centro universitário FEI

AULA 5- PROCESSAMENTO DE CONSULTAS - PII

PROFA. DRA. LEILA BERGAMASCO

CC6240 – Tópicos Avançados em Banco de Dados

RECAPITULANDO

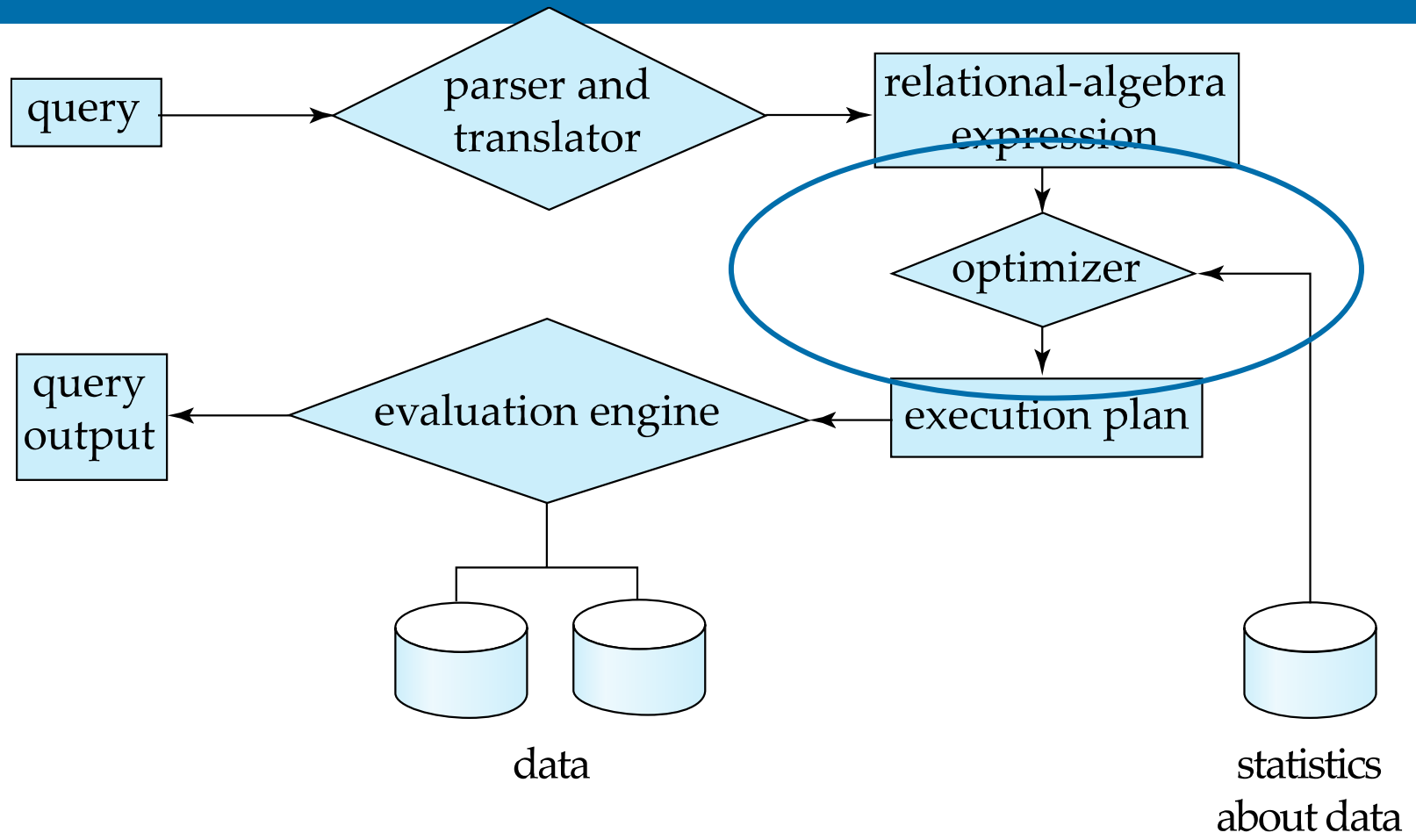
- Como SGBD processa consultas
- Fatores envolvidos no processamento de consultas
- Custos de consulta da operação de seleção, com igualdade, >, >=,<, <=
  - Ex. : $\sigma_{num-agencia=123}$(Agencia)

centro universitário FEI

AGORA VEREMOS

- Processamento de Consulta
  - Classificação
  - Junção

PROCESSAMENTO DE CONSULTAS



Silberchartz, 2018

COMO ESTIMAR O CUSTO DE UMA PEC?

Operação	Algoritmo
Seleção	A1 (busca linear)
	A2 (busca binária)
	A3 (índice primário, igualdade de chave)
	A4 (índice primário, igualdade em atributo não chave)
	A5 (índice secundário, igualdade)
	A6 (índice primário, comparação)
	A7 (índice secundário, comparação)
	A8 (seleção de conjunção usando um índice)
	A9 (seleção de conjunção usando índice composto)
	A10 (seleção de conjunção usando intersecção de identificadores)
	A11 (seleção de disjunção usando união de identificadores)
Classificação	sort-merge externo
Junção	Junção de laço aninhado
	Junção de laço aninhado de blocos
	Junção de laço aninhado indexada
	Merge-junção
	Hash-Junção

CLASSIFICAÇÃO

- Finalidade
  - **Ordenar** os dados
- Por que?
  - 1) É possível que a própria consulta SQL peça para que o resultado retorne de forma ordenada: ORDER BY
  - 2) Algumas operações relacionais mais complexas como junção podem ser mais eficientemente processadas se as relações de entrada estiverem ordenadas
    - Ex. $\sigma_{num-agencia=123}$(Agencia |X|Cliente)

centro universitário FEI

CLASSIFICAÇÃO

- É possível criar um índice sobre o atributo a ser ordenado/classificado.
  - Operação lógica e não física
    - Logo...Dependendo da quantidade de registros a consulta acessará mais de um bloco atrás do próximo registro ordenado
    - PROBLEMÁTICO

- Solução: Ordenar fisicamente
  - Como?
    - Se todos os registros da relação a ser classificada estão na memória principal, podemos utilizar técnicas clássicas como quicksort
    - **Se não estiverem**: classificação externa

CLASSIFICAÇÃO EXTERNA

- O algoritmo mais utilizado é o sort-merge externo
- M = quantidade de blocos que cabem na memória principal

```
Fase 1
i = 0
repita
   1. leia M blocos da relação (ou relação até final se for menor que M)
   2. ordene a parte que está na memória
   3. escreva dados ordenados no arquivo temporário Ri – gerando arquivos
   i = i + 1
até fim da relação
```

CLASSIFICAÇÃO EXTERNA

- O algoritmo mais utilizado é o sort-merge externo
- M = quantidade de blocos que cabem na memória principal

Fase 2

```
1. leia um bloco de cada um dos N arquivos Ri para uma página
de buffer na memória (considerando N < M)
repita
   2. escolha a primeira tupla entre todas as páginas de buffer
   3. escreva tupla no resultado e apague-a da página do buffer se página de
buffer de qualquer Ri for vazia e não for fim de Ri
   4. leia próximo bloco de Ri na página do buffer até todas páginas de buffer
estiverem vazias
fim
```

centro universitário FEI

a1	a2
d	5
a	10
g	54
c	98
h	21
j	20

d	5
a	10
g	54
c	98
h	21
j	20

a	10
d	5
g	54
c	21
h	21
j	20

a	10
d	5
g	54

c	21
h	21
j	20

a	10
c	21

g	54

a	10
b	21
d	5

h	21
j	20

a	10
b	21
d	5

h	21
j	20

a	10
b	21
d	5
g	54

h	21
j	20

a	10
b	21
d	5
g	54
h	21

j	20

a	10
b	21
d	5
g	54
h	21
j	20

Supondo que em memória caibam 3 blocos, que 1 tupla ocupe 1 bloco , logo → memória suporta 3 blocos.

Cada sub-parte será ordenada em memoria e armazenada em um arquivo temporário

Fase dois: Olhamos para as tuplas iniciais de cada 2 blocos (1 sera usada para resposta) e ordenamos. Guardamos em arquivo temporário novamente

Classificação final

CLASSIFICAÇÃO EXTERNA

- Custo
  - todo bloco da relação é lido e escrito novamente: $\boldsymbol{2b_r}$
  - número inicial de temporários: $\frac{\boldsymbol{b_r}}{\boldsymbol{M}}$
  - número de temporários diminui por um fator de $M - 1$ em cada passo de merge
    - número total de passos de merge necessários: teto de $(\log_{M-1}(\frac{b_r}{M}))$.
  - $b_r(2[\log_{M-1}(\frac{b_r}{M})]+1)$

centro universitário FEI

a1	a2
d	5
a	10
g	54
c	98
h	21
j	20

d	5
a	10
g	54
c	98
h	21
j	20

a	10
d	5
g	54
c	21
h	21
j	20

a	10
d	5
g	54
c	21
h	21
j	20

Qual o custo?

1 tupla – 1 bloco
Capacidade de memória (M) –
3 frames/blocos -> M=3

$$b_r(2[\log_{M-1}(\frac{b_r}{M})]+1)$$

$$= 6 * (2 * [\log_2(\frac{6}{3})] + 1)$$

6*(2*[1]+1) =6*3 = 18

a	10
b	21
d	5
g	54
h	21
j	20

d	5
g	54
c	21
h	21
j	20

a	10

g	54
h	21
j	20

a	10
b	21
d	5

g	54
h	21
j	20

a	10
b	21
d	5

h	21
j	20

a	10
b	21
d	5
g	54

j	20

a	10
b	21
d	5
g	54
h	21

Classificação final

JUNÇÃO

- Como visto na aula passada, durante o processo de junção, reunimos informação de duas relações com base em atributos chave e estrangeiros.
- Vamos utilizar o termo join para indicar a junção natural r|X|s, onde r e s são relações
- O que queremos saber?
  - Algoritmos e seus custos para executar junções

centro universitário FEI

JUNÇÃO

- Vários algoritmos:
  - Junção de laço aninhado
  - Junção de laço aninhado de blocos
  - Junção de laço aninhado indexada
  - Merge-junção
  - Hash-Junção

Relação externa

SELECT *
FROM tableA
INNER JOIN tableB on tableA.id = tableB.id

JUNÇÃO DE LAÇO ANINHADO

Relação externa

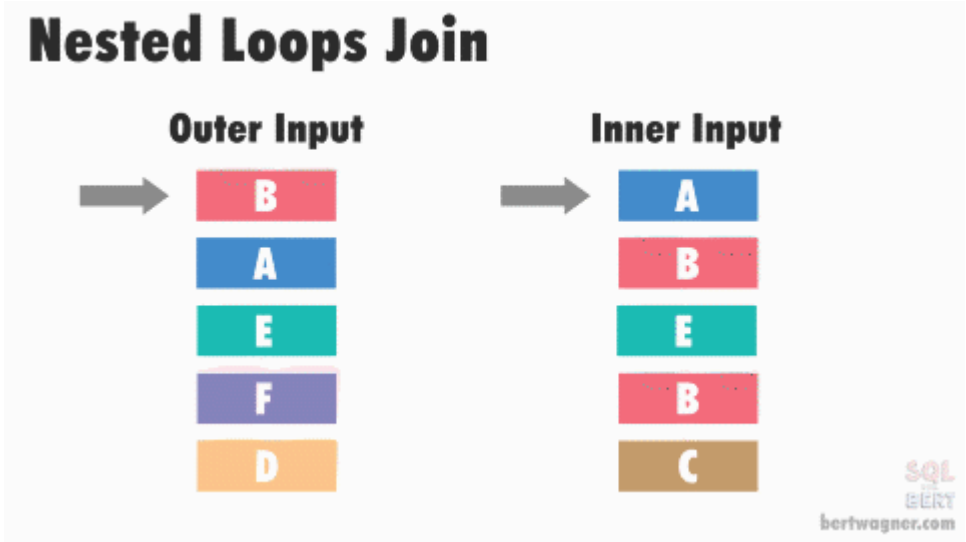
Relação interna

- Junção de laço aninhado --- r|X|s

```
para cada tupla tr, em r faça              // r é relação externa
    para   cada tupla ts em s faça  // s é relação interna se par                (tr,ts)
satisfaz condição θ
          adicione tr,ts ao resultado
    fim para
fim para
```

- custo semelhante varredura linear de arquivo para seleção
  - Comparação da primeira tupla de uma relação externa com todas as outras tuplas da relação interna
    - Lógica de comparação todos com todos.
  - número de pares de tupla a ser considerado: $n_r * n_s$
  - pior caso: buffer pode manter somente um bloco de cada relação -> $b_r + (n_r * b_s)$ acessos de blocos necessários
  - melhor caso: buffer consegue manter relações inteiras na memória -> $b_r + b_s$ acessos de blocos

JUNÇÃO DE LAÇO ANINHADO



https://bertwagner.com/2018/12/11/visualizing-nested-loops-joins-and-understanding-their-implications/

LEMBRANDO: COMO ESTIMAR O CUSTO DE UMA PEC?

- n_r- número de tuplas na relação r
- b_r – número de blocos que contêm tuplas da relação r
- s_r– tamanho em bytes de uma tupla da relação r
- f_r– fator de bloco da relação r: número de tuplas da relação r que cabe em um bloco
- V(A,r)– número de valores distintos para o atributo A na relação r
- SC(A,r) – cardinalidade da seleção do atributo A da relação r (número médio de registros que satisfazem uma condição de igualdade no atributo A)
- f_i – fanout médio dos nós internos do índice i para índices estruturados em árvore
- HT_i– número de níveis no índice i (altura do índice). Se árvore balanceada, HT_i=[$\log_{fi}$ (V(A,r))]
- LB_i – número de blocos de índice de nível mais baixo no índice i (nível de folha)

Relação externa (r)

Relação interna (s)

JUNÇÃO DE LAÇO ANINHADO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$

`depositante |x| cliente`

V(nome_cliente,depositante) = 2500 -> em média um cliente possui duas contas
V(nome_cliente,cliente) = 10000 -> em média um cliente possui duas contas

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando junção aninhada?

1) depositante como relação externa
- pior caso = 100+(5000 * 400) = 2.000.100 acessos de bloco
- melhor caso = ler ambas relações de uma só vez = 100 + 400 = 500 acessos a blocos

2) cliente como relação externa
- pior caso: 400+(10000 * 100) = 1.000.400 acessos de bloco
- melhor caso = 500 acessos a blocos

Pior caso: $n_r * b_s + b_r$
Melhor caso: $b_r + b_s$

JUNÇÃO DE LAÇO ANINHADO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$
$V(nome\_cliente,depositante)$
$V(nome\_cliente,cliente) = 10$

depositante |x| cliente

1 MILHÃO DE ACESSOS. CUSTO MUITO ALTO

n_r- número de tuplas na re
b_r – número de blocos que
s_r– tamanho em bytes de un
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando junção aninhada?

depositante como relação externa
pior caso = 5000 * 400 + 100 = 2.000.100
cessos de bloco
melhor caso = ler ambas relações de uma
vez = 100 + 400 = 500 acessos a blocos

cliente como relação externa
pior caso: 10000 * 100 + 400 = 1.000.400
cessos de bloco
- melhor caso = item (1)

Pior caso: $n_r * b_s + b_r$
Melhor caso: $b_r + b_s$

JUNÇÃO DE LAÇO ANINHADO EM BLOCOS

- Junção de laço aninhado de blocos

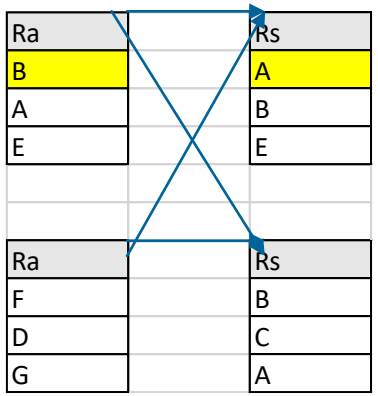
```
para cada bloco Br de r faça       // r é relação externa
     para cada bloco Bs de s faça  // s é relação interna
          para cada tupla tr em Br faça
               para cada tupla ts em Bs
                    faça se par (tr,ts) satisfaz condição θ
                         adicione tr,ts ao resultado
               fim para
          fim para
     fim para
fim para
```

- Dividir para conquistar: Separamos as tuplas em blocos e comparamos as tuplas em cada bloco paralelamente.
  - cada bloco da relação interna é emparelhado com cada bloco da relação externa;
  - dentro de cada par de blocos, as tuplas são emparelhadas
  - pior caso: cada bloco da relação interna s é lido apenas uma vez para cada bloco da relação externa r -> $b_r + (b_r * b_s)$ acessos
  - melhor caso: $b_r + b_s$ acessos de blocos

JUNÇÃO DE LAÇO ANINHADO EM BLOCOS

Ra
B
A
E
F
D
G

Rs
A
B
E
B
C
A



Ra
B
A
E

Rs
A
B
E

Ra
A
E

Rs
A
E

Ra
E

Rs
E

Join aninhado Simples

- (6*2)+2 = 14

Join aninhado por Blocos

1. (2*2)+2 = 6

Dividir para conquistar, em paralelo fez a mesma coisa no segundo bloco

JUNÇÃO DE LAÇO ANINHADO EM BLOCOS

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$

`depositante |x| cliente`

V(nome_cliente,depositante) = 2500 -> em média um cliente possui duas contas
V(nome_cliente,cliente) = 10000 -> em média um cliente possui duas contas

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando laço aninhado de blocos?

1) depositante como relação externa (r)
•pior caso = 100 * 400 + 100 = 40100 acessos de bloco
•melhor caso = ler ambas relações de uma só vez = 100 + 400 = 500 acessos a blocos

2) cliente como relação externa
•pior caso: 400 * 100 + 400 = 40400 acessos de bloco
•melhor caso = item (1)

Pior caso: $(b_r * b_s) + b_r$
Melhor caso: $b_r + b_s$

JUNÇÃO DE LAÇO ANINHADO EM BLOCOS

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$
V(nome_cliente,depositante) = 2500 -> e
V(nome_cliente,cliente) = 10000 -> em n

`depositante |x| cliente`

Exemplo anterior:
No pior caso
junção aninhada: ~2.000.000 acessos.

Qual o custo utilizando laço aninhado de blocos?

depositante como relação externa (r)
or caso = 100 * 400 + 100 = 40100
ssos de bloco
elhor caso = ler ambas relações de uma vez = 100 + 400 = 500 acessos a blocos

cliente como relação externa
•pior caso: 400 * 100 + 400 = 40400 acessos de bloco
•melhor caso = item (1)

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Pior caso: $(b_r * b_s) + b_r$
Melhor caso: $b_r + b_s$

JUNÇÃO DE LAÇO ANINHADO INDEXADA

- Como melhorar desempenho dos 2 algoritmos vistos ?
  - se atributos formarem chave na relação interna: laço interno pode parar quando a primeira correspondência for encontrada;
  - varrer laço interno alternadamente para frente e para trás -> ordena varredura de forma que dados da varredura anterior que permanecem no buffer podem ser usados novamente reduzindo acesso a disco;
  - usar índice no atributo de junção do laço interno (próximo algoritmo).

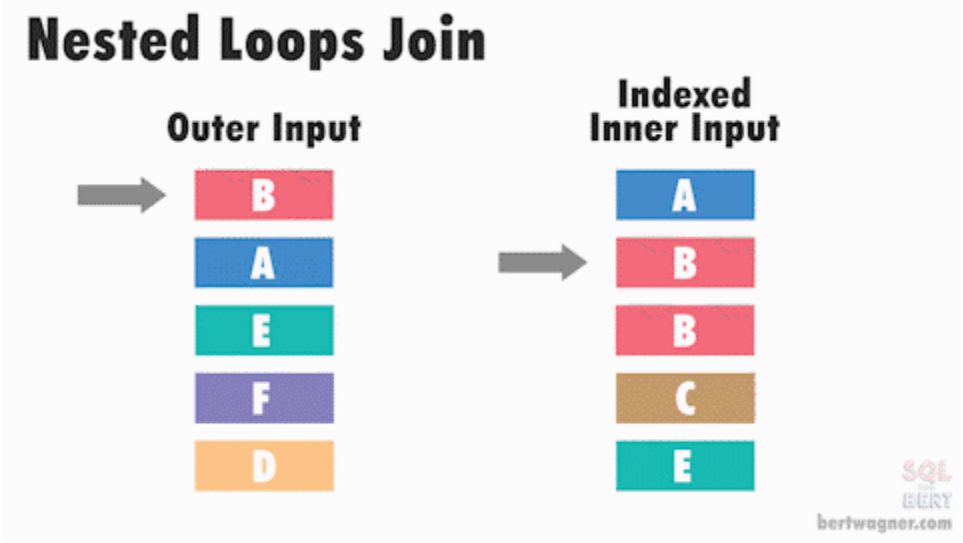
JUNÇÃO DE LAÇO ANINHADO INDEXADA

- Junção de laço aninhado indexada

```
para cada tupla tr, em r faça      // r é relação externa
      usar índice para procurar tuplas ts em s que satisfaz condição θ
      adicione tr,ts ao resultado
fim para
```

- alternativa quando há índice para atributo de junção do laço interno
- índices podem ser permanentes ou temporários
- procura de tuplas é uma seleção em s
- pior caso: espaço no buffer somente para uma página de r e uma página do índice → $b_r + (n_r * c)$
- c é o custo de seleção única em s usando condição de junção (Hti+SC/fr)

JUNÇÃO DE LAÇO ANINHADO INDEXADA



https://bertwagner.com/2018/12/11/visualizing-nested-loops-joins-and-understanding-their-implications/

JUNÇÃO DE LAÇO ANINHADO INDEXADO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$

`depositante |x| cliente`

V(nome_cliente,depositante) = 2500 -> em média um cliente possui duas contas
V(nome_cliente,cliente) = 10000 -> em média um cliente possui duas contas

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando laço aninhado indexado?

1) depositante como relação externa (r)
•pior caso = 100+5000*c
c=log_(20/2)2500+((5000/2500)/50)
c=log_10 2500+0.08 = 4 +1 = 4.33 = 5
==100+5000*5 = 25100

2)Cliente como relação externa(r)
Pior caso = 400+10000*c
C= 4+[SC/fr]
SC=10000/10000=1 // 1/25=1
Pior caso = 400+10000*5 = 50400

Pior caso: $b_r + (n_r * c)$

$$c = Ht_i + \frac{SC}{f_r}$$

$$HT_i = [\log_{\frac{f_i}{2}}(\mathrm{V(A,r)})\ ]$$

JUNÇÃO DE LAÇO ANINHADO INDEXADO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$
V(nome_cliente,depositante) = 2500 -
V(nome_cliente,cliente) = 10000 -> e

`depositante |x| cliente`

Exemplos anteriores:
No pior caso
- junção aninhada: ~2.000.000 acessos.
- junção aninhada em blocos: ~41000 acessos.

1) depositante como relação externa (r)
•pior caso = 100+5000*c
c=log_(20/2)2500+((5000/2500)/50)
C=log_102500+0.08 = 3.33+1 = 4.33 = 5

externa(r)
c

SC=10000/10000=1 // 1/25=1
Pior caso = 400+10000*5 = 50400

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

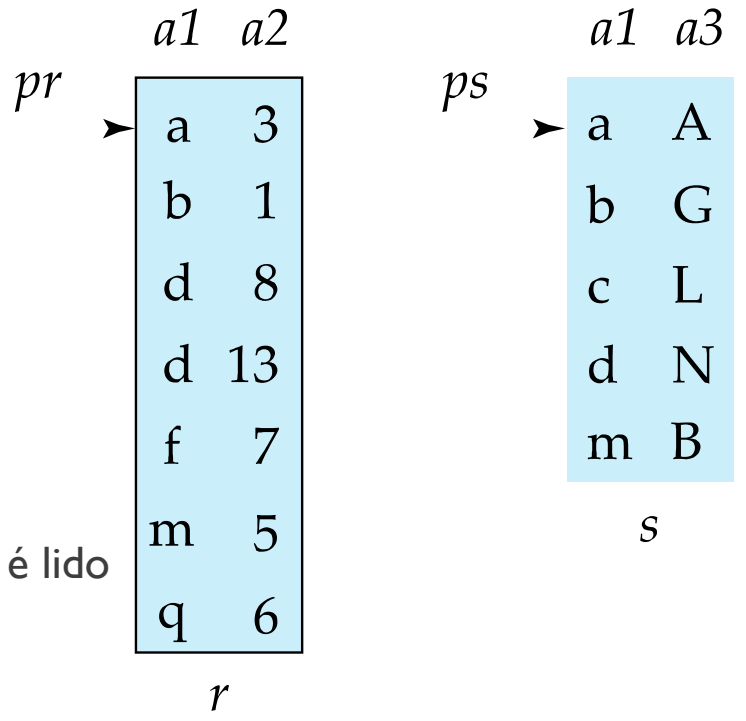
Pior caso: $b_r + (n_r * c)$

$$c = Ht_i + \frac{SC}{f_r}$$

$$HT_i = [\log_{\frac{f_i}{2}}(\mathrm{V(A, r)})\ ]$$

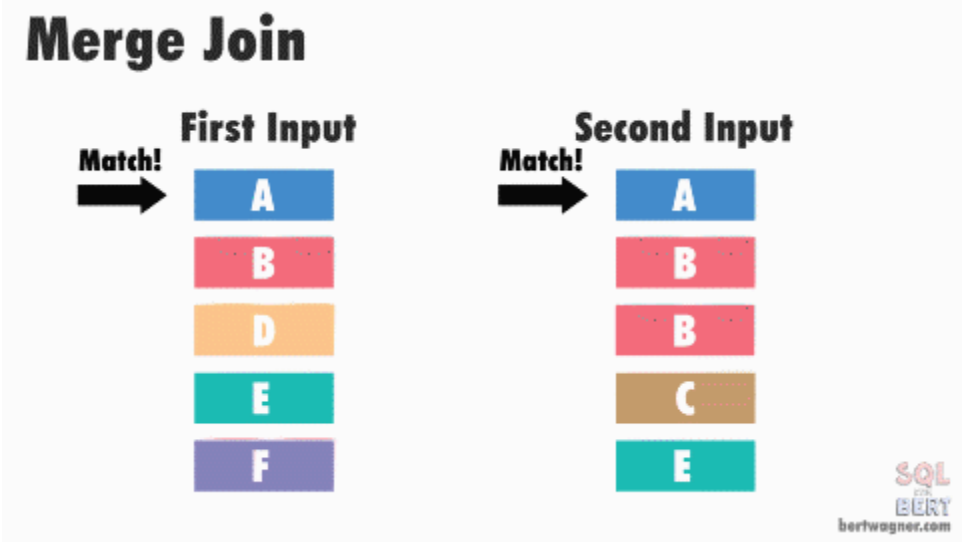
MERGE-JUNÇÃO

- Merge-junção
  - usado para calcular junção natural ou equi-join
  - r(R) e s(S): relações sobre as quais será calculada junção
  - R ∩ S: atributos em comum
- Algoritmo:
  - Ambas as relações são ordenadas pelo mesmo atributo
  - associa ponteiro a cada relação – apontam para 1ª tupla
  - ponteiros se movem ao longo da relação
  - grupo de tuplas de uma relação com mesmo valor para os atributos de junção é lido em S
  - tuplas correspondentes da outra relação são lidas e armazenadas



- Custo: $b_r + b_s$
  - Cada tupla será lida apenas uma vez

MERGE-JUNÇÃO



https://bertwagner.com/2018/12/11/visualizing-nested-loops-joins-and-understanding-their-implications/

MERGE X JUNÇÃO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$

`depositante |x| cliente`

V(nome_cliente,depositante) = 2500 -> em média um cliente possui duas contas
V(nome_cliente,cliente) = 10000 -> em média um cliente possui duas contas

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando merge junção?

1) depositante como relação externa (r)
- 400+100 =500

E se atributo não estiver ordenado?
Classifica-lo! $b_r(2[\log_{M-1}(\frac{b_r}{M})]+1)$

Considerando tamanho de memória no pior caso = 3 blocos:
- custo para classificar cliente = 7200
- custo para classificar depositante = 1600
- custo total = 7200 + 1600 + 100 + 400 = 9300 acessos

Pior caso: $b_r + b_s$ (e custo adicional de classificação se necessário)

MERGE-JUNÇÃO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$
$V(nome\_cliente,depositante) = 250$
$V(nome\_cliente,cliente) = 10000$ ->

`depositante |x| cliente`

Exemplos anteriores:
No pior caso
- junção aninhada: ~2.000.000 acessos.
- junção aninhada em blocos: ~41000 acessos.
- junção aninhada com index: ~25000 acessos.

n_r- número de tuplas na relaçã
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando merge junção?

1) depositante como relação externa (r)
• 400+100 =500

E se atributo não estiver ordenado?
$\lceil_{M-1}(\frac{b_r}{M})\rceil+1)$

no de memória no pior

cliente = 7200
•custo para classificar depositante = 1600
•custo total = 7200 + 1600 + 100 + 400 = 9300 acessos

Pior caso: $b_r + b_s$ (e custo adicional de classificação se necessário)

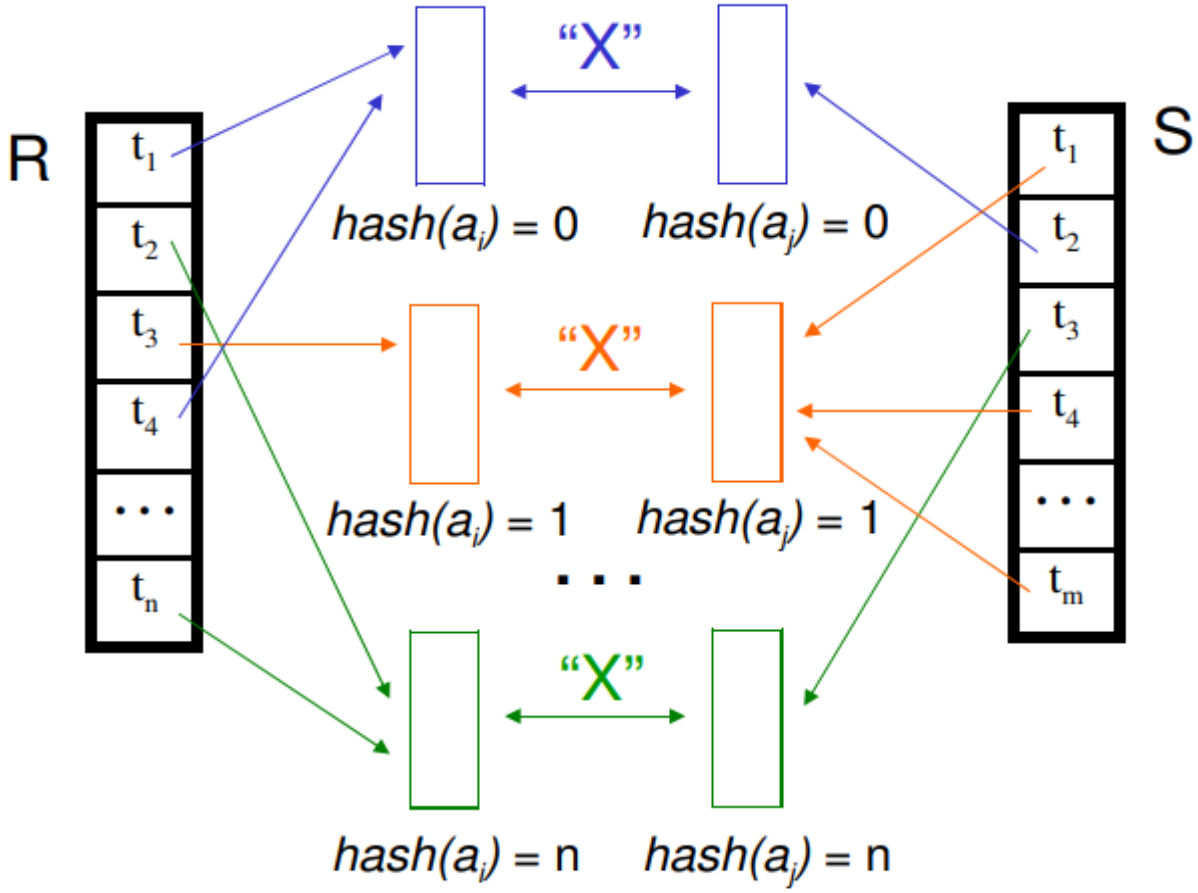
HASH JUNÇÃO

- Hash-junção
  - usado para calcular junção natural ou equi-join
  - R ∩ S: atributos em comum (atributos de junção)
  - Função hash é usada para particionar as tuplas de ambas as relações em conjuntos que têm o mesmo valor nos atributos de junção
  - h: função hash que faz o mapeamento de R ∩ S
  - Hr0, Hr1,..., Hrmax : partições das tuplas de r
  - Hs0, Hs1,..., Hsmax : partições das tuplas de s

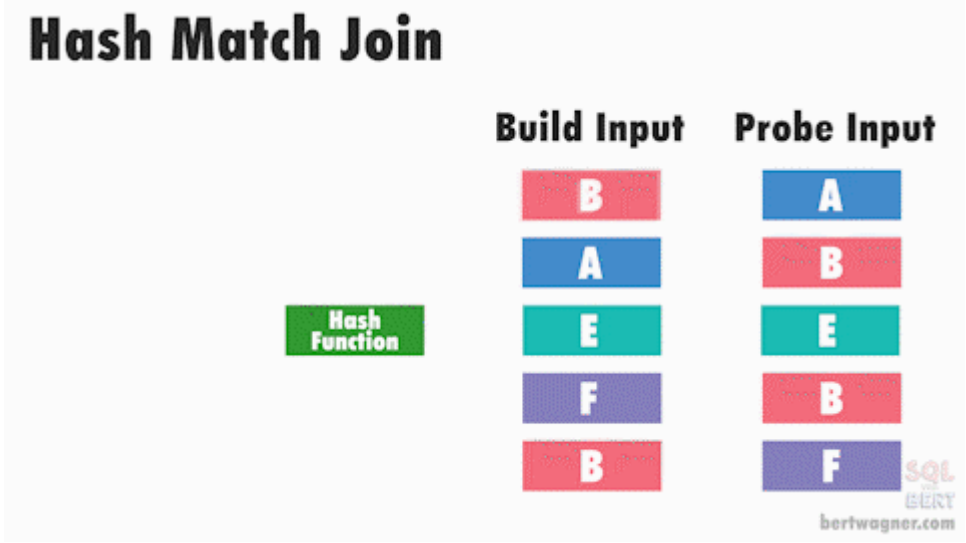
HASH JUNÇÃO

- Hash-junção – Algoritmo
  - se uma tupla de r e uma tupla de s satisfazem condição de junção → têm mesmo valor para atributos de junção
- Melhor opção:
  - B(R) <= M
    - Escaneia R, constrói buckets na memória principal;
    - Escaneia S e realiza o join.
    - Custo: B(R) + B(S)
  - Se não: Necessário o particionamento, fazendo com que pelo menos os buckets caibam na memória principal
    - Melhor custo: 3(br+bs) -> 2 para particionar e 1 para fazer o match
    - Se nem os buckets cabem na memória (particionamento recursivo): $2(b_r + b_s)\ [log_{\lfloor M-1}(b_s)-1] + b_r + b_s$

3(BR+BS) →


R
S
t1
t2
t3
t4
tn
tm
"X"
"X"
"X"
hash(ai) = 0
hash(aj) = 0
hash(ai) = 1
hash(aj) = 1
hash(ai) = n
hash(aj) = n

HASH JUNÇÃO



https://bertwagner.com/2018/12/11/visualizing-nested-loops-joins-and-understanding-their-implications/

HASH JUNÇÃO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 100$

depositante |x| cliente

V(nome_cliente,depositante) = 2500 -> em média um cliente possui duas contas
V(nome_cliente,cliente) = 10000 -> em média um cliente possui duas contas

n_r- número de tuplas na relação *r*
b_r – número de blocos que contêm tuplas da relação *r*
s_r– tamanho em bytes de uma tupla da relação *r*
f_r– fator de bloco da relação *r*: número de tuplas da relação *r* que cabe em um bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando hash junção?
Supondo que M =3

1) depositante tem menos tuplas, logo é o escolhido para fazer a construção

$=2(b_r + b_s)[\log_{M-1}(b_r) - 1] + b_r + b_s$
$= 2(100 + 400)[\log_2 100 - 1] + 100 + 400$
= 1000[7-1]+500
=1000*6+500
= 6500 acessos

Se escolhêssemos clientes -> 8500 acessos

Pior caso: $2(b_r + b_s)[\log_{M-1}(b_r)-1] + b_r + b_s$
Melhor caso: $b_r + b_s$

HASH JUNÇÃO

$n_{cliente} = 10000$
$f_{cliente} = 25$
$b_{cliente} = \frac{10000}{25} = 400$
$n_{depositante} = 5000$
$f_{depositante} = 50$
$b_{depositante} = \frac{5000}{50} = 10$
V(nome_cliente,depositar
V(nome_cliente,cliente) =

`depositante |x| cliente`

Exemplos anteriores:
No pior caso
- junção aninhada: ~2.000.000 acessos.
- junção aninhada em blocos: ~41000 acessos.
- junção aninhada com index: ~25000 acessos.
- Merge junção: ~10000 acessos.

n_r- número de tuplas r
b_r – número de blocos
s_r– tamanho em bytes
f_r– fator de bloco da re
bloco
V(A,r)– número de valores distintos para o atributo A na relação *r*
SC(A,r) – cardinalidade da seleção do atributo A da relação *r* (número médio de registros que satisfazem uma condição de igualdade no atributo A)

Qual o custo utilizando hash junção?
Supondo que M =3

1) depositante tem menos tuplas, logo é o escolhido para fazer a construção

$) [\log_{M-1}(b_r) - 1] + b_r + b_s$
$400)[\log_2 100 - 1] + 100 +$

]+500
0
sos

Pior caso: $2(b_r + b_s) [log_{M-1}(b_r)-1] + b_r + b_s$
Melhor caso: $b_r + b_s$

centro universitário FEI

RESUMÃO

	Nested Loop Join	Hash Join	Merge Join
Estratégias de índices	Aplique índices no atributo da condição da relação interna	-	Aplique índices em ambos os atributos da condição
Bom se	Relação externa é pequena	A tabela hash cabe na memória principal `show work_mem;`	Ambas as tabelas são grandes

```
SET enable_hashjoin = off;
SET enable_mergejoin = off;
SET enable_nestloop = off;
```

Altera somente a sessão atual!

MOMENTO "E O KIKO PROFESSORA?"

- Performance
  - Alguns exemplos:
    - Em geral, realizar consultas no qual o WHERE utiliza atributos que são índices, torna sua consulta mais rápida
    - Realizar consultas com igualdade é sempre mais eficiente do que utilizar >,<
    - Ao realizar joins, verifique os atributos da sua condição de igualdade
      - É possível aplicar um índice sobre o atributo?
        - Se sim, FAÇA.

BIBLIOGRAFIA

- ABRAHAM SILBERSCHATZ, HENRY F. KORTH, S. SUDARSHAN. Sistema de Banco de Dados. 6. Campus. 0. ISBN 9788535245356.

- ELMASRI, RAMEZ, SHAMKANT B. NAVATHE. Sistemas de banco de dados. Vol. 6. São Paulo: Pearson Addison Wesley, 2011.

- DATE, CHRISTHOPER J. Introdução a Sistemas de Bancos de Dados, 5ª. Edição. Campus, Rio de Janeiro (2004).

OBRIGADO E ATÉ A PRÓXIMA AULA!