Compilação, Otimização e Execução de Consultas

Cristina Dutra de Aguiar Ciferri

Decomposição de Consultas

- Bloco de consulta
  - é uma unidade básica que pode ser traduzida em operadores algébricos e otimizada
  - contém
    - uma única expressão SELECT-FROM-WHERE
    - cláusulas GROUP BY e HAVING, caso necessário
- Consultas SQL aninhadas
  - identificadas como blocos de consulta distintos

Visão Geral

Para todo localizado em 'Stafford' listar o número do projeto, o número do departamento responsável, o

Análise de alternativas de definição de estratégias de acesso
- escolha de algoritmos para implementação de operações
- existência de índices
- estimativas sobre os dados
(tamanho de tabelas, seletividade, ...)

-cláusulas SQL e nomes válidos.

- **Análise sintática**
  - validação da gramática.
- **Análise semântica**
  - nomes usados de acordo com a estrutura do esquema.
- **Conversão para uma árvore algébrica da consulta**

Decomposição de Consultas

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

```
SELECT nome_cli, end_cli
FROM cliente
WHERE  saldo > ( SELECT max (saldo)
                 FROM cliente
                 WHERE cod_vend = 5);
```

```
SELECT  nome_cli, end_cli
FROM cliente
WHERE saldo > c
```

```
SELECT  max (saldo)
FROM cliente
WHERE cod_vend = 5
```

$\pi_{nome\_cli,\ end\_cli}\ (\sigma_{saldo>C}\ (cliente))$

$\xi_{max(saldo)}\ (\sigma_{cod\_vend\ =\ 5}\ (cliente))$

Custos da consulta

- **Acessos ao disco;**

✓Em grandes BD, o custo para acessar dados no disco é muito mais importante, pois o acesso ao disco é muito mais lento em comparação as operações na memória.

- **Tempo de CPU para executá-la;**

✓As velocidades de CPU melhoraram muito mais rapidamente do que as velocidades do disco.

✓Difícil de estimular.

Ordenação

- Amplamente utilizada no processamento de consultas
  - oferece suporte para a cláusula ORDER BY
  - classifica dados a serem combinados nas operações de junção, união, intersecção
  - elimina duplicatas na operação de projeção
- Pode ser evitada caso exista um índice que permita acesso ordenado aos registros

Ordenação Externa

- Funcionalidade
  - classifica registros de arquivos armazenados em disco
- Restrição
  - tamanho do arquivo é maior do que o tamanho da memória principal disponível
- Algoritmo típico
  - *sort-merge* externo

Ordenação externa

- cláusula ORDER BY
- arquivos grandes
- FASE 1: Quebra-Ordenação interna (sort)
- gera “runs” (fragmentos ordenados do arquivo original)
- ordena runs
- grava runs
- *b = numero de blocos do arquivo*
- *nb = espaço disponível de buffer*
- *nr = número de runs iniciais*
- *nr = b/nb*
- Complexidade ordem = 2 * b (leitura e escrita)

Sort-Merge Externo

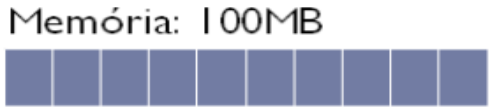
- Fase 1
  - cria subarquivos ordenados (i.e., *runs*) a partir do arquivo original
- Fase 2
  - combina os subarquivos ordenados em subarquivos ordenados maiores até que o arquivo completo esteja ordenado

Ordenação externa

A ordenação em memória externa é utilizada para grandes arquivos que não cabem na memória principal;

- Por exemplo, a ordenação de 900 megabytes de dados com apenas 100 megabytes de memória RAM, usando o mergesort.

Representação da memória e do arquivo a ordenar.

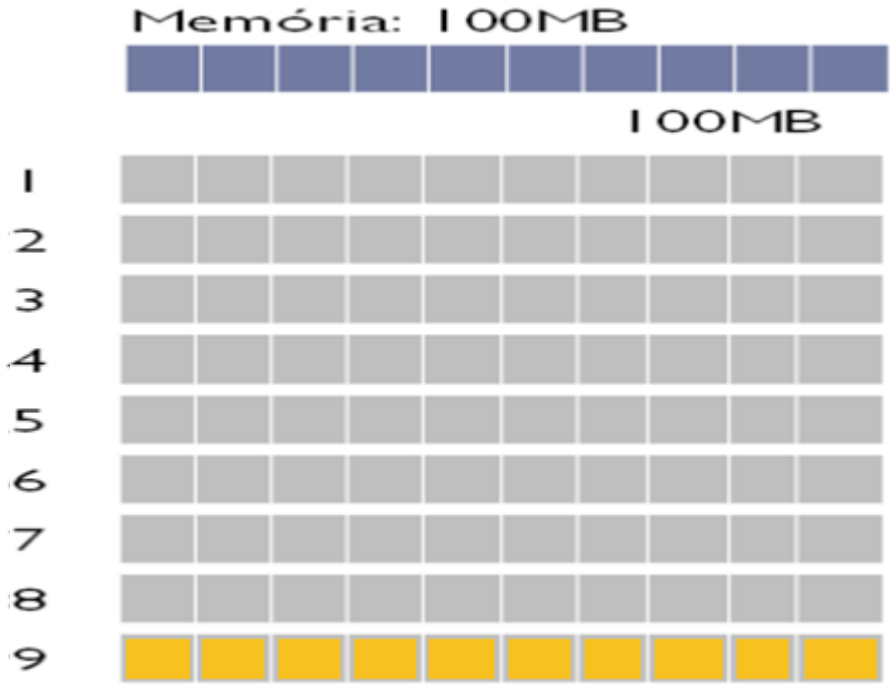


n_b ->100 - Espaço dispinível em disco;

b ->900- tamanho do arquivo original em blocos

1. Leia os 100MB de dados que estão na memória principal, ordene-os.
2. Escolha os 99 menores valores e escreva-os em um arquivo temporário.
3. Repita os passos 1 e 2 até que todos os dados do arquivo original tenham sido processados.

O exemplo a seguir mostra a divisão do arquivo em tamanhos de 100MB, o tamanho da memória para não ocorrer acesso a disco. Neste caso, é lido o vetor 1.



$$\frac{b}{n_b}$$

$$\frac{900}{100} = 9$$

Arquivo ordenado

4. Realiza a função merge nos 9 buffers de entrada para fundir e armazenar o resultado no buffer de saída. Quando estiver cheio, escreva-o no final do arquivo ordenado.

Algoritmo

// inicializações

i ← 1

j ← b // tamanho do arquivo original em blocos

k ← n_b // tamanho do buffer em blocos

m ← ⌈(j/k)⌉ // número de runs iniciais

merge-sort externo requer espaço disponível nos buffers da memória principal

número de subarquivos iniciais gerados, de acordo com o tamanho do arquivo original e o tamanho do buffer de memória

Algoritmo

```
// fase 1 – fase de ordenação
enquanto ( i ≤ m ) faça {
   leia os próximos k blocos do arquivo no buffer ou
        os blocos remanescentes caso não hajam k blocos
   ordene os registros que estão no buffer usando algum
        algoritmo de ordenação interna
   escreva os dados ordenados em um subarquivo temporário
   i ← i + 1
}
```

- resultado da fase 1
  - m subarquivos ordenados armazenados em disco

Algoritmo

// fase 2 – fase de combinação dos subarquivos ordenados

i ← 1

p ← $\lceil \log_{k-1} m \rceil$ // número de passos

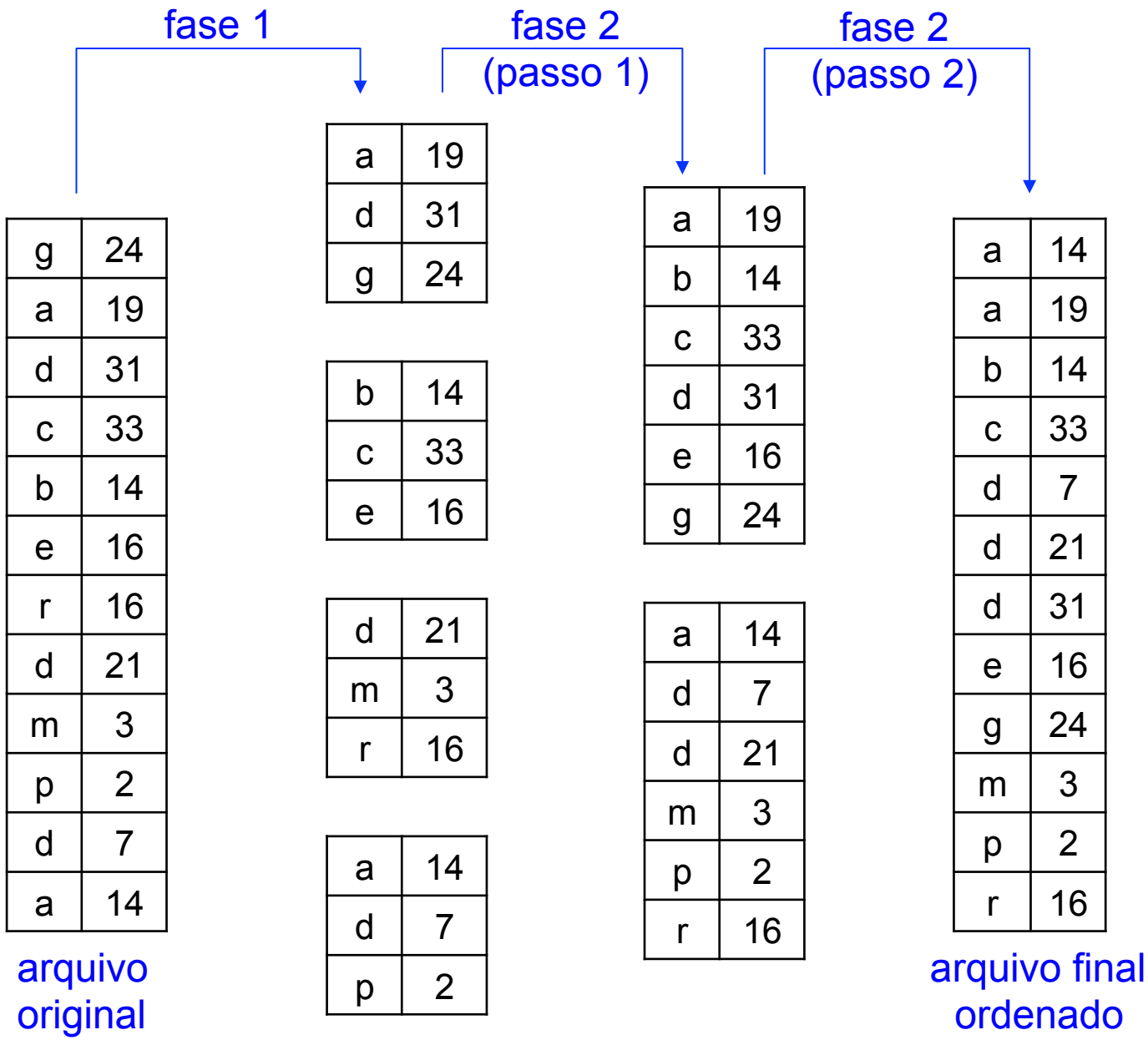
j ← m

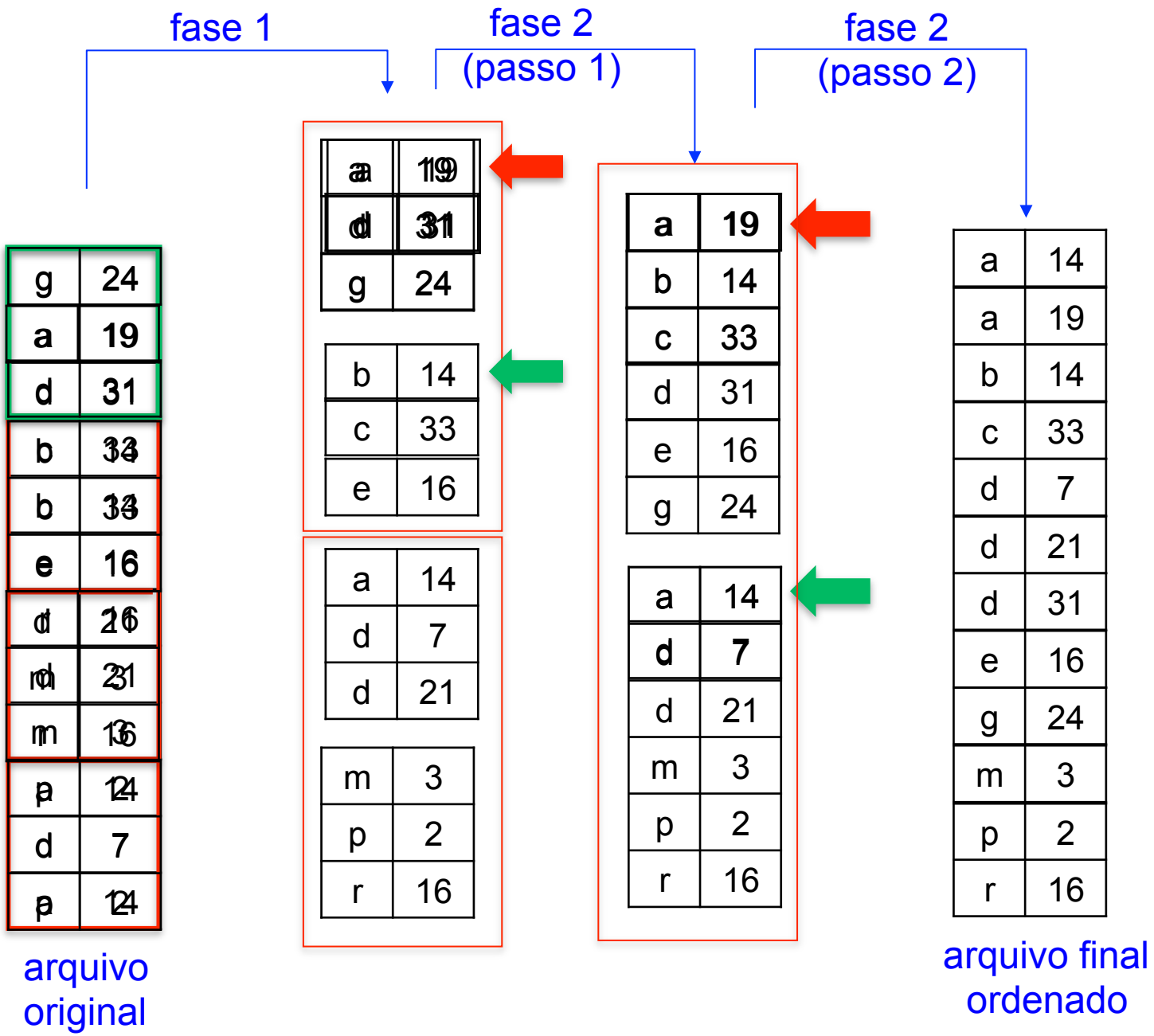
- *merge-sort* externo
  - combina os subarquivos ordenados em subarquivos ordenados maiores em um ou mais passos
- grau de combinação
  - número de subarquivos ordenados que podem ser combinados em cada passo

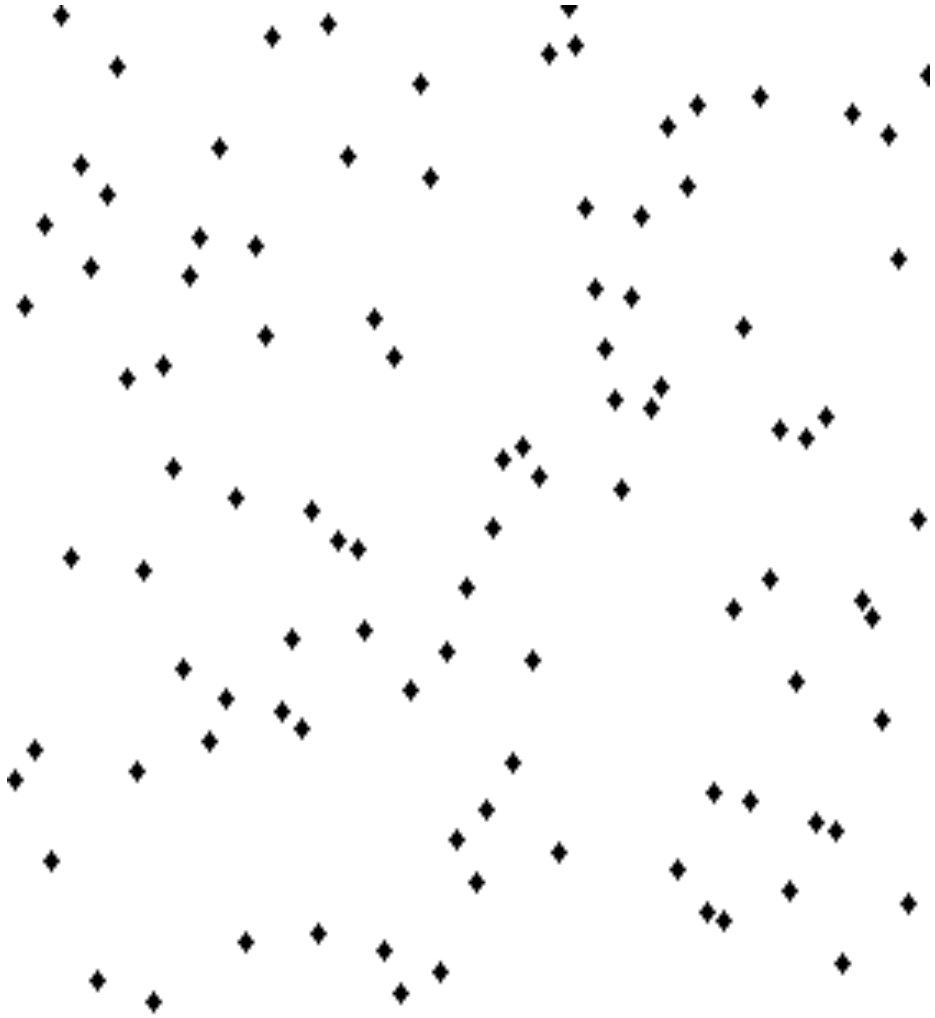
Algoritmo

```
enquanto ( i ≤ p ) faça {
   n ← 1
   q ← ⌈(j/(k-1))⌉        // número de subarquivos a serem escritos no passo
   enquanto ( n ≤ q ) faça {
      leia  os próximos k-1 subarquivos ordenados remanescentes
            (do passo anterior) um bloco por vez
      combine os subarquivos ordenados
      escreva os dados ordenados em um subarquivo temporário
   }
   j ← q;
   i ← i + 1
}
```

- resultado da fase 2
  - arquivo original ordenado armazenado em disco


fase 1
fase 2
(passo 1)
fase 2
(passo 2)
g 24
a 19
d 31
c 33
b 14
e 16
r 16
d 21
m 3
p 2
d 7
a 14
arquivo original
a 19
d 31
g 24
b 14
c 33
e 16
d 21
m 3
r 16
a 14
d 7
p 2
a 19
b 14
c 33
d 31
e 16
g 24
a 14
d 7
d 21
m 3
p 2
r 16
a 14
a 19
b 14
c 33
d 7
d 21
d 31
e 16
g 24
m 3
p 2
r 16
arquivo final ordenado


fase 1
fase 2
(passo 1)
fase 2
(passo 2)
arquivo
original
arquivo final
ordenado



Custo do Algoritmo

$$(2 * b) + (2 * (b * \lceil \log_{d_m} b \rceil))$$

- (2 * b)
  - número de acessos a disco na fase 1
- cada bloco do arquivo é acessado duas vezes
  - fase de leitura dos dados para a memória
  - fase de escrita dos dados ordenados no disco

Custo do Algoritmo

$$(2 * b) + (2 * (b * \lceil \log_{d_m} b \rceil)))$$

- $(2 * (b * \lceil \log_{d_m} b \rceil))$
  - número de acessos a disco na fase 2
- cada bloco dos subarquivos é acessado várias vezes, dependendo do grau de combinação
  - fase de leitura dos dados para a memória
  - fase de escrita dos dados ordenados no disco

Custo do Algoritmo

$b\ (2 * \lceil \log_{m-1} b/n_b \rceil + 1\ )$

- Não incluem o custo da escrita do resultado final

$b + b\ (2 * \lceil \log_{m-1} b/n_b \rceil + 1\ )$

- Incluindo o custo da escrita do resultado final

(Silberschatz, Korth, Sudarshan)

Custo do Algoritmo

- Qual é o custo de classificar um arquivo que tem 108 páginas, usando apenas 5 páginas da memória principal?

b = tamanho do arquivo original em blocos

n_b = tamanho do buffer em blocos

m = $\lceil (b/n_b) \rceil$ número de runs iniciais (passos)

$b\ (2 * \lceil \log_{m-1} b/n_b \rceil + 1\ )$ Sem resultado final

$b + b\ (2 * \lceil \log_{m-1} b/n_b \rceil + 1\ )$ resultado final

Seleção

- Algoritmos dependem
  - da existência de índices
  - das condições de seleção
- Métodos para seleção simples
  - varredura de arquivos (i.e., *file scan*)
    - busca linear e binária
  - varredura de índices (i.e., *index scan*)
    - busca baseada em índice primário, secundário e cluster

Busca Linear

- Características
  - recupera cada registro do arquivo
  - verifica se os valores dos atributos satisfazem à condição de seleção
- Vantagem
  - pode ser aplicada a qualquer arquivo
- Desvantagem
  - pode ser ineficiente

Custo da Busca Linear

$$C_{busca\_linear} = b$$

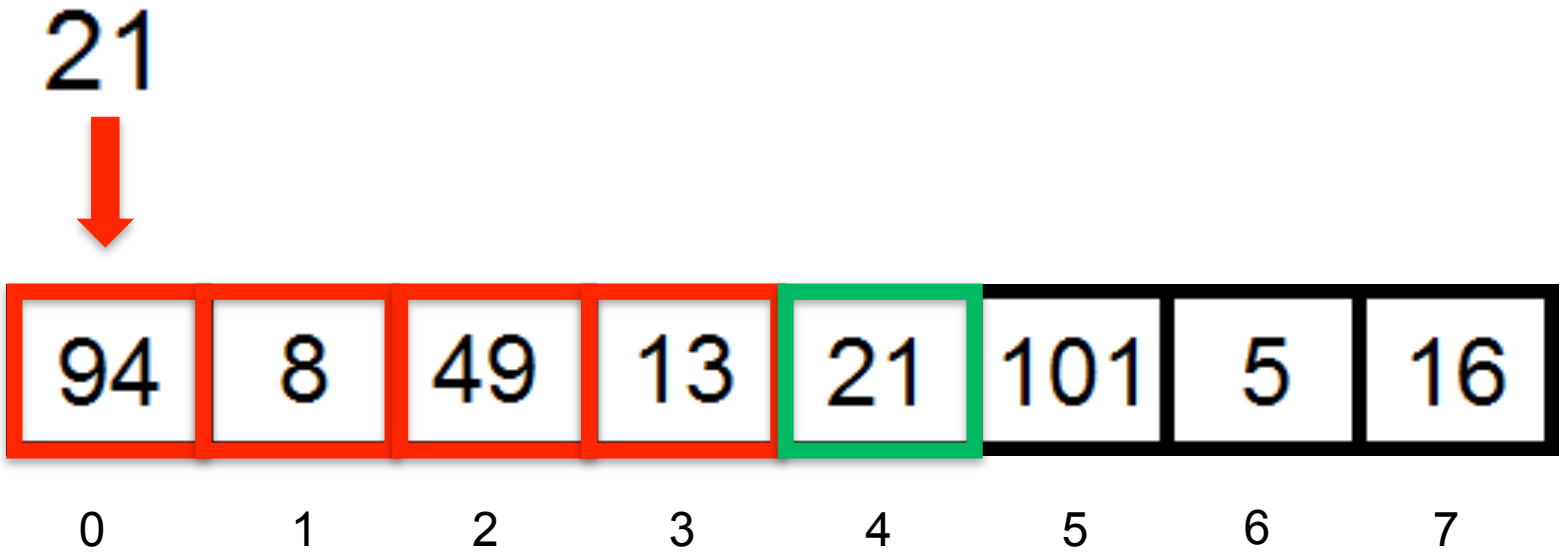
- b: número de blocos que contêm os registros
- todos os blocos são varridos

$$C_{busca\_linear} = (b/2)$$

- igualdade na chave primária
- metade dos blocos é varrido, em média

Exemplo: busca linear

- Encontre o valor 21;



Busca Binária

- Característica
  - recupera registros quando a condição de seleção envolve uma comparação de igualdade no atributo que determina a ordenação do arquivo
- Vantagem
  - mais eficiente do que busca linear
- Desvantagem
  - requer a ordenação do arquivo

Custo da Busca Binária

$$C_{busca\_binária} = \log_2(b) + \lceil s/bfr \rceil - 1$$

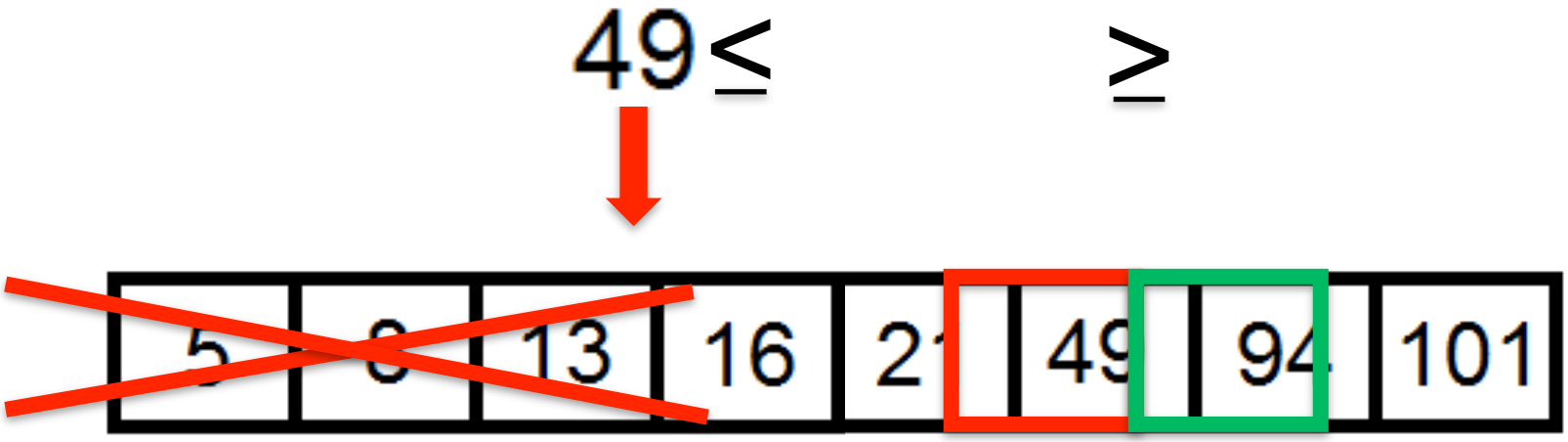
- $\log_2(b)$: custo para localizar o primeiro registro
- $\lceil s/bfr \rceil$: blocos ocupados pelos registros que satisfazem à condição de seleção
- 1: custo para recuperar o primeiro registro

$$C_{busca\_binária} = \log_2(b)$$

- igualdade na chave primária

Exemplo: busca binária

- Encontre o valor 49;



Busca Baseada em Índice

Índice	Condição de Seleção	Campo Indexado	Registros Recuperados
primário	igualdade	chave primária	0 ou 1
	desigualdade	chave primária	0 ou vários
cluster	igualdade	atributo não chave	0 ou vários
secundário	igualdade	UNIQUE atributo não chave	0 ou 1 0 ou vários
	desigualdade	UNIQUE atributo não chave	0 ou vários

Busca Baseada em Índice

- Vantagem
  - mais eficiente do que busca linear e busca binária (em geral)
- Desvantagem
  - necessidade de índice no atributo usado na condição de seleção
    - custo de armazenamento
    - custo de manutenção

Custo com Índice Primário

igualdade

$$C_{prim\_igual} = x + 1$$

- x: número de níveis no índice
- 1: bloco adicional recuperado

desigualdade

$$C_{prim\_des} = x + (b/2)$$

- b/2: estimativa de que somente metade dos registros satisfazem à condição de seleção

Custo com Cluster

$$C_{cluster} = x + \lceil s/bfr \rceil$$

- x: número de níveis no índice
- $\lceil s/bfr \rceil$: quantidade de blocos ocupada pelos registros que satisfazem à condição de seleção

Custo com Índice Secundário

igualdade

$$C_{sec\_igual} = x + s$$

- x: número de níveis no índice
- s: cardinalidade de seleção do atributo indexado
  - s = 1 para atributo UNIQUE

desigualdade

$$C_{sec\_des} = x + (b_{I1}/2) + (r/2)$$

- $b_{I1}/2$: metade dos blocos serão acessados
- r/2: metade dos registros serão acessados

(estimativas)

Seleção

- Métodos para seleção complexa
  - conjuntiva
    - $\sigma_{c_1 \wedge c_2 \wedge ... \wedge c_n}$ (relação)
  - disjuntiva
    - $\sigma_{c_1 \vee c_2 \vee ... \vee c_n}$ (relação)
- Algoritmos dependem
  - da existência de índices

Seleção Conjuntiva – índice único –

- Característica
  - índice definido sobre um único atributo da condição
- Passos
  - recupera os registros para o atributo usando qualquer algoritmo de busca
  - verifica se cada registro recuperado satisfaz às demais condições
- Custo dependente do tipo de índice

Seleção Conjuntiva – índice composto –

- Característica
  - dois ou mais atributos envolvidos em condições de igualdade
  - índice composto definido sobre os atributos combinados
  - Comparações Ligadas com **AND**
- Passo
  - recupera os registros usando o índice composto
- Custo dependente do tipo de índice

Seleção Conjuntiva – intersecção de ponteiros –

- Característica
  - índices com ponteiros de registros definidos nos atributos das condições individuais
- Passos
  - varre cada índice em busca de ponteiros para registros que satisfaçam à condição individual
  - intersecta todos os ponteiros recuperados
  - recupera os registros propriamente ditos

σ(ALUNO) (centro=CTC OR curso=SIN) AND sexo=F AND ia>9,5

Seleção Conjuntiva – intersecção de ponteiros –

- Característica
  - índices com ponteiros de registros definidos sobre somente alguns dos atributos das condições individuais
- Passo adicional
  - verifica se cada registro recuperado satisfaz às demais condições

Seletividade

- Razão entre o número de registros que satisfazem à condição de seleção e o número de registros da relação
- Funcionalidade
  - determina a ordem na qual as condições da seleção conjuntiva devem ser testadas
- Otimizador de consultas
  - escolhe como primeiro atributo a ser buscado o que possui a condição mais seletiva

Seleção Disjuntiva

- Registros que satisfazem à condição disjuntiva são a união dos registros que satisfazem cada condição individual
- Característica
  - índices não definidos sobre todos os atributos da condição
- Passo
  - recupera os registros usando busca linear

Custo = C_{busca_linear}

Seleção Disjuntiva

- Característica
  - índices definidos sobre cada um dos atributos da condição
- Passos
  - recupera os registros para cada condição individual usando o índice específico
  - aplica a operação de união para eliminar duplicatas
- Custo dependente do tipo de índice

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)

Exemplos

Consulta	Busca
$\sigma_{nro\_cli = 4}$ (cliente)	• linear • binária • índice primário; igualdade na chave primária
$\sigma_{nro\_cli > 4}$ (cliente)	• linear • binária • índice primário; desigualdade na chave primária
$\sigma_{cod\_vend = 5}$ (cliente)	• linear • cluster • secundário

Exemplos

Consulta	Busca
$\sigma_{cod\_vend = 5 \wedge saldo > 100}$ (cliente)	• linear • seleção conjuntiva – índice único – intersecção de ponteiros
$\sigma_{nro\_cli = 4 \wedge cod\_vend = 5}$ (cliente)	• linear • seleção conjuntiva – índice único – índice composto – intersecção de ponteiros
$\sigma_{nro\_cli = 4 \vee cod\_vend > 5}$ (cliente)	• linear • seleção disjuntiva

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 (chave primária)
  - 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_{l1}) = 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_{l1}) = 4

Consultas

Consulta	Busca
$\sigma_{nro\_cli = 4}$ (cliente)	• linear • binária • índice primário; igualdade na chave primária

- $C_{busca\_linear} = (b/2) = 2.000/2 = 1.000$
- $C_{busca\_binária} = \log_2(b) = \log_2(2.000) = 11$
- $C_{prim\_igual} = x + 1 = 4 + 1 = 5$

escolha do otimizador de consultas

Consultas

Consulta	Busca
$\sigma_{nro\_cli > 4}$ (cliente)	• linear • índice primário; desigualdade na chave primária

- $C_{busca\_linear} = b = 2.000$
- $C_{prim\_des} = x + (b/2) = 4 + (2.000/2) = 1004$

escolha do otimizador de consultas

Consultas

Consulta	Busca
$\sigma_{cod\_vend = 5}$ (cliente)	• linear • secundário

- $C_{busca\_linear} = b = 2.000$
- $C_{sec\_igual} = x + s = 2 + 80 = 82$

escolha do otimizador de consultas

Consultas

Consulta	Busca
$\sigma_{cod\_vend = 5 \wedge saldo > 100}$ (cliente)	• linear • seleção conjuntiva – índice único

- $C_{busca\_linear} = b = 2.000$
- $C_{sec\_igual}^{cod\_vend} = x + s = 2 + 80 = 82$
- $C_{sec\_des}^{saldo} = x + (b_{I1}/2) + (r/2)$
  $= 3 + (4/2) + (1.000/2)$
  $= 3 + 2 + 500 = 505$

otimizador de consultas busca primeiro por cod_vend = 5

Consultas

Consulta	Busca
$\sigma_{nro\_cli = 4 \wedge cod\_vend = 5}$ (cliente)	• linear • seleção conjuntiva – índice único

- $C_{busca\_linear} = b = 2.000$
- $C_{prim\_igual}^{nro\_cli} = x + 1 = 4 + 1 = 5$
- $C_{sec\_igual}^{cod\_vend} = x + s = 82$

otimizador de consultas busca primeiro por nro_cli = 4

Projeção

$\pi_{lista\_atributos}$ (relação)

- lista_atributos inclui chave primária
  - recupera os registros
  - considera somente os atributos desses registros constantes em lista_atributos

número de registros: mesmo número da relação original

Projeção

$$\pi_{\text{lista\_atributos}} (\text{relação})$$

- lista_atributos não inclui chave primária
  - recupera os registros
  - considera somente os atributos desses registros constantes em lista_atributos
  - ordena os registros
  - elimina as duplicatas

cláusula DISTINCT

Produto Cartesiano

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

- Relação produzida
  - grau: j + k
  - número de registros: n * m
- Otimizador de consultas
  - deve evitar a operação, substituindo-a por outras operações equivalentes

Modelo

conta

nome_agencia	numero_conta	saldo
SAL-1	0001	1200
SAL-1	0002	3000
NOH-1	0003	4500
POA-1	0004	4000
POA-1	0005	1500
NOH-1	0006	200
SAL-2	0007	3750
SAL-2	0008	1800

cliente

nome	rua	cidade
João	Getúlio Vargas	São Leopoldo
Pedro	Getúlio Vargas	São Leopoldo
Francisco	Olavo Bilac	Novo Hamburgo
Maria	João Pessoa	Porto Alegre
Paulo	Cecília Meireles	São Leopoldo
José	João Goulart	Novo Hamburgo
Ana	Assis Brasil	Porto alegre
Beatriz	Floriano Peixoto	Novo Hamburgo

agencia

nome_agencia	cidade_agencia	saldo
NOH-1	Novo Hamburgo	260050
SAL-1	São Leopoldo	455580
POA-1	Porto Alegre	1250369
SAL-2	São Leopoldo	125588

depositante

nome_cliente	numero_conta
João	0001
Pedro	0002
Francisco	0003
Maria	0004
Paulo	0007
José	0006
Ana	0005

devedor

nome_cliente	num_emprestimo
João	E-001
Ana	E-005
Helena	E-008

emprestimo

nome_agencia	num_emprestimo	total
SAL-1	E-001	40000
POA-1	E-005	25400
NOH-1	E-008	5420

Operação Produto Cartesiano: permite combinar informações de duas relações quaisquer. Representado por “r x s”.

(devedor x emprestimo) = (nome_cliente, devedor.num_emprestimo, nome_agencia, emprestimo.num_emprestimo, total)

devedor x emprestimo

nome_cliente	devedor.num_emprestimo	nome_agencia	emprestimo.num_emprestimo	total
João	E-001	SAL-1	E-001	40000
João	E-001	POA-1	E-005	25400
João	E-001	NOH-1	E-008	5420
Ana	E-005	SAL-1	E-001	40000
Ana	E-005	POA-1	E-005	25400
Ana	E-005	NOH-1	E-008	5420
Helena	E-008	SAL-1	E-001	40000
Helena	E-008	POA-1	E-005	25400
Helena	E-008	NOH-1	E-008	5420

Operações sobre Conjuntos

- Operações
  - união
  - intersecção
  - diferença
- Características
  - atuam sobre relações compatíveis
  - eliminam duplicadas da relação resultado

Duas relações são compatíveis quando:
- possuem o mesmo grau
- seus atributos possuem os mesmos domínios
  (os domínios dos i-ésimos atributos de cada relação são os mesmos)

Operações sobre Conjuntos

- Passos
  - ordena cada uma das relações (mesmo atributo)
  - varre e combina os registros ordenados concorrentemente
  - elimina duplicatas mantendo somente um dos registros repetidos na relação resultado

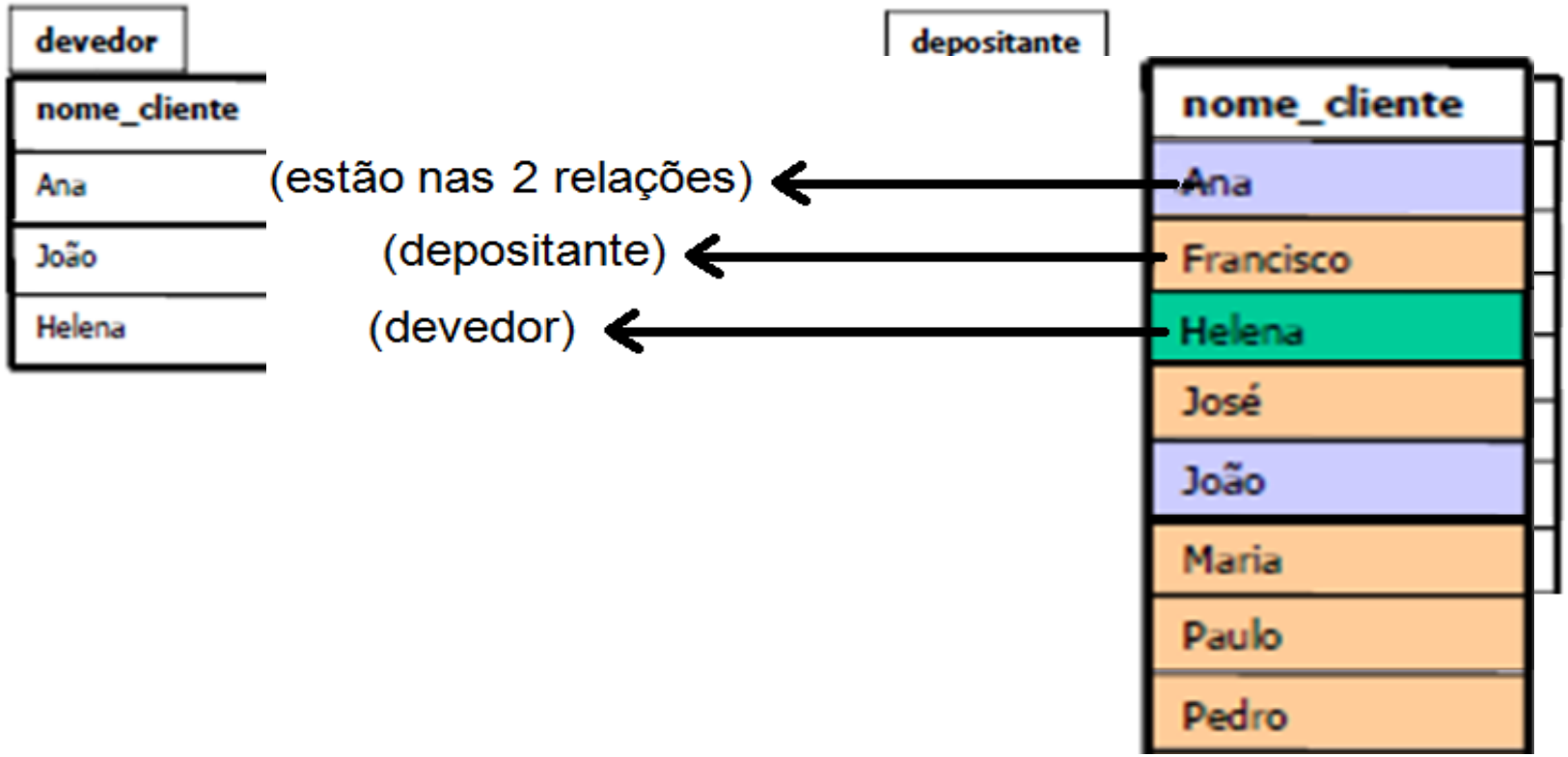
Operações sobre Conjuntos

Operação	Relação resultado – registros pertencentes –
$R \cup S$	a R, a S ou a ambas R e S
$R \cap S$	tanto a R quanto a S
$R - S$	a R mas não a S

R ∪ S

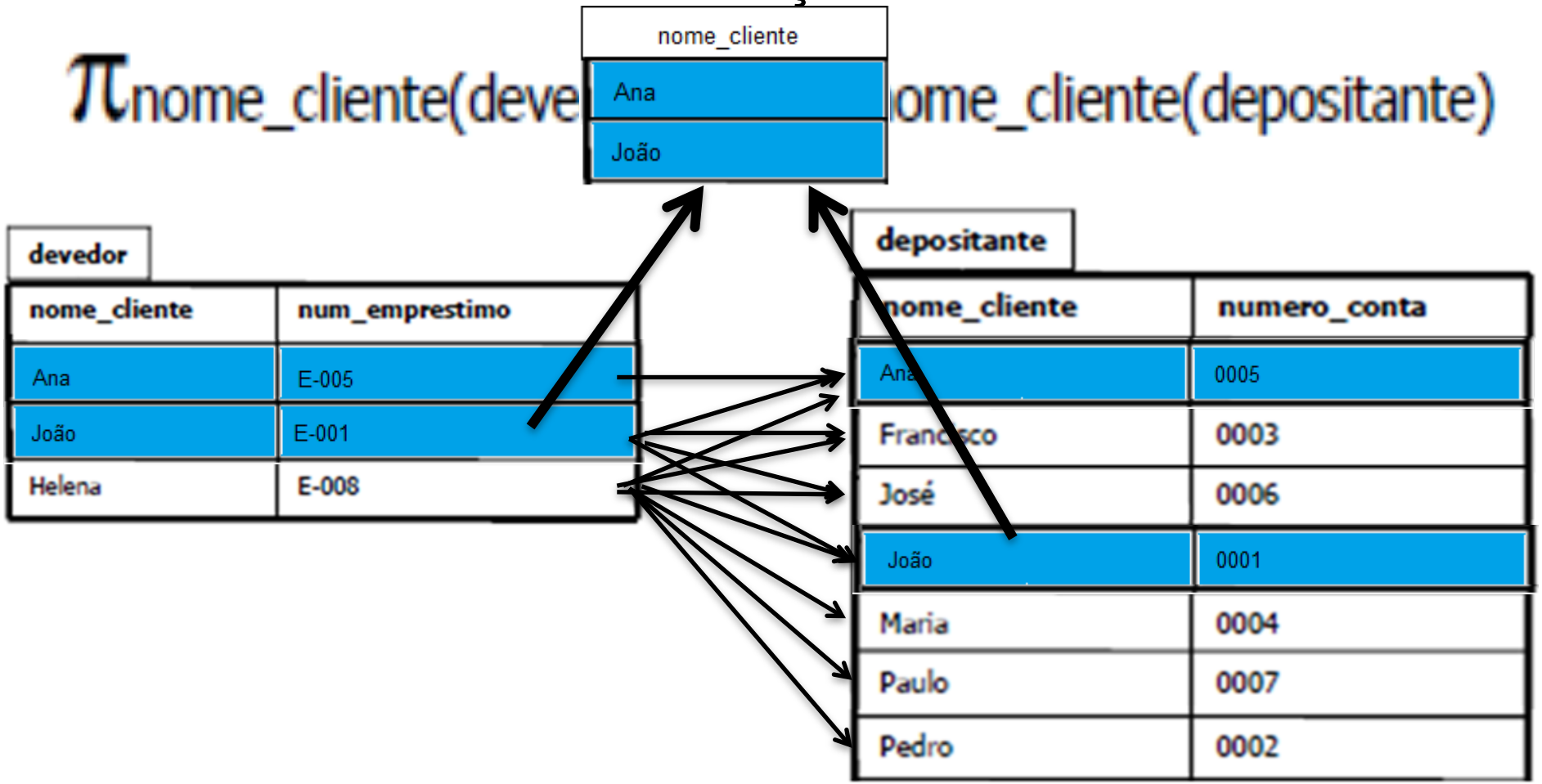
Operação UNIÃO: considere a seguinte consulta: selecionar todos os nomes de clientes que tenham um empréstimo, uma conta ou ambos. Deve-se utilizar as relações “depositante” e “devedor”.

$$\Pi_{nome\_cliente}(devedor) \cup \Pi_{nome\_cliente}(depositante)$$



R ∩ S

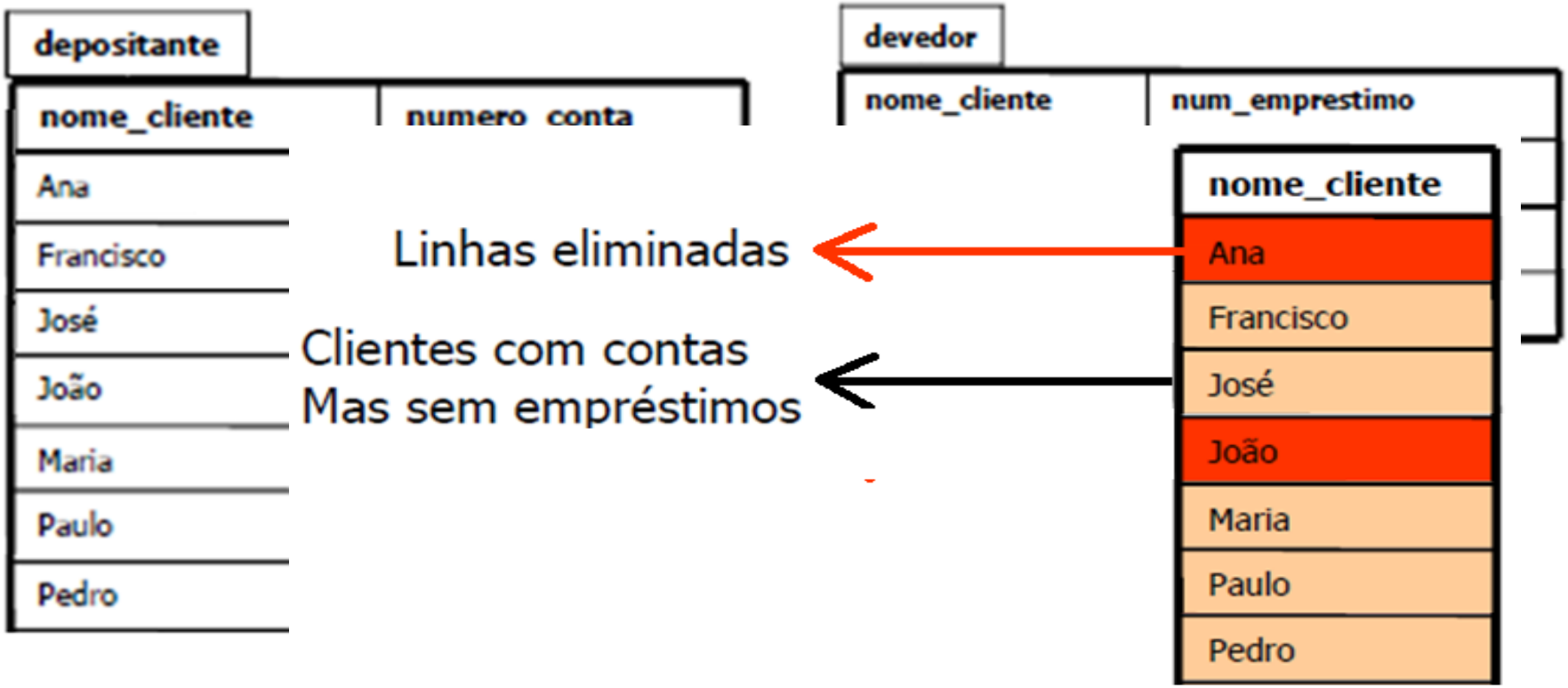
Operação de Interseção de conjuntos: captura todas as tuplas que encontram-se em uma relação "r", e que também encontram-se na relação "s".



R – S

Operação Diferença entre conjuntos: sendo "r" e "s" duas relações, "r-s" tem como resultado o conjunto de tuplas que estão na relação "r", mas não encontram-se na relação "s".

$$\pi_{nome\_cliente}(depositante) - \pi_{nome\_cliente}(devedor)$$



Referência

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S.
Sistema de banco de dados. Rio de Janeiro: Elsevier, 2006

Exercícios 1

1- Descreva brevemente os processos envolvidos no processamento de uma consulta.

2 - Por que utilizar ordenação?

3 - Qual a importância de mergesort externo?

4 - Suponha por simplicidade, dado o tamanho do buffer em blocos é $n_b = 5$ e o tamanho do arquivo original em blocos é b = 108. Calcule o custo do algoritmo.

a) Os números de passos iniciais M.
b) Calcule o total de acesso SEM considerar o resultado final.
c) Como no anterior, considere o resultado final.

Exercícios 2

5 - Dado:
Número de registros (r) = 20.000
Número de blocos de disco (b) = 3.000
Fator de bloco de disco (bfr) = 7
Índice primário em num_cliente "Chave primária"
número de níveis (x) = 6

a)Usando o algoritmo linear, binária e índice primário; igualdade na chave primária. Calcule e indique qual será a escolha do otimizador de consulta.

$\sigma_{\mathbf{num\_cliente = 9}}$ (cliente)

b) Usando o algoritmo linear e índice primário; desigualdade na chave primária. Calcule.

$\sigma_{\mathbf{num\_cliente > 16}}$ (cliente)