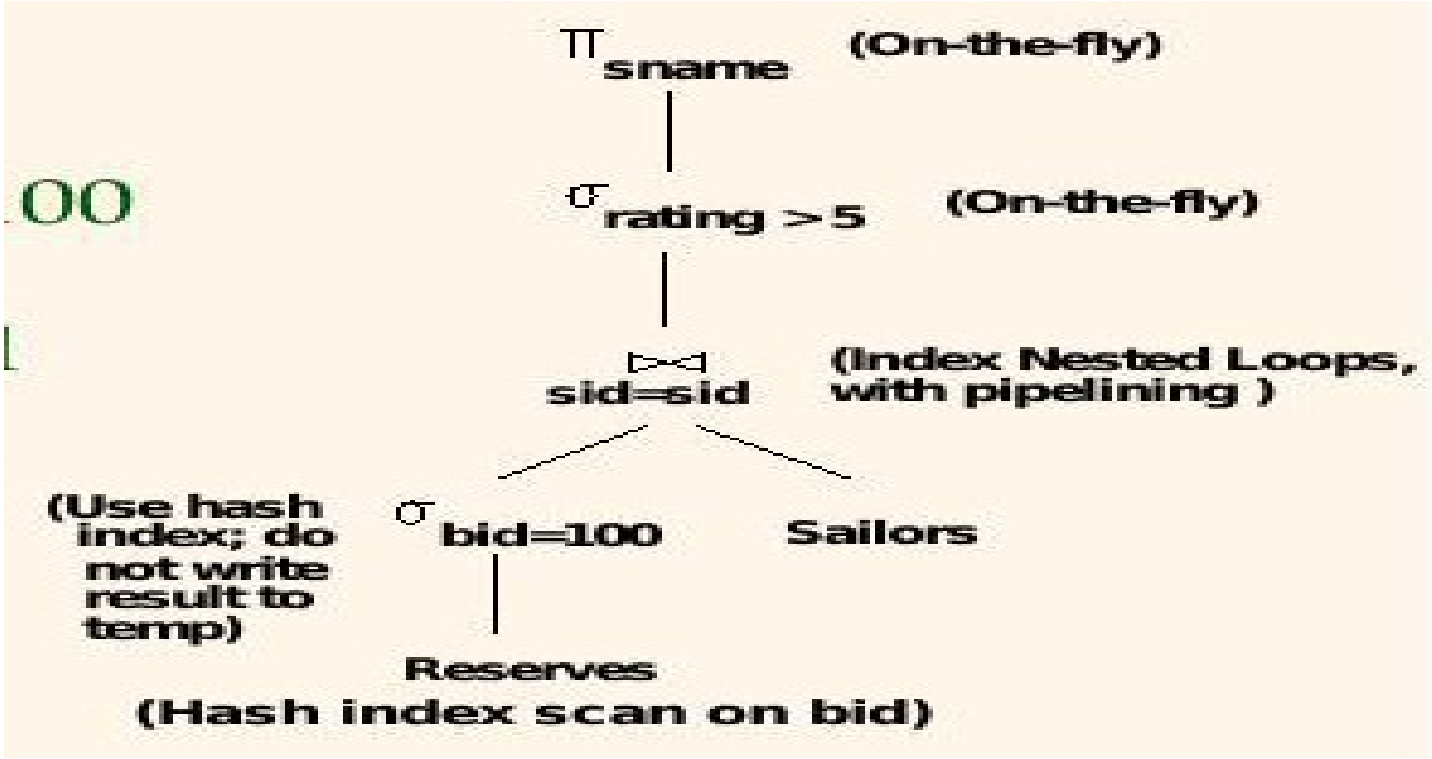
- *Exemplo Expressão da Álgebra e Plano de Consulta usando índices: Hash(Reserves.bid), Hash(Sailors.sid)*

$$\pi_{sname}\left(\sigma_{bid=100 \wedge rating>5}\left(Reserves \bowtie_{sid=sid} Sailors\right)\right)$$



Plano Alternativo usando índice

- *Custo:*



Custo(bid=100) = 10, todos em um bucket, junção com pipeline, Custo(junção) = 1,2 x 1000

Total = 10 + 1200 = 1210 IOs

Plano Alternativo usando índice e sorting

Custo supondo MergeSort após seleção, supondo índice agrupado em Sailors

- *Custo(bid=100) = 10 + 10, materializa resultado*
- *Custro(sorting) = 2 x 2 x 10 = 40*
- *Agora varre-se Sailors e localiza correspondente em Reserves com apenas um acesso em ambas, portanto:*
- *Custo aproximado = 20 + 40 + 500 + 10 = 570 Ios*

- *Logo, pipeline nem sempre é a melhor alternativa*

Plano Alternativo – Junção trivial

SE todas as tuplas da tabela outer tiverem somente uma tupla correspondente na tabela inner

SELECT S.sname

FROM Reserves R, Sailors S

WHERE R.sid=S.sid

AND R.bid=100 AND S.rating > 5

AND R.day='8/9/2002'

Índice em R.bid => Custo(R.bid=100) = 10

R.day on-the-fly => Custo(R.day='8/9/2002') = 0

Índice em S.sid => Custo(busca em sailors) = 1

S.rating on-the-fly => Custo(S.rating > 5) = 0

- *Custo total = 11 Ios*

  OBS: supondo chaves: <bid,day> em R e <sid> em S

Otimização de Consultas

Objetivo:

encontrar um bom plano de execução para uma consulta

Como?:

- *enumerar um conjunto(não necessariamente exaustivo) de planos*
- *usar o catálogo para estimar tamanhos e custos*
- *escolher o plano de menor custo*

Otimização de Consultas

Como?

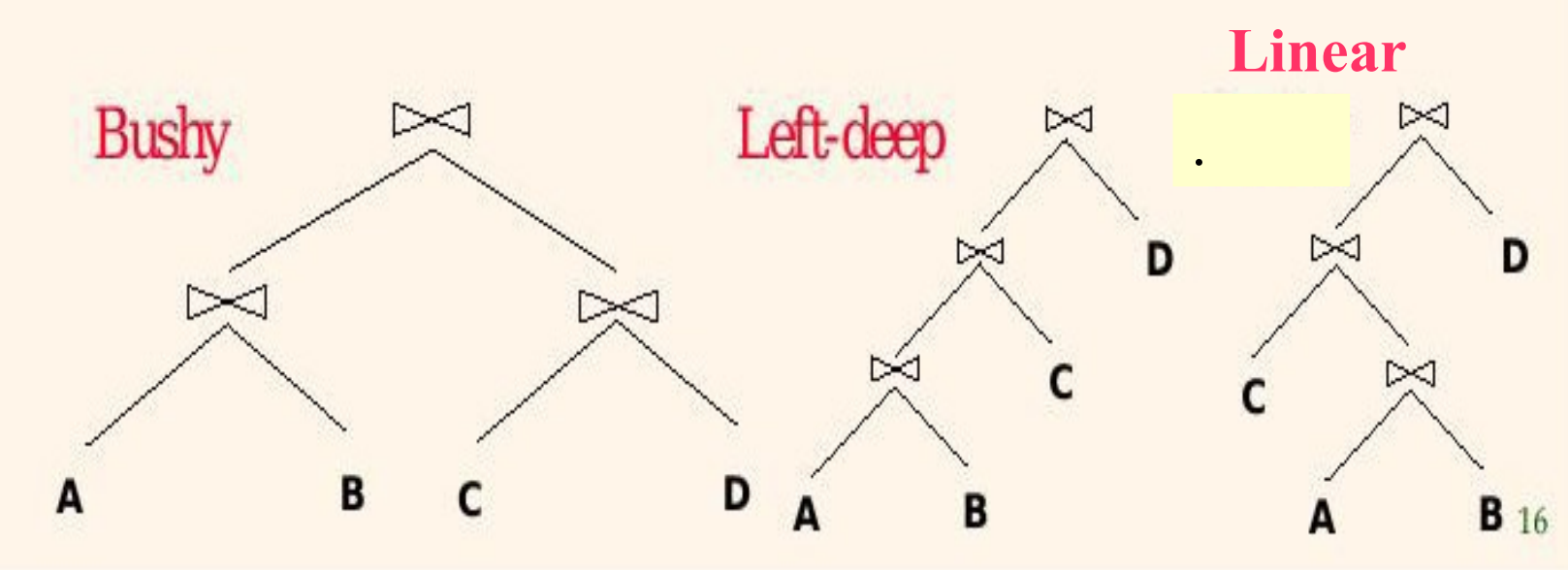
- *Transformando SQL para Álgebra Relacional*
- *Decomposição de consultas aninhadas em blocos*
- *Bloco é definido pela cláusula SELECT*
- *Otimização de blocos*
  - ✓*Identificadno expressões equivalentes*
  - ✓*Reordenação de junções*
  - ✓*Antecipação de seleções e projeções*
- *Limitação do espaço de opções*

Otimização de Consultas

Árvore de consulta linear:

- *Pelo menos um filho é uma tabela base*
- *Possibilita uso de pipeline*

Left-Deep: linear onde filho a direita é tabela base, usada para limitar o espaço de opções de planos



Otimização de Consultas

Exemplo de consulta e blocos

SELECT $S.sid,\ MIN(R.day)$

FROM $Sailors\ S, Reserves\ R, Boats\ B$

WHERE $S.sid = R.sid\ AND\ R.bid = B.bid\ AND\ B.color = '\ red'$

AND $S.rating = (SELECT\ MAX(S2.rating)\ FROM\ Sailors\ S2)$

GROUPY BY $S.sid$

HAVING $COUNT(*) > 1$

Bloco interno: (SELECT MAX(S2.rating) FROM Sailors S2)

Otimização de Consultas

Exemplo bloco externo

SELECT $S.sid,\ MIN(R.day)$

FROM $Sailors\ S, Reserves\ R, Boats\ B$

WHERE $S.sid = R.sid\ AND\ R.bid = B.bid\ AND\ B.color = '\ red'$

AND $S.rating = referencia\ ao\ bloco\ aninhado$

GROUPY BY $S.sid$

HAVING $COUNT(*) > 1$

Otimização de Consultas

Exemplo Expressão da Álgebra incluindo Group by

- *Geração do plano*
- *Transformando SQL para Álgebra Relacional*

$$\textbf{SELECT, FROM, WHERE} \rightarrow \pi,\ \times,\ \sigma$$

Sejam $g,\ h$ os operadores *GROUP BY* e *HAVING*

$$\pi_{S.sid,\ MIN(R.day)}($$
$$h_{count(*)>2}($$
$$g_{S.sid}($$
$$\sigma_{((S.sid=R.sid)\wedge(R.bid=B.bid)\wedge(B.color='red')\wedge(S.rating=subquery\_value))}$$
$$(Sailors\ S \times Reservers\ R \times Boatas\ B))))$$

Otimização de Consultas

Custo:

- *Para cada nó da árvore, capítulo 12 e 14*
  - ✓*Ver tabela de complexidades*
- *Estimar tamanhos com base no catálogo,*
- *máximo é o produto das cardinalidades*
- *reduções baseadas em termos do WHERE*

Otimização de Consultas

Estimativas de Redução

- **uso de $Nkey$ para busca com igualdade e índices**

$$(coluna = valor) \rightarrow \frac{1}{NKey(I)}$$
$$(coluna1 = coluna2) \rightarrow \frac{1}{MAX(NKey(I1),Nkey(I2))}$$

- **uso de estatíticas não havendo índice**

$$(coluna = valor) \rightarrow \frac{1}{X}$$

- **não havendo índice ou estatística**

$$(coluna = valor) \rightarrow \frac{1}{10}$$
$$(coluna1 = coluna2) \rightarrow \frac{1}{10}$$

- **busca por intervalo com e sem índices**

$$coluna > valor \rightarrow \frac{High(I)-valor}{High(I)-Low(I)}$$
$$coluna > valor \rightarrow \frac{1}{2}$$

Otimização de Consultas

Estimativas de Redução

- cláusula IN

$$coluna\ IN\ (lista\_valores) \rightarrow MAX(fator(=) \times tamanho\_lista, 50\%)$$

$$coluna\ IN\ (subconsulta) \rightarrow \frac{estimativa\ \ subconsulta}{valores\ \ distintos\ \ coluna}$$

- cláusula NOT

$$NOT\ condicao \rightarrow (1 - fator(condicao))$$

- Seja o exemplo:$(Low = 1, High = 15, coluna > 14)$ e $N = 45\ tuplas$

  Se distribuição uniforme: $\rightarrow \frac{15-14}{15-1} \times 45 = (3)$

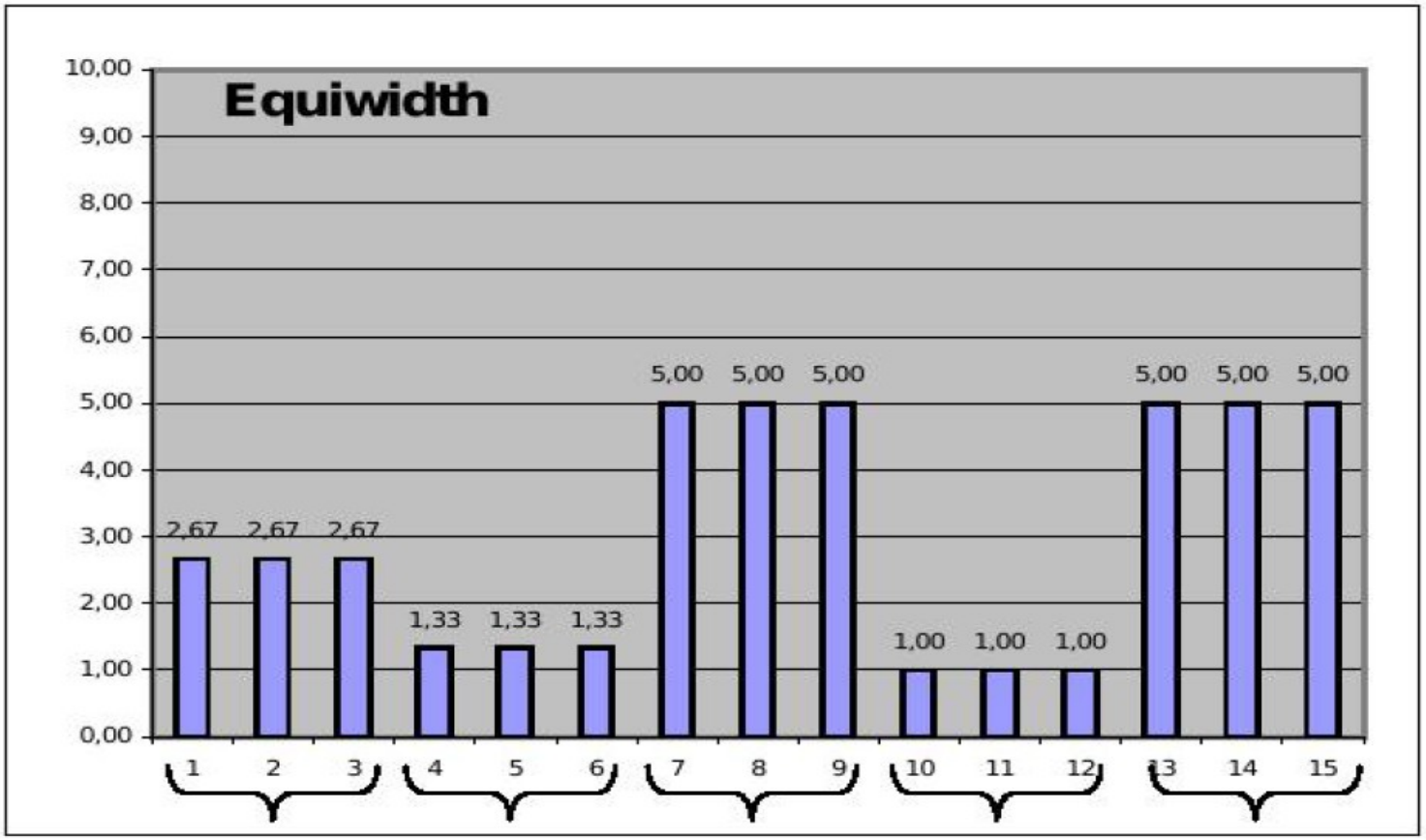
Otimização de Consultas

Estimativas de tamanho de resultado de junção, assumindo que cada tupla do menor índice tem um casamento no outro índice

$$\frac{|R| \times |S|}{MAX(NKey(Indice.c1), NKey(Indice.c2))}$$

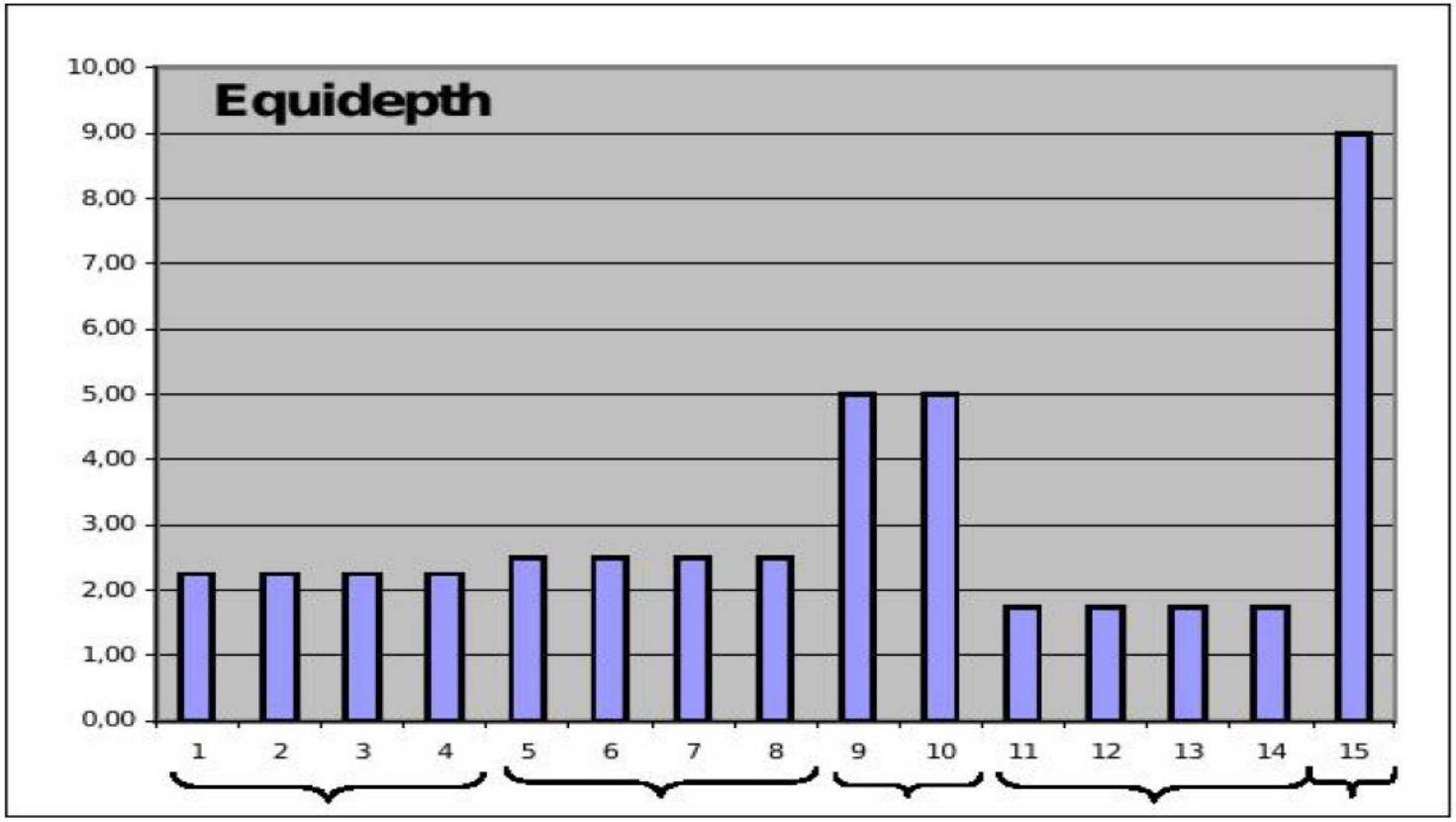
Otimização de Consultas

Estimativas de frequencia para grupos com três valores :
(coluna > 14) => 5



Otimização de Consultas

Estimativas de frequencia para grupos baseados em frequencia: (coluna > 14) => 9

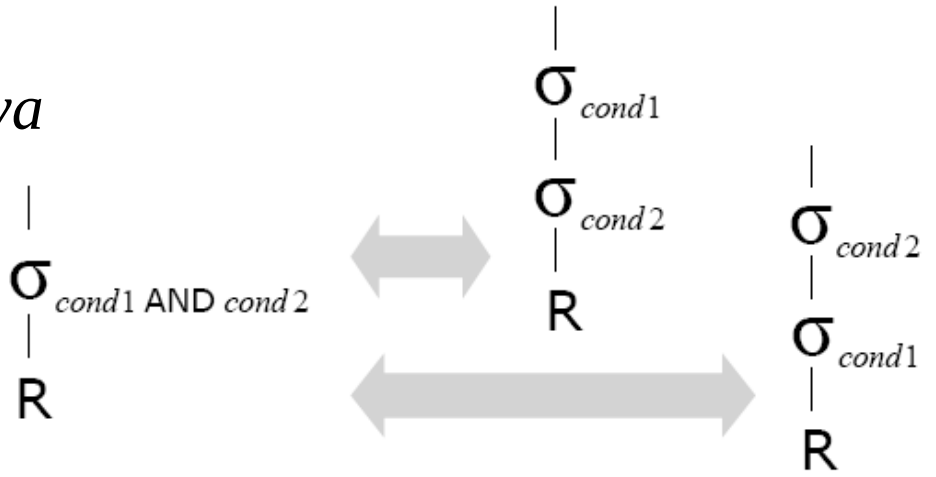


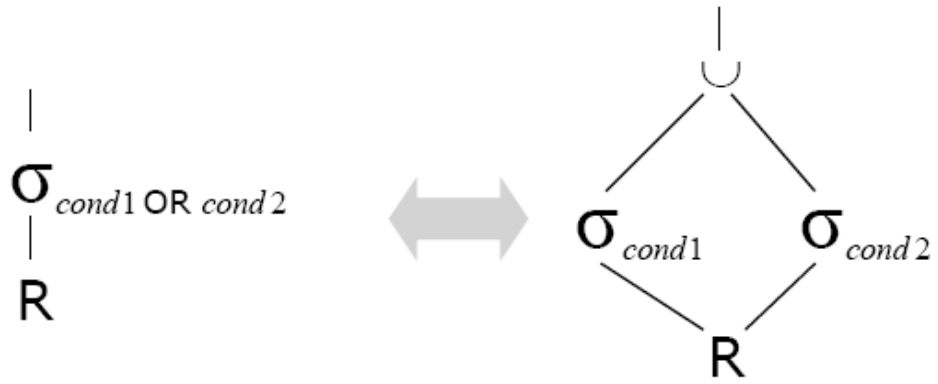
Otimização de Consultas - Equivalências

Transformações Algébricas – Predicados de Seleção

Cascata;

Comutativa

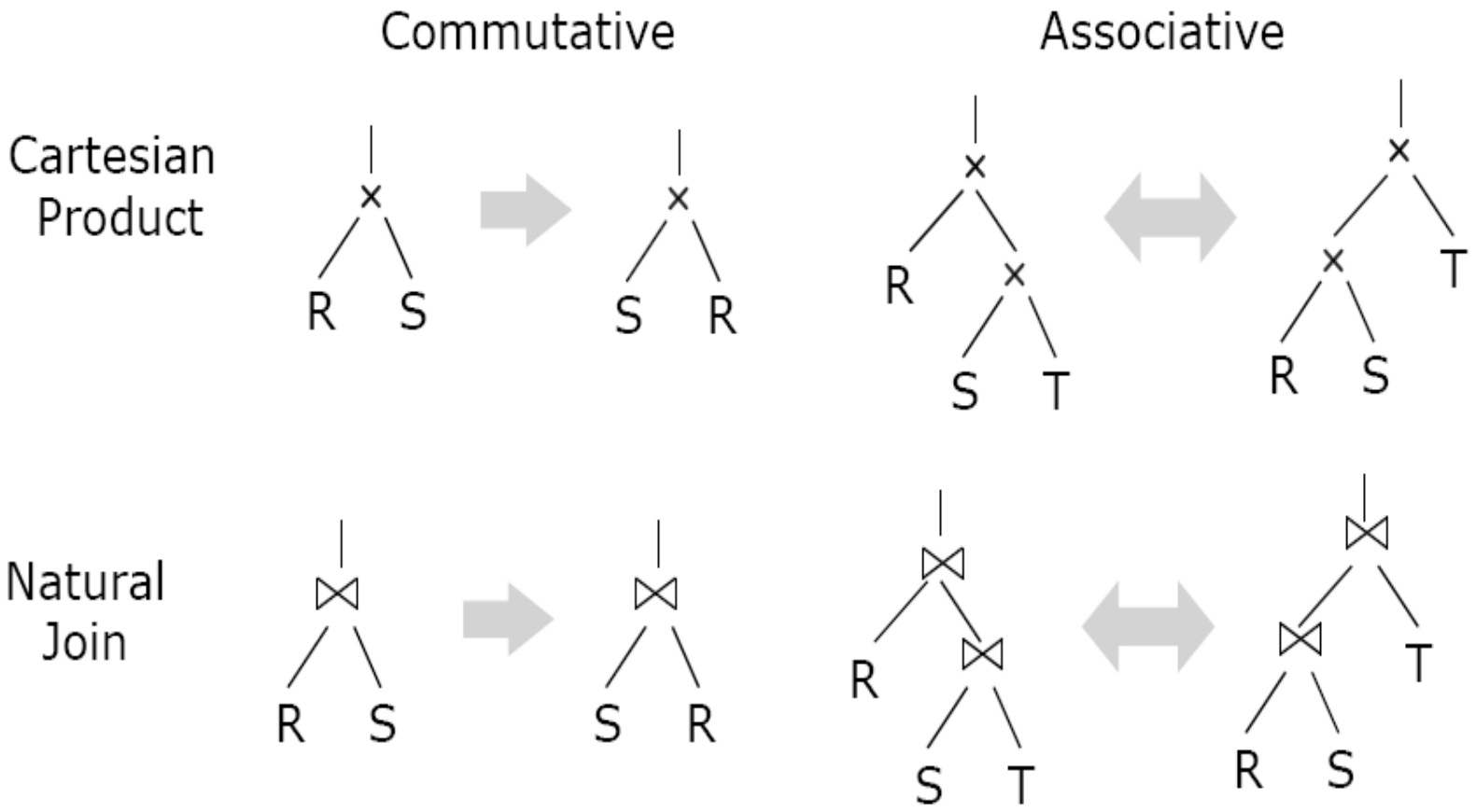




Does it apply to bags?

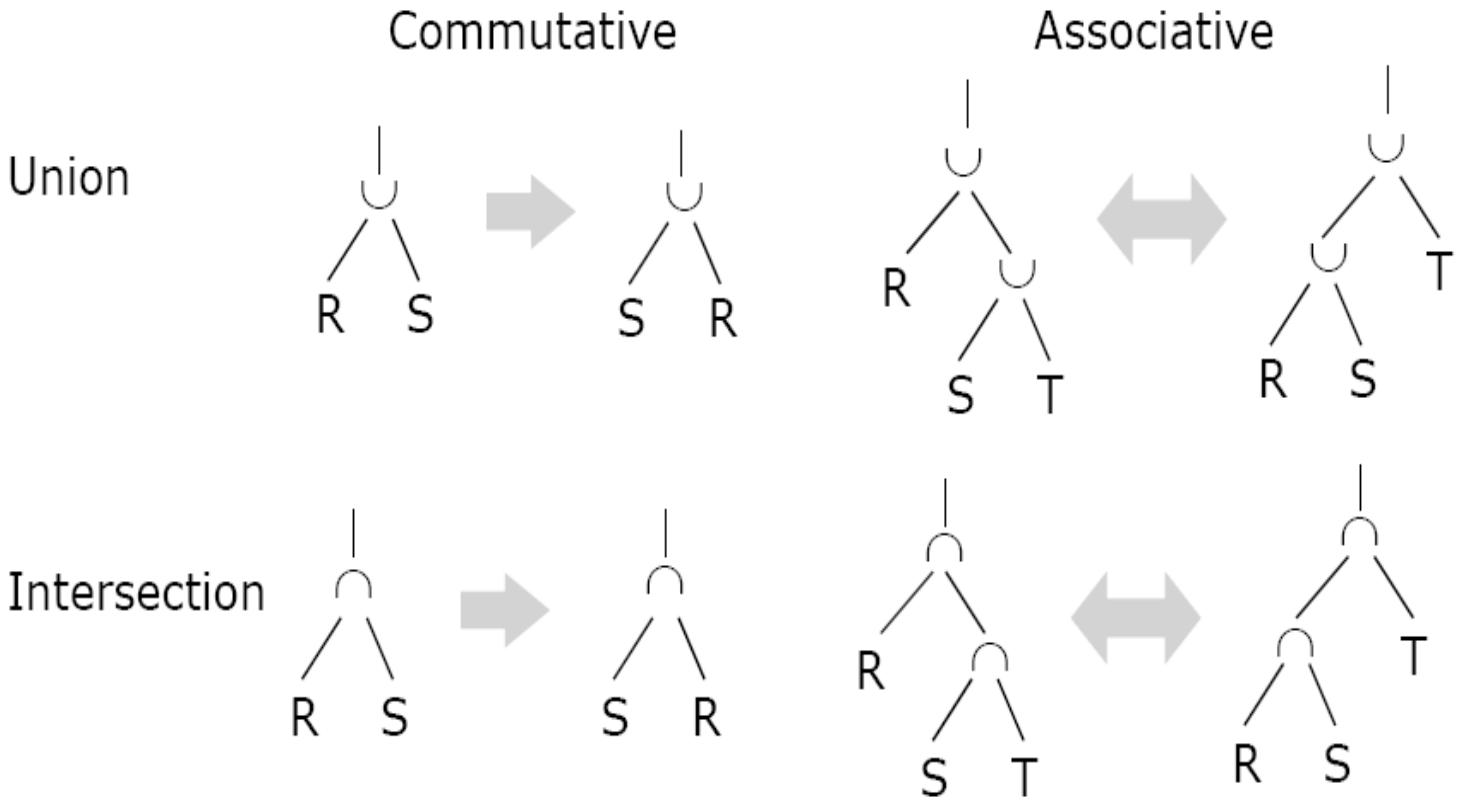
Otimização de Consultas - Equivalências

Transformações Algébricas – Junção Natural



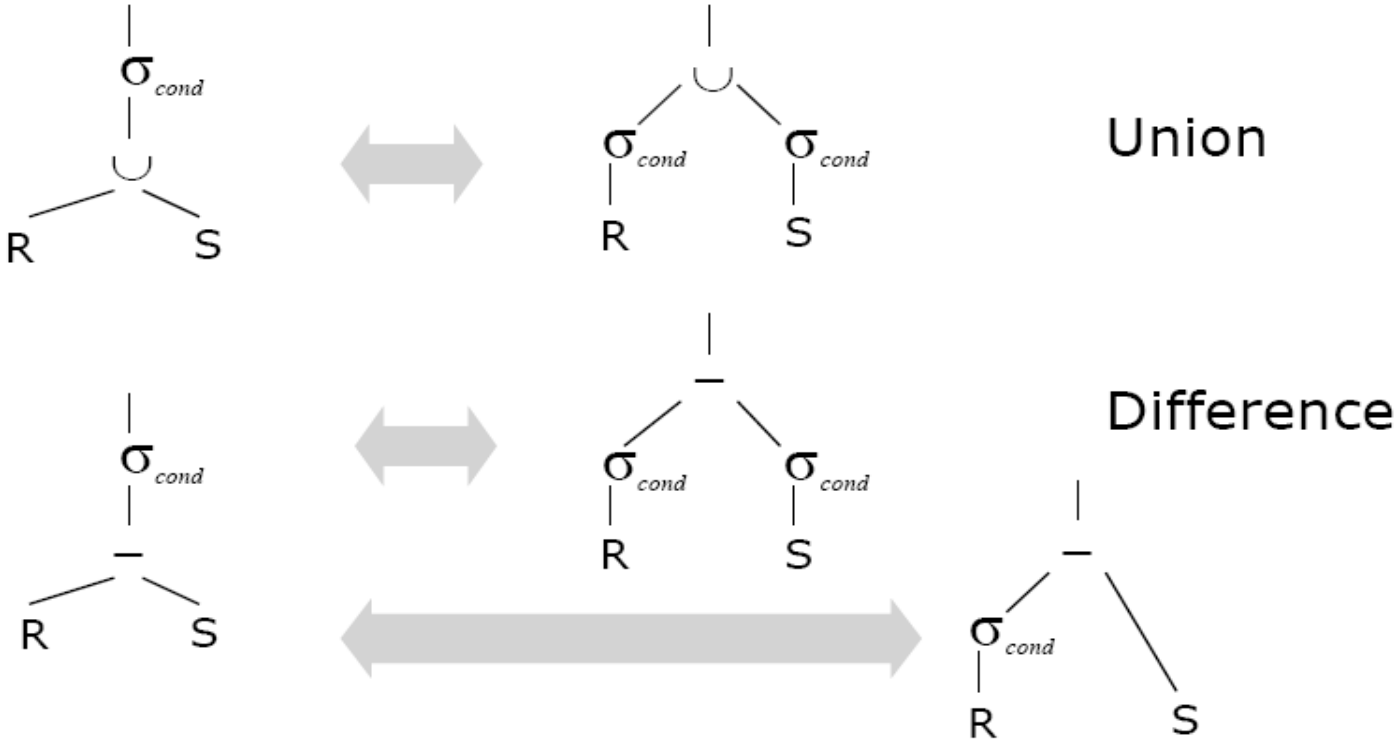
Otimização de Consultas

Transformações Algébricas – União e Intersecção



Otimização de Consultas

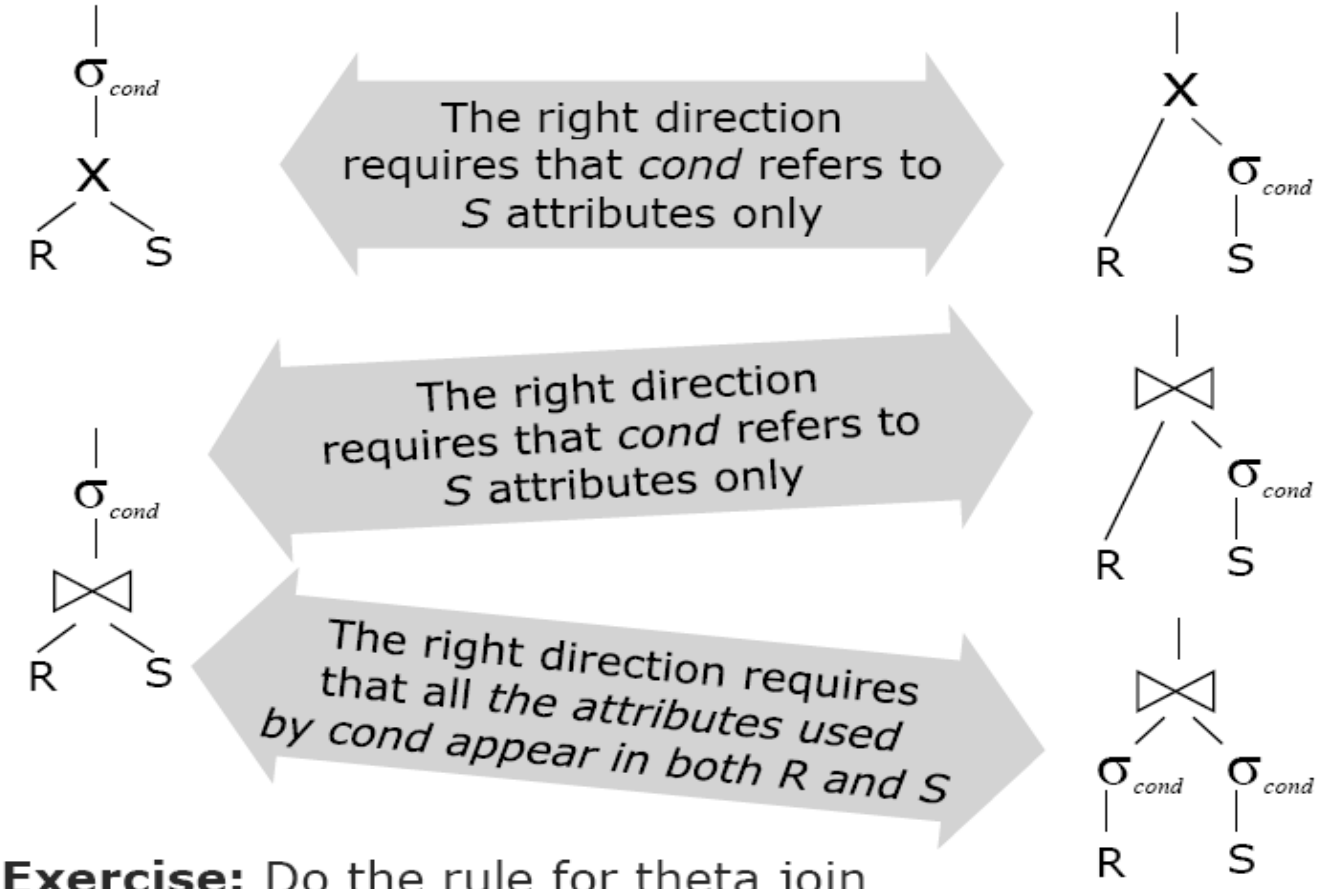
Transformações Algébricas – União e Diferença



- **Exercise:** Do the rules for intersection

Otimização de Consultas

Transformações Algébricas – Seleção e Junção



- **Exercise:** Do the rule for theta join

Transformações Algébricas - Seleção e Projeção

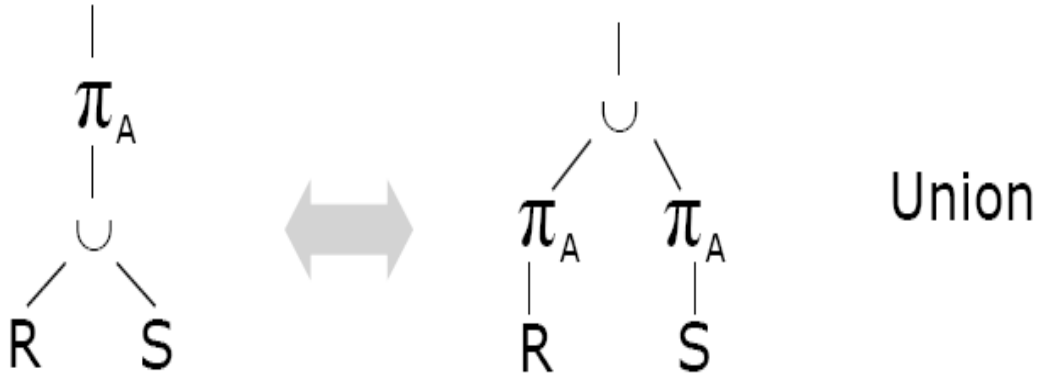
Let **X** = subset of R attributes

Z = attributes in predicate P (subset of R attributes)

$$\pi_X[\sigma_P(R)] = \pi_X\{\sigma_P[\pi_{XZ}(R)]\}$$

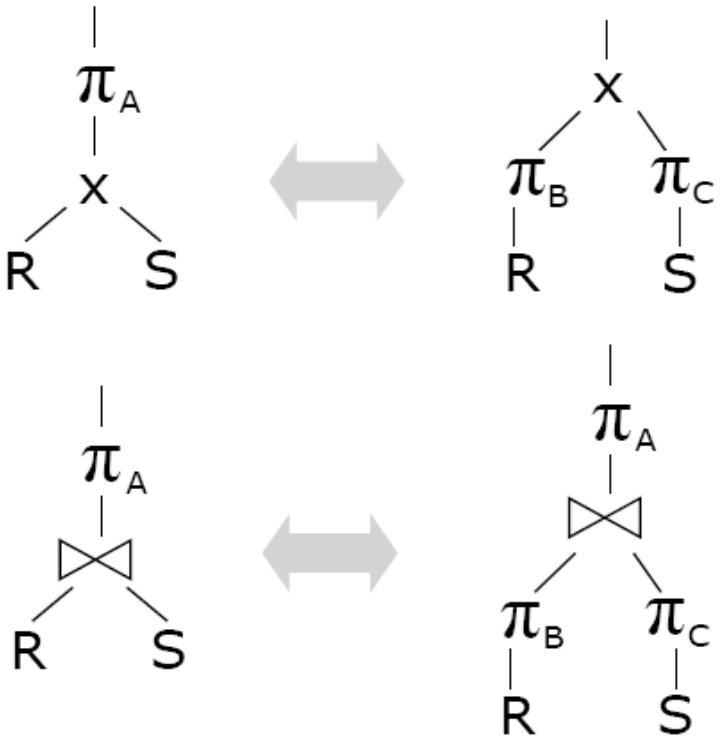
Transformações Algébricas - União e Projeção

- A projection is simple if it only consists of an attribute list



Otimização de Consultas

Transformações Algébricas - Antecipando Projeção

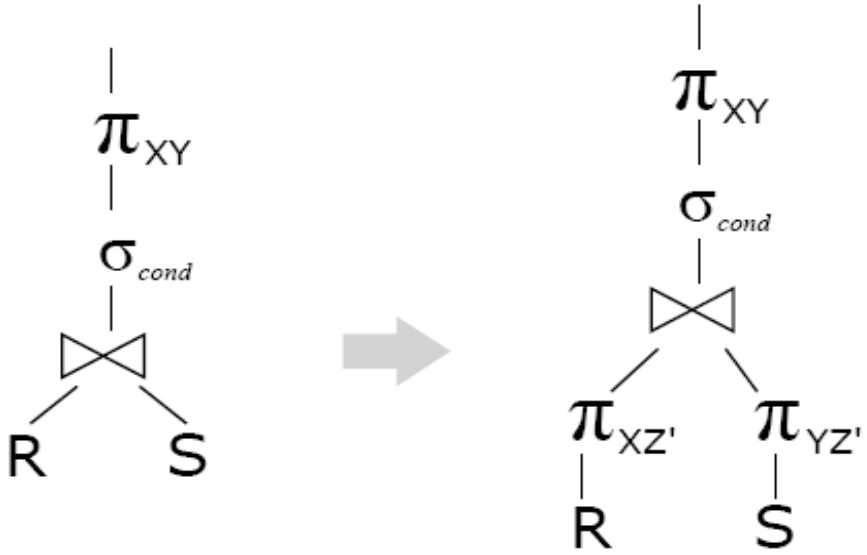


- B is the list of R attributes that appear in A
- Similar for C

- **Question:** What is B and C ?

Otimização de Consultas

Transformações Algébricas – Seleção, Projeção, Junção



- Z′ = Z U {attributes used in *cond*}

Planos

Enumeração de planos envolvendo múltiplas relações $(\bowtie, \times, \sigma, \pi)$

SELECT *lista_atributos LA*

FROM *lista_relacoes LR*

WHERE $term_1 \wedge term_2 \wedge \cdots \wedge term_n$

Enumerando planos *left-deep*:

Planos

Enumerando planos *left-deep*:

- Passo 1: Para cada $A \in LR$

  identificar termos que necessitam de atributos de A

  identificar atributos de A não mencionados em termos/LA

  identificar métodos de acesso para seleção/projeção em A

  considere o melhor plano para cada ordem de saída

Planos

Passo 2: Para cada plano gerado no Passo 1

Sua saída(A) será tabela *outer* e cada outra(B) será *inner*

Para cada B como *inner*, identifique:

seleções em B que podem ser aplicadas antes do $\bowtie$

atributos de A e B que definem a $\bowtie$

seleções com outras relações que serão aplicadas após $\bowtie$

Assuma que:

tuplas de A são *pipelined* para a junção

alguns planos exigem materialização

escolha acesso a B para cada método de junção com A

Planos

Passo 3: repetir sucessivamente o Passo 2, considerando como tabela *outer* a saída do passo anterior, até obter planos com todas as relações na cláusula FROM.

Otimização de Consultas

Considerações sobre subconsultas aninhadas

Otimizadores trabalham melhor com junções e, em geral, subconsultas não são bem otimizadas

– O principal problema é com subconsultas correlatas, por exemplo:

```
SELECT S.sname
FROM   Sailors S
WHERE EXISTS (
        SELECT *
        FROM Reserves R
        WHERE R.bid=103 AND R.sid=S.sid
        )
```

Em geral, a estratégia usada é *INDEX NESTED LOOP*, com subconsulta como tabela *inner*

Otimização de Consultas

Subconsultas executadas uma só vez

```
SELECT S.sname
FROM   Sailors S
WHERE S.sid IN (
            SELECT R.sid
            FROM Reserves R
            WHERE R.bid=103)
```

Neste caso, a subconsulta deve ser executada primeiro.

Otimização de Consultas

OBS: nem todos os otimizadores são capazes de transformar uma consulta aninhada em versão não aninhada, então, sempre que possível, especificar consultas não aninhadas ou não correlatas usando, principalmente, junções.

Por exemplo, a consulta correlata acima, pode ser:

SELECT $S.sname$
FROM $Sailors\ S, Reserves\ R$
WHERE $R.bid = 103\ AND\ R.sid = S.sid$

Neste caso o otimizador deverá explorar as várias alternativas para a junção.

Exercícios – Processamento de Consultas

EXERCÍCIOS CAP 12, 14 e 15 LIVRO-TEXTO

FIM – Processamento de Consultas

** material baseado no livro-texto e páginas públicas da internet*