Compilação, Otimização e Execução de Consultas – parte II

Cristina Dutra de Aguiar Ciferri

Função *Hash*

R	S
n registros e j atributos	m registros e k atributos

- $R \cup S$; $R \cap S$; $R - S$
- Passos
  - particiona cada relação usando a mesma função *hash*
  - cria H_{R_0}, ..., $H_{R_{max}}$ e H_{S_0}, ..., $H_{s_{max}}$
  - para cada partição i ($0 \leq i \leq max$)
    - constrói um índice *hash* na memória em H_{r_i}

Função *Hash*

Particionar utilizando a mesma função Hash –- h(x) = X MOD 4

Departamento (R)			
id_Depto	Nome	Id_ger	X mod 4
4	RH	8	0
1	Produção	2	2
3	Gerencia	1	1
2	TI	7	3

Funcionário (S)				
id_func	Nome	Depto	Salario	Data_Admiss
2	Antonio	1	1500	02/02/2002
6	Bruno	1	1500	03/02/2002
10	Cesar	1	1500	04/02/2002
14	Dalva	1	1500	05/02/2002
5	Eleonor	3	2000	06/02/2002
9	Francisca	3	2000	07/02/2002
3	Gilson	2	2500	08/02/2002
7	Heitor	2	2500	09/02/2002
12	Ivone	4	3000	10/02/2002
4	Kelly	4	3000	11/02/2002
8	Luana	4	3000	12/02/2002
18	Maria	1	1500	13/02/2002
11	Mario	2	1500	14/02/2002
15	Nair	2	3000	15/02/2002
1	Otávio	3	4000	16/02/2002
13	Odacir	3	4000	17/02/2002

Função *Hash*

Particionar utilizando a mesma função Hash –- h(x) = X MOD 4

Departamento			
id_Depto	Nome	Id_ger	
4	RH	**8**	H_{R0}
3	Gerencia	**1**	H_{R1}
2	TI	**7**	H_{R3}
1	Produção	**2**	H_{R2}

Funcionario					
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

Função *Hash*

Particionar utilizando a mesma função Hash –- h(x) = X MOD 4

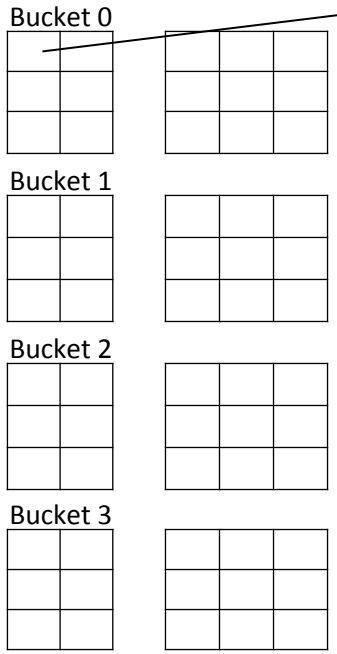
Departamento		
id_Depto	Nome	Id_ger
4	RH	**8**
3	Gerencia	**1**
2	TI	**7**
1	Produção	**2**

H_{R0} H_{R1} H_{R3} H_{R2}

Funcionario				
id_func	Nome	Depto	Salario	Data_Admiss
12	Ivone	4	3000	10/02/2002
4	Kelly	4	3000	11/02/2002
8	Luana	4	3000	12/02/2002
5	Eleonor	3	2000	06/02/2002
9	Francisca	3	2000	07/02/2002
1	Otávio	3	4000	16/02/2002
13	Odacir	3	4000	17/02/2002
3	Gilson	2	2500	08/02/2002
7	Heitor	2	2500	09/02/2002
11	Nair	2	3000	15/02/2002
15	Mario	2	1500	12/01/2002
2	Antonio	1	1500	02/02/2002
6	Bruno	1	1500	03/02/2002
10	Cesar	1	1500	04/02/2002
14	Dalva	1	1500	05/02/2002
18	Maria	1	1500	13/02/2002

H_{S0} H_{S1} H_{S3} H_{S2}

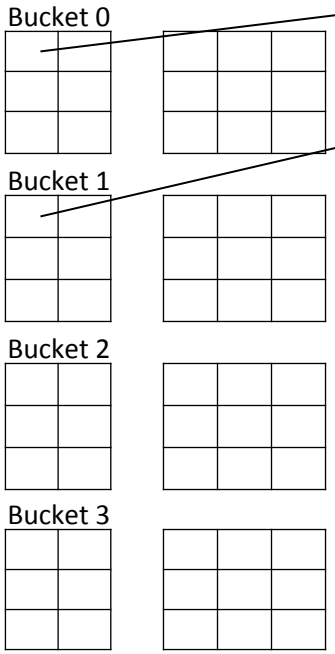
Função *Hash*



Departamento			
id_Depto	Nome	Id_ger	
4	RH	**8**	H_{R0}
3	Gerencia	**1**	H_{R1}
2	TI	**7**	H_{R3}
1	Produção	**2**	H_{R2}

Funcionario					
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

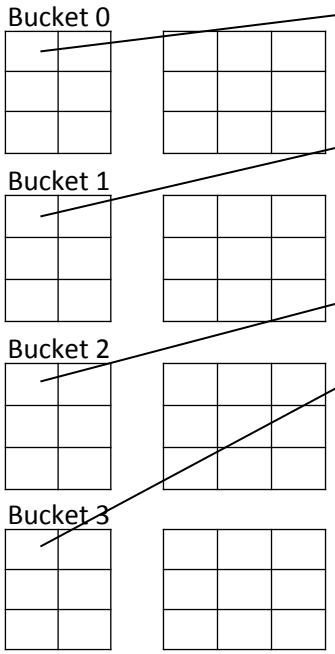
Função *Hash*



Departamento			
id_Depto	Nome	Id_ger	
4	RH	**8**	H_{R0}
3	Gerencia	**1**	H_{R1}
2	TI	**7**	H_{R3}
1	Produção	**2**	H_{R2}

Funcionario					
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

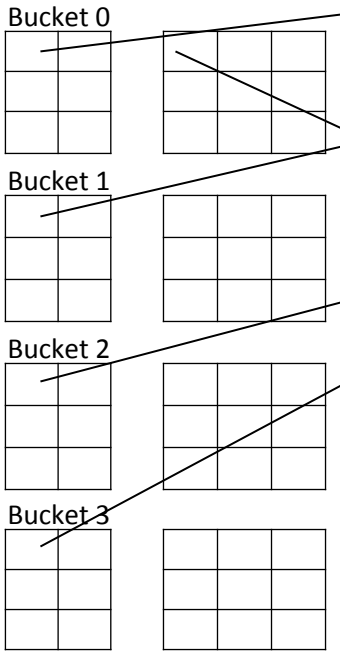
Função *Hash*



Departamento			
id_Depto	Nome	Id_ger	
4	RH	**8**	H_{R0}
3	Gerencia	**1**	H_{R1}
2	TI	**7**	H_{R3}
1	Produção	**2**	H_{R2}

Funcionario					
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

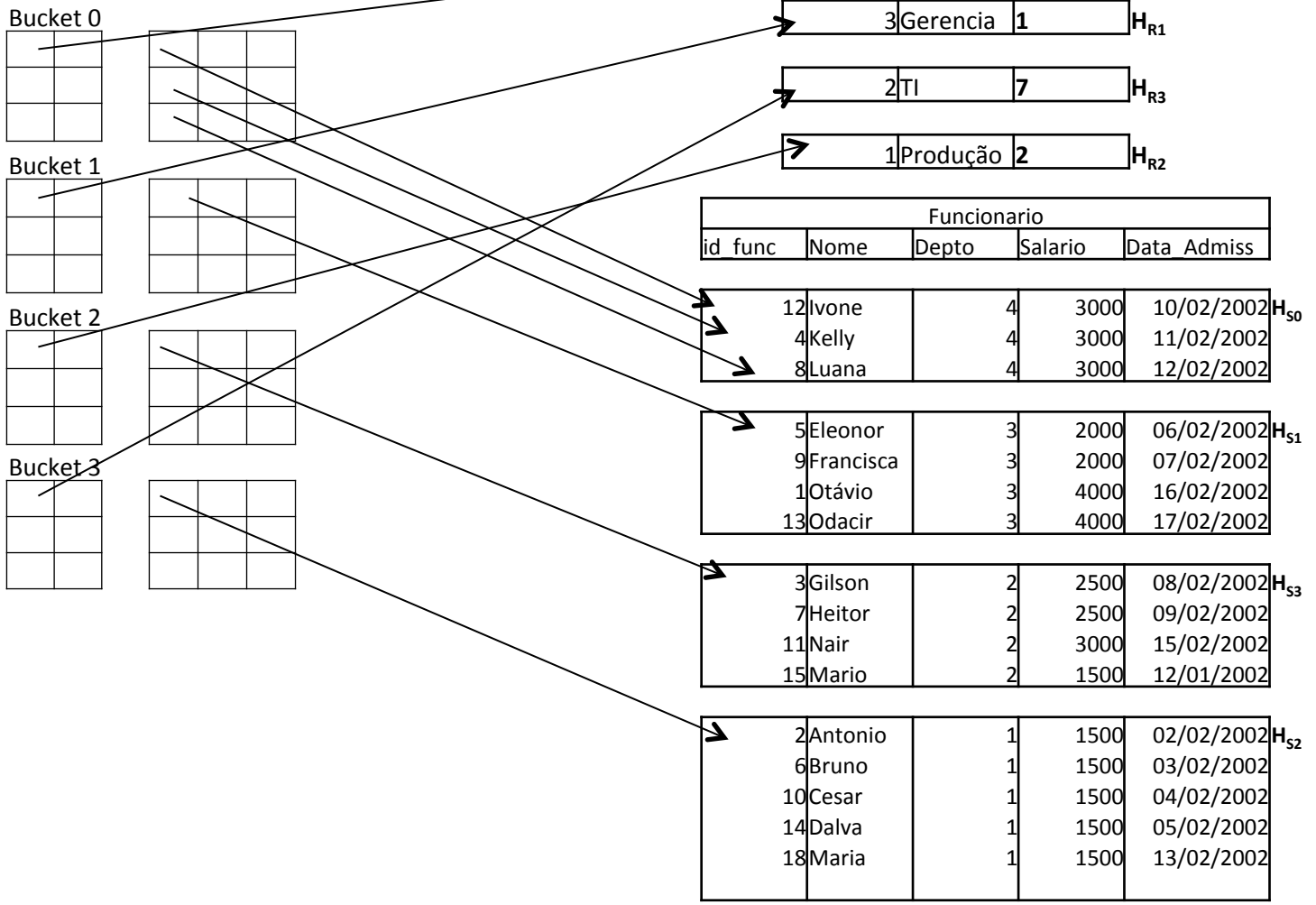
Função *Hash*



Departamento			
id_Depto	Nome	Id_ger	
4	RH	**8**	H_{R0}
3	Gerencia	**1**	H_{R1}
2	TI	**7**	H_{R3}
1	Produção	**2**	H_{R2}

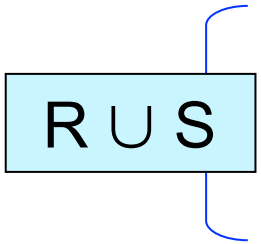
Funcionario					
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

Função *Hash*

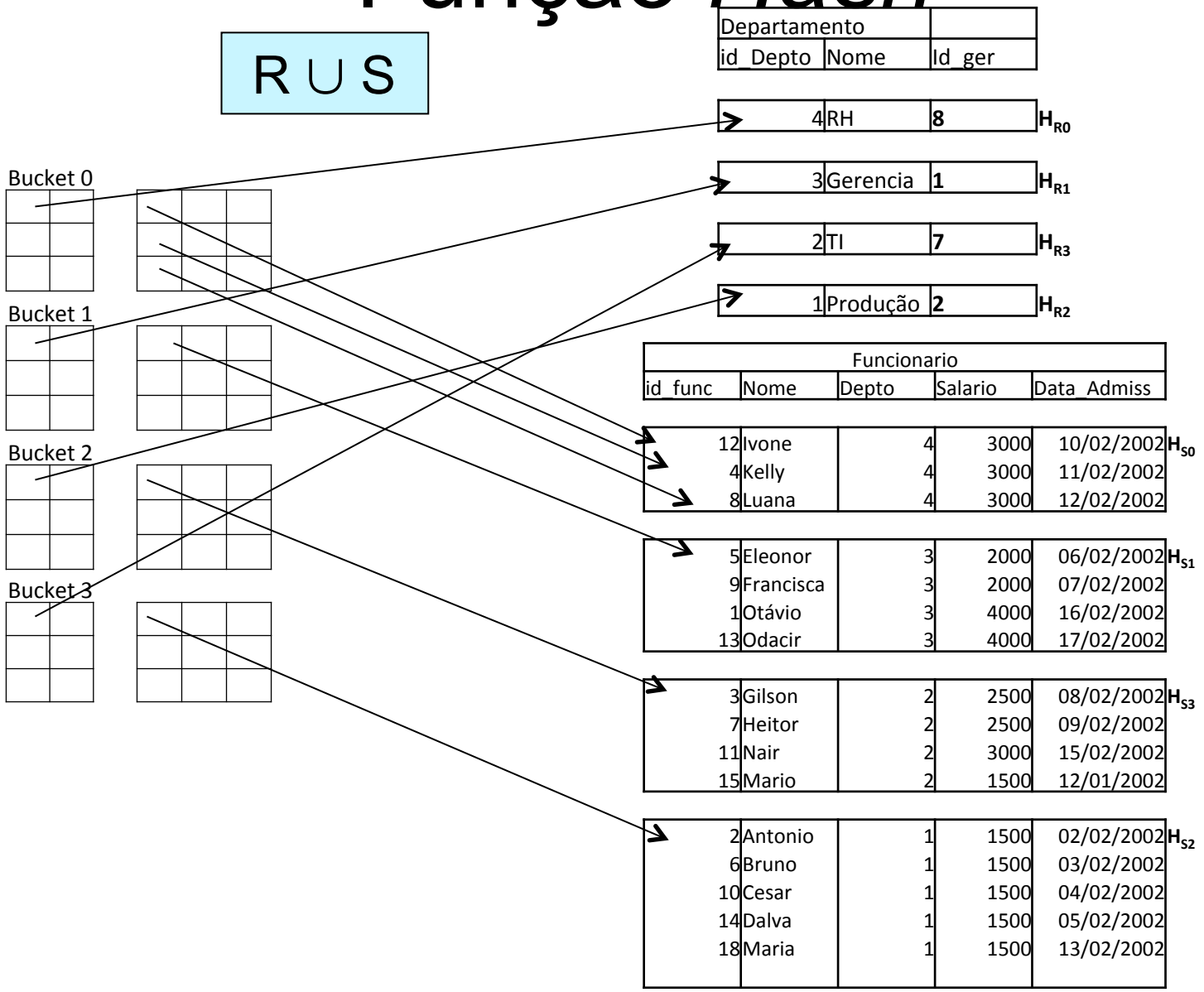

Bucket 0
Bucket 1
Bucket 2
Bucket 3
Departamento
id_Depto Nome Id_ger
4 RH 8 H_{R0}
3 Gerencia 1 H_{R1}
2 TI 7 H_{R3}
1 Produção 2 H_{R2}
Funcionario
id_func Nome Depto Salario Data_Admiss
12 Ivone 4 3000 10/02/2002 H_{S0}
4 Kelly 4 3000 11/02/2002
8 Luana 4 3000 12/02/2002
5 Eleonor 3 2000 06/02/2002 H_{S1}
9 Francisca 3 2000 07/02/2002
1 Otávio 3 4000 16/02/2002
13 Odacir 3 4000 17/02/2002
3 Gilson 2 2500 08/02/2002 H_{S3}
7 Heitor 2 2500 09/02/2002
11 Nair 2 3000 15/02/2002
15 Mario 2 1500 12/01/2002
2 Antonio 1 1500 02/02/2002 H_{S2}
6 Bruno 1 1500 03/02/2002
10 Cesar 1 1500 04/02/2002
14 Dalva 1 1500 05/02/2002
18 Maria 1 1500 13/02/2002

Função *Hash*

- Passos
- particiona cada relação usando a mesma função *hash*
  - cria H_{R_0}, ..., $H_{R_{max}}$ e H_{S_0}, ..., $H_{s_{max}}$
  - para cada partição i ($0 \leq i \leq max$)
    - constrói um índice *hash* na memória em H_{r_i}
    - adiciona os registros em H_{s_i} ao índice somente se eles não estiverem presentes
    - adiciona os registros do índice ao resultado

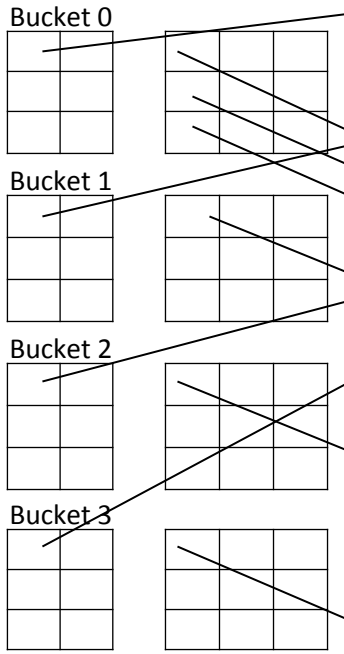


Função *Hash*


R ∪ S
Bucket 0
Bucket 1
Bucket 2
Bucket 3
Departamento
id_Depto Nome Id_ger
4 RH 8 H_{R0}
3 Gerencia 1 H_{R1}
2 TI 7 H_{R3}
1 Produção 2 H_{R2}
Funcionario
id_func Nome Depto Salario Data_Admiss
12 Ivone 4 3000 10/02/2002 H_{S0}
4 Kelly 4 3000 11/02/2002
8 Luana 4 3000 12/02/2002
5 Eleonor 3 2000 06/02/2002 H_{S1}
9 Francisca 3 2000 07/02/2002
1 Otávio 3 4000 16/02/2002
13 Odacir 3 4000 17/02/2002
3 Gilson 2 2500 08/02/2002 H_{S3}
7 Heitor 2 2500 09/02/2002
11 Nair 2 3000 15/02/2002
15 Mario 2 1500 12/01/2002
2 Antonio 1 1500 02/02/2002 H_{S2}
6 Bruno 1 1500 03/02/2002
10 Cesar 1 1500 04/02/2002
14 Dalva 1 1500 05/02/2002
18 Maria 1 1500 13/02/2002

Função *Hash*

R ∪ S



Departamento		
id_Depto	Nome	Id_ger
4	RH	**8**
3	Gerencia	**1**
2	TI	**7**
1	Produção	**2**

H_{R0} H_{R1} H_{R3} H_{R2}

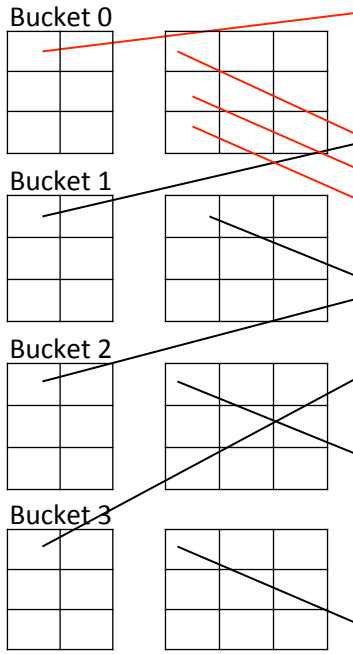
Funcionario				
id_func	Nome	Depto	Salario	Data_Admiss
12	Ivone	4	3000	10/02/2002
4	Kelly	4	3000	11/02/2002
8	Luana	4	3000	12/02/2002
5	Eleonor	3	2000	06/02/2002
9	Francisca	3	2000	07/02/2002
1	Otávio	3	4000	16/02/2002
13	Odacir	3	4000	17/02/2002
3	Gilson	2	2500	08/02/2002
7	Heitor	2	2500	09/02/2002
11	Nair	2	3000	15/02/2002
15	Mario	2	1500	12/01/2002
2	Antonio	1	1500	02/02/2002
6	Bruno	1	1500	03/02/2002
10	Cesar	1	1500	04/02/2002
14	Dalva	1	1500	05/02/2002
18	Maria	1	1500	13/02/2002

H_{S0} H_{S1} H_{S3} H_{S2}

SELECT id_ger **FROM** DEPARTAMENTO
WHERE id_depto = 4
UNION
SELECT id_func **FROM** FUNCIONARIO
WHERE depto = 4

Função *Hash*

R ∪ S



Departamento		
id_Depto	Nome	Id_ger
4	RH	**8**
3	Gerencia	**1**
2	TI	**7**
1	Produção	**2**

H_{R0} H_{R1} H_{R3} H_{R2}

Funcionario				
id_func	Nome	Depto	Salario	Data_Admiss
12	Ivone	4	3000	10/02/2002
4	Kelly	4	3000	11/02/2002
8	Luana	4	3000	12/02/2002
5	Eleonor	3	2000	06/02/2002
9	Francisca	3	2000	07/02/2002
1	Otávio	3	4000	16/02/2002
13	Odacir	3	4000	17/02/2002
3	Gilson	2	2500	08/02/2002
7	Heitor	2	2500	09/02/2002
11	Nair	2	3000	15/02/2002
15	Mario	2	1500	12/01/2002
2	Antonio	1	1500	02/02/2002
6	Bruno	1	1500	03/02/2002
10	Cesar	1	1500	04/02/2002
14	Dalva	1	1500	05/02/2002
18	Maria	1	1500	13/02/2002

H_{S0} H_{S1} H_{S3} H_{S2}

SELECT id_ger **FROM** DEPARTAMENTO
WHERE id_depto = 4
UNION
SELECT id_func **FROM** FUNCIONARIO
WHERE depto = 4

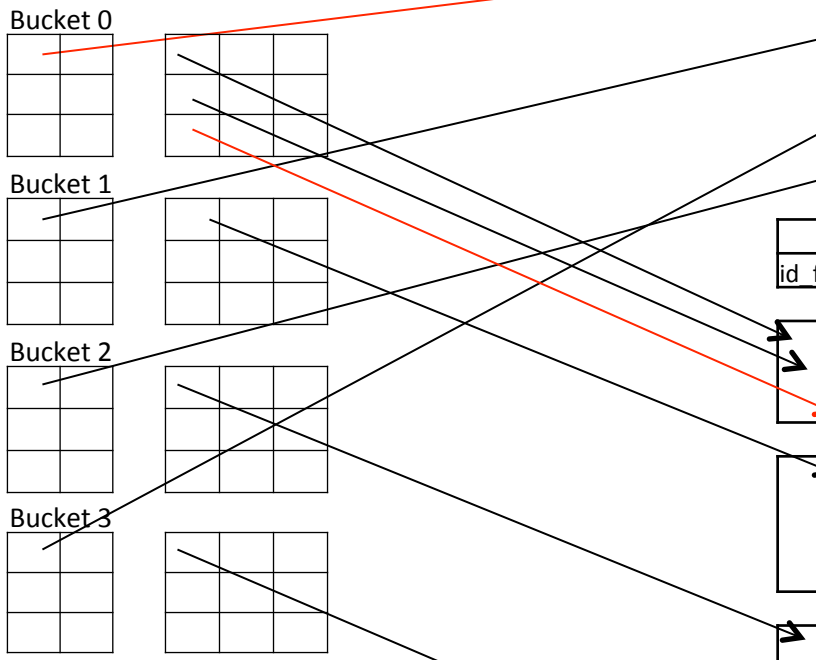
Função *Hash*

- Passos
  - particiona cada relação usando a mesma função *hash*
  - cria H_{R_0}, ..., $H_{R_{max}}$ e H_{S_0}, ..., $H_{s_{max}}$
  - para cada partição i ($0 \leq i \leq max$)
    - constrói um índice *hash* na memória em H_{r_i}
    - para cada registro em H_{s_i}
      - testa o índice
      - adiciona o registro ao resultado somente se ele já estiver presente no índice

$R \cap S$

Função *Hash*

R ∩ S



Departamento		
id_Depto	Nome	Id_ger
4	RH	**8**
3	Gerencia	**1**
2	TI	**7**
1	Produção	**2**

H_{R0} H_{R1} H_{R3} H_{R2}

Funcionario				
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

SELECT id_ger **FROM** DEPARTAMENTO
WHERE id_depto = 4
INTERSECT
SELECT id_func **FROM** FUNCIONARIO
WHERE depto = 4

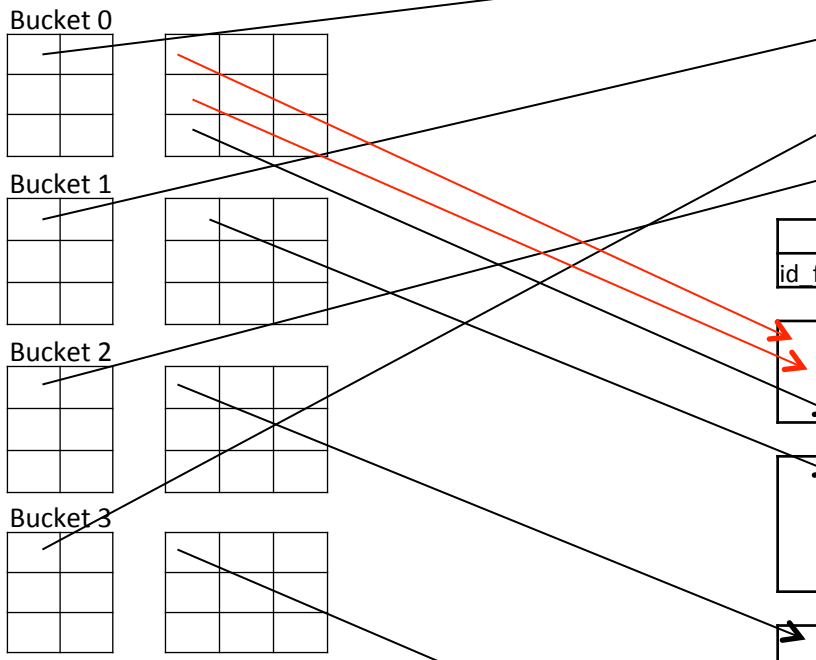
Função *Hash*

- Passos
  - particiona cada relação usando a mesma função *hash*
  - cria H_{R_0}, ..., $H_{R_{max}}$ e H_{S_0}, ..., $H_{s_{max}}$
  - para cada partição i ($0 \leq i \leq max$)
    - constrói um índice *hash* na memória em H_{r_i}
    - para cada registro em H_{s_i}
      - testa o índice
      - remova o registro do índice se ele já estiver presente
    - adiciona os registros do índice ao resultado

R – S

Função *Hash*

R - S



Departamento		
id_Depto	Nome	Id_ger
4	RH	8
3	Gerencia	1
2	TI	7
1	Produção	2

Funcionario				
id_func	Nome	Depto	Salario	Data_Admiss	
12	Ivone	4	3000	10/02/2002	H_{S0}
4	Kelly	4	3000	11/02/2002	
8	Luana	4	3000	12/02/2002	
5	Eleonor	3	2000	06/02/2002	H_{S1}
9	Francisca	3	2000	07/02/2002	
1	Otávio	3	4000	16/02/2002	
13	Odacir	3	4000	17/02/2002	
3	Gilson	2	2500	08/02/2002	H_{S3}
7	Heitor	2	2500	09/02/2002	
11	Nair	2	3000	15/02/2002	
15	Mario	2	1500	12/01/2002	
2	Antonio	1	1500	02/02/2002	H_{S2}
6	Bruno	1	1500	03/02/2002	
10	Cesar	1	1500	04/02/2002	
14	Dalva	1	1500	05/02/2002	
18	Maria	1	1500	13/02/2002	

SELECT id_func **FROM** FUNCIONARIO
WHERE depto = 4
MINUS
SELECT id_ger **FROM** DEPARTAMENTO
WHERE id_depto = 4

Junção

$$R \bowtie_{condição\_junção} S$$

- Concatena registros relacionados de R e S em registros únicos
- Métodos
  - junção de laço aninhado
    - simples, de blocos, indexada
  - *sort-merge* junção
  - *hash* junção

Seletividade de Junção

- Razão entre o número de registros do arquivo resultante da junção e o número de registros do produto cartesiano

$$js = \frac{|(R \bowtie_c S)|}{|(R \times S)|} = \frac{|(R \bowtie_c S)|}{|R| * |S|} \Rightarrow |(R \bowtie_c S)| = js * |R| * |S|$$

  - $|$relação$|$: número de registros da relação

Seletividade de Junção

- Particularidades
  - js = 1 $\Rightarrow$ produto cartesiano
  - js = 0 $\Rightarrow$ nenhum registro satisfaz *c*
  - $0 \leq$ js $\leq 1 \Rightarrow$ alguns registros satisfazem *c*
- *c*: R.A = S.B
  - js $\leq (1/|R|) \Rightarrow$ A é chave primária de R
  - js $\leq (1/|S|) \Rightarrow$ B é chave primária de S

Junção de Laço Aninhado

para cada registro t_R em R faça
    para cada registro t_S em S faça
        se o par (t_R, t_S) satisfaz à condição de junção
            adicione $t_R \bowtie t_S$ ao resultado
    fim_para
fim_para

- R: relação externa
- S: relação interna

Junção de Laço Aninhado

$$C_{laço\_simples} = b_R + (r_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$$

- b: número de blocos que contêm os registros
- r: número de registros
- bfr_{RS}: fator de bloco de disco (arquivo resultante)
- $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

$$C_{laço\_simples} = b_R + b_S + ((js * |R| * |S|)/bfr_{RS})$$

- melhor caso: pelo menos S cabe na memória

Junção de Laço Aninhado

- Vantagens
  - não requer o uso de índices
  - pode ser usado para qualquer condição de junção
- Desvantagem
  - é muito caro, pois examina todos os pares de registros nas duas relações

Junção de Laço Aninhado de Blocos

para cada bloco B_R em R faça
    para cada bloco B_s em S faça
        para cada registro t_R em B_R faça
            para cada registro t_S em B_S faça
                se o par (t_R, t_S) satisfaz à condição de junção
                    adicione $t_R \bowtie t_S$ ao resultado
            fim_para
        fim_para
    fim_para
fim_para

as relações são processadas em termos de blocos, ao invés de registros

Junção de Laço Aninhado de Blocos

$$C_{laço\_blocos} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$$

- b: número de blocos que contêm os registros
- bfr_{RS}: fator de bloco de disco (arquivo resultante)
- $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Junção de Laço Aninhado de Blocos

- Vantagem
  - diminui a quantidade de acessos a blocos de disco, quando comparado com a junção de laço aninhado
- Desvantagem
  - é muito caro, pois examina todos os pares de registros nas duas relações

Melhorias

- Terminar o laço interno na primeira correspondência encontrada, caso os atributos de junção formem uma chave na relação interna
- Varrer o laço interno de maneira alternada para frente e para trás
- Usar o maior tamanho que couber na memória, ao invés de blocos de disco como unidade de bloco

Junção de Laço Aninhado Indexada

- Característica
  - índice no atributo de junção do laço interno
- Melhoria

  para cada registro t_R na relação externa R

    selecione os registros de S que satisfaçam à condição de junção usando o índice
- Pode ser utilizada tanto com índices já existentes, quanto com índices temporários

Junção de Laço Aninhado Indexada

- Índice secundário

$$C_{laço\_sec} = b_R + (|R| * (x_B + s_B)) + ((js * |R| * |S|)/bfr_{RS})$$

  - b: número de blocos que contêm os registros
  - $|R|$: número de registros de R
  - x: número de níveis do índice
  - s: cardinalidade de seleção
  - bfr_{RS}: fator de bloco de disco (arquivo resultante)
  - $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Junção de Laço Aninhado Indexada

- Índice de cluster

$$C_{laço\_cluster} = b_R + (|R| * (x_B + (s_B/bfr_B))) + ((js * |R| * |S|)/bfr_{RS})$$

  - b: número de blocos que contêm os registros
  - $|R|$: número de registros de R
  - x: número de níveis do índice
  - s: cardinalidade de seleção
  - bfr: fator de bloco de disco
  - $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Junção de Laço Aninhado Indexada

- Índice primário

$$C_{laço\_prim} = b_R + (|R| * (x_B + 1)) + ((js * |R| * |S|)/bfr_{RS})$$

  - b: número de blocos que contêm os registros
  - $|R|$: número de registros de R
  - x: número de níveis do índice
  - bfr_{RS}: fator de bloco de disco (arquivo resultante)
  - $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Junção de Laço Aninhado Indexada

- Índice *hash*

$$C_{laço\_hash} = b_R + (|R| * h) + ((js * |R| * |S|)/bfr_{RS})$$

  - b: número de blocos que contêm os registros
  - $|R|$: número de registros de R
  - h: número médio de acessos a blocos para recuperar um registro com o índice *hash*
  - bfr_{RS}: fator de bloco de disco (arquivo resultante)
  - $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Sort-Merge Junção

- Fase de ordenação (i.e., *sort*)
  - ordena fisicamente cada relação, caso a relação não esteja ordenada pelos valores do seu atributo de junção
- Fase de concatenação (i.e., *merge*)
  - percorre cada relação concorrentemente na ordem dos atributos de junção
  - concatena os registros que satisfazem à condição de junção

Algoritmo: $R \bowtie_{A = B} S$

// fase 1 – fase de ordenação

ordene os registros de R pelo atributo A

ordene os registros de S pelo atributo B

ordenação externa

// fase 2 – fase de concatenação

$p_R \leftarrow$ endereço do primeiro registro de R;

$p_S \leftarrow$ endereço do primeiro registro de S;

enquanto ($p_s \neq$ nulo) e ($p_R \neq$ nulo) faça {

  $t_s \leftarrow$ registro para o qual p_s aponta

  $S_s \leftarrow \{t_s\}$

  configure p_s para apontar para o próximo registro de S

  acabou $\leftarrow$ falso

Algoritmo: $R \bowtie_{A = B} S$

enquanto (não acabou) e ($p_s \neq$ nulo) faça {
  t_s' ← registro para o qual p_s aponta
  se ($t_s'[B] = t_s[B]$)
    então {
      $S_s = S_s \cup \{t_s'\}$
      configure p_s para apontar para o próximo registro de S
    }
    senão acabou ← verdadeiro
}
t_R' ← registro para o qual
  p_R aponta

S_S armazena os registros de S que têm o mesmo valor no atributo de junção

Algoritmo: $R \bowtie_{A = B} S$

enquanto ($p_R \neq$ nulo) e $t_R[A] < t_s[B]$) faça {
  configure p_R para apontar para o próximo registro de R
  $t_R \leftarrow$ registro para o qual p_R aponta
}

percorre os registros de R

enquanto ($p_R \neq$ nulo) e $t_R[A] = t_s[B]$) faça {
  para cada t_s em S_s faça
    adicione $t_R \bowtie t_S$ ao resultado
  configure p_R para apontar para o próximo registro de R
  $t_R \leftarrow$ registro para o qual p_R aponta
}
}

concatena os registros de R e S

Custo do *Sort-Merge* Junção

$$C_{sort\_merge} = C_{ordenaR} + C_{ordenaS} + C_{concat}$$

$$C_{ordena} = 2 * b * (1 + \log_2 b)$$

$$C_{concat} = b_R + b_S + ((js * |R| * |S|)/ bfr_{RS})$$

- b: número de blocos que contêm o registro
- $|$relação$|$: número de registros da relação
- bfr_{RS}: fator de bloco de disco (arquivo resultante)
- $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

Hash Junção

- Divide os registros de cada relação em partições que têm o mesmo valor *hash* nos atributos de junção
- Vantagens
  - algoritmo eficiente
  - aplicado a relações ordenadas ou não
- Desvantagem
  - necessidade de uma função *hash* que tenha as propriedades randômica e de uniformidade

Hash Junção

- Exemplo ($R \bowtie_{A = B} S$)
  - h: função *hash* que mapeia os valores dos atributos de junção para {0, 1, ... , max}
  - H_{R_0}, ..., $H_{R_{max}}$: partições dos registros de R
    - cada registro $t_R \in R$ é colocado na partição H_{R_i}, $i = h(t_R[A])$
  - H_{S_0}, ..., $H_{S_{max}}$: partições dos registros de S
    - cada registro $t_S \in S$ é colocado na partição H_{S_i}, $i = h(t_S[B])$

Hash Junção

- Consideração

  se $t_R[A] = t_S[B]$

  então $t_R \rightarrow H_{R_i}$

  $t_S \rightarrow H_{S_i}$

- Significado
  - os registros de R em H_{R_i} precisam ser comparados apenas com os registros de S em H_{S_i}

a mesma função *hash* aplicada ao mesmo valor retorna a mesma partição i

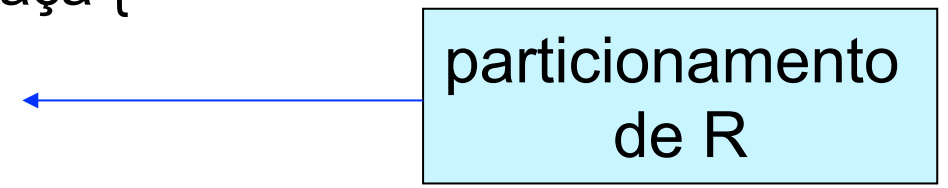
Algoritmo: $R \bowtie_{A = B} S$

para cada registro t_R em R faça {

  $i \leftarrow h(t_R, [A])$

  $H_{R_i} \leftarrow H_{R_i} \cup \{t_R\}$

}

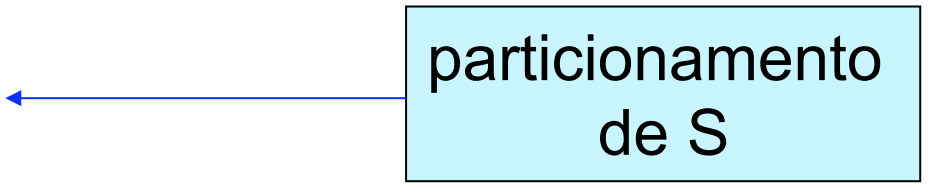


para cada registro t_S em S faça {

  $i \leftarrow h(t_s, [A])$

  $H_{s_i} \leftarrow H_{s_i} \cup \{t_s\}$

}



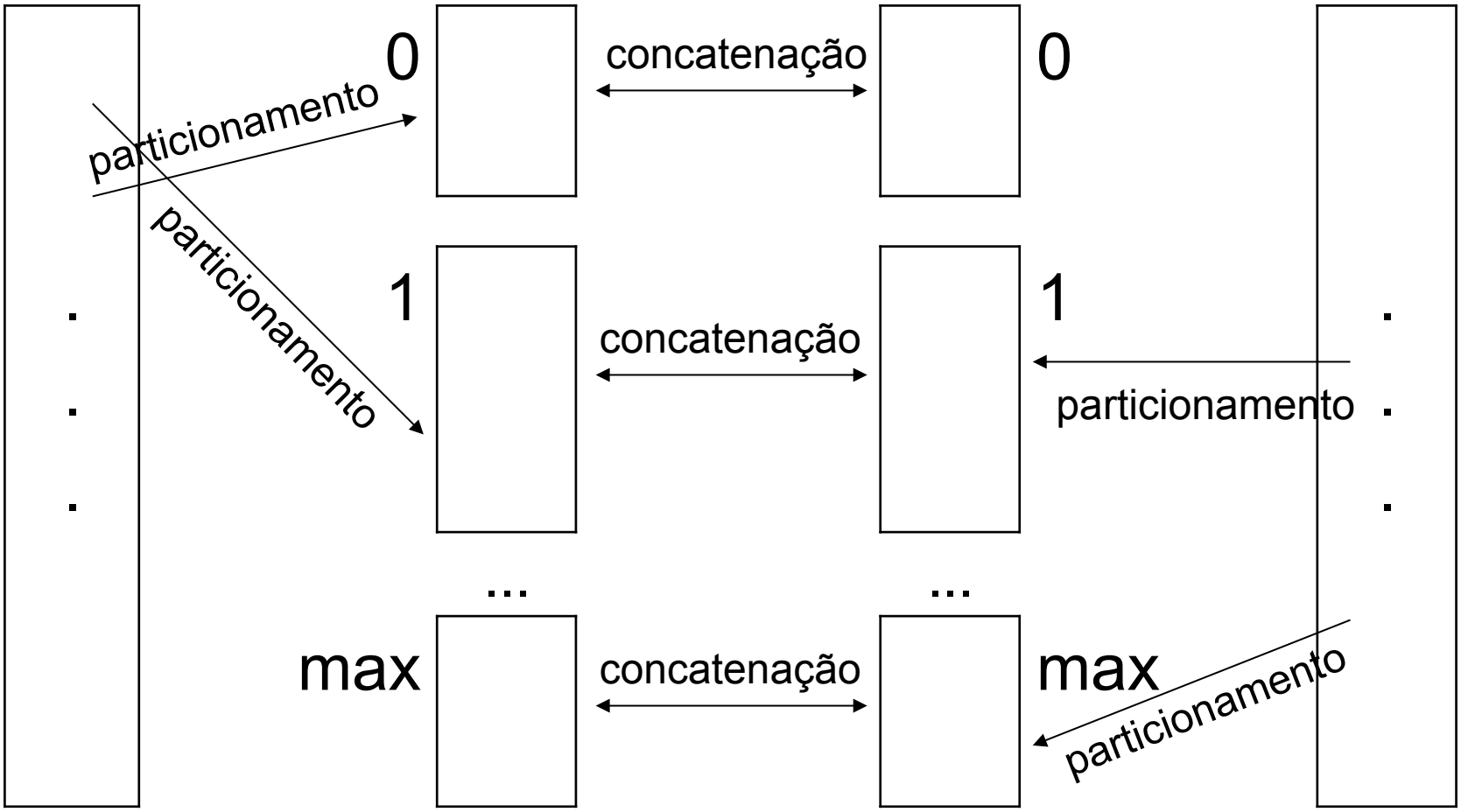
Algoritmo: $R \bowtie_{A = B} S$

para i = 0 até max faça {
  leia H_{s_i} e construa um índice *hash* na memória para ele
  para cada registro t_R em H_{R_i} faça {
    verifique o índice *hash* H_{s_i} para localizar todas os registros t_s
      tal que $t_R[A] = t_S[B]$
    para cada registro t_s coincidente em H_{s_i} faça
      adicione $t_R \bowtie t_S$ ao resultado
  }
}

junção dos registros em cada partição

- S: entrada da construção
- R: entrada de teste

Exemplo


0
concatenação
0
particionamento
particionamento
1
concatenação
1
particionamento
...
...
max
concatenação
max
particionamento
R
partições de R
partições de S
S

Custo do *Hash* Junção

$$C_{hash\_junção} = 3 * (b_R + b_S) + ((js * |R| * |S|)/bfr_{RS})$$

- b: número de blocos que contêm os registros
- $|$relação$|$: número de registros da relação
- bfr_{RS}: fator de bloco de disco (arquivo resultante)
- $(js * |R| * |S|)/bfr_{RS}$: custo de escrever o arquivo resultante em disco

- Cada registro
  - lido uma vez e escrito uma vez no particionamento
  - lido uma vez na concatenação

Considerações

- Valor max
  - grande o suficiente para permitir que os registros na partição H_{s_i} e o índice *hash* caibam na memória
- Partições de teste
  - não precisam caber na memória
- Relação de construção
  - menor número de registros

Algoritmos de Junção

- Fatores determinantes de desempenho
  - espaço disponível em memória principal
  - tamanho da relação externa em uma junção de laço aninhado
    - menor número de blocos
  - fator de seleção de junção
    - porcentagem de registros de uma relação a ser concatenado com os registros da outra relação

Relações

cliente (nro_cli, nome_cli, end_cli, saldo, cod_vend)

vendedor (cod_vend, nome_vend)

pedido (nro_ped, data, nro_cliente)

pedido_peça (nro_ped, nro_peça)

peça (nro_peça, descrição_peça)

Relação Cliente

- Número de registros (r) = 10.000
- Número de blocos de disco (b) = 2.000
- Fator de bloco de disco (bfr) = 5
- Índice primário em nro_cli
  - número de níveis (x) = 4
  - número médio de registros que satisfazem à condição de igualdade (s) = 1

Relação Cliente

- Índice secundário em cod_vend
  - número de níveis (x) = 2
  - número de blocos no nível de folha (b_{I1}) = 4
  - número de valores distintos (d) = 125
  - número médio de registros que satisfazem à condição de igualdade (s) = 80
- Índice secundário em saldo
  - número de níveis (x) = 3
  - número de blocos no nível de folha (b_{I1}) = 4

Relação Vendedor

- Número de registros (r) = 125
- Número de blocos de disco (b) = 13
- Índice primário em cod_vend
  - número de níveis (x) = 1
  - número médio de registros que satisfazem à condição de igualdade (s) = 1

Demais Estatísticas

- Seletividade de junção
  - $js = (1 / |vendedor|) = 1/125 = 0,008$
- Fator de bloco de disco do arquivo resultante
  - $bfr_{ClienteVendedor} = 4$
- Custo de escrever em disco o arquivo resultante
  - $(js * r_{cliente} * r_{vendedor})/bfr_{ClienteVendedor} = 2.500$

Consulta

cliente $\bowtie_{cliente.cod\_vend = vendedor.cod\_vend}$ vendedor

- Junção de laço aninhado de blocos
  - cliente: relação externa

$$C_{laço\_blocos} = b_{cliente} + (b_{cliente} * b_{vendedor}) + 2.500$$

$$= 2.000 + (2.000 * 13) + 2.500$$

$$= 30.500$$

Consulta

cliente $\bowtie_{cliente.cod\_vend = vendedor.cod\_vend}$ vendedor

- Junção de laço aninhado de blocos
  - vendedor: relação externa

$$C_{laço\_blocos} = b_{vendedor} + (b_{vendedor} * b_{cliente}) + 2.500$$

$$= 13 + (13 * 2.000) + 2.500$$

$$= 28.513$$

Consulta

cliente $\bowtie_{cliente.cod\_vend = vendedor.cod\_vend}$ vendedor

- Junção de laço aninhado indexado
  - cliente: relação externa

$$C_{laço\_sec} = b_{cliente} + (r_{cliente} * (x_{cod\_vend} + 1)) + 2.500$$

$$= 2.000 + (10.000 * (1 + 1)) + 2.500$$

$$= 24.500$$

índice primário em cod_vend de vendedor

Consulta

cliente $\bowtie_{cliente.cod\_vend = vendedor.cod\_vend}$ vendedor

- Junção de laço aninhado indexado
  - vendedor: relação externa

$C_{laço\_sec} = b_{vendedor} + (r_{vendedor} * (x_{cod\_vend} + s_{cod\_vend})) + 2.500$

$= 13 + (125 * (2 + 80)) + 2.500$

$= 12.763$

índice secundário em cod_vend de cliente

Consulta

cliente $\bowtie_{cliente.cod\_vend = vendedor.cod\_vend}$ vendedor

- *Sort-merge* junção
  - $C_{ordCli} = 2 * b_{cliente} * (1 + \log_2 b_{cliente}) = 48.000$
  - $C_{ordVend} = 2 * b_{vendedor} * (1 + \log_2 b_{vendedor}) = 130$
  - $C_{concat} = b_{cliente} + b_{vendedor} + 2.500 = 4.513$

$C_{sort\_merge} = C_{ordCli} + C_{ordVend} + C_{concat} = 52.643$

Consulta

$$\text{cliente} \bowtie_{\textit{cliente.cod\_vend = vendedor.cod\_vend}} \text{vendedor}$$

- *Hash* junção

$$C_{hash\_junção} = 3 * (b_{cliente} + b_{vendedor}) + 2.500$$

$$= 3 * (2.000 + 13) + 2.500$$

$$= 8.539$$