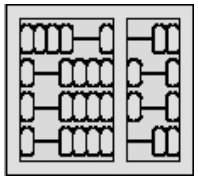


INSTITUTO DE COMPUTAÇÃO
Universidade Estadual de Campinas

Processamento de Consultas

Ricardo da Silva Torres
rtorres@ic.unicamp.br

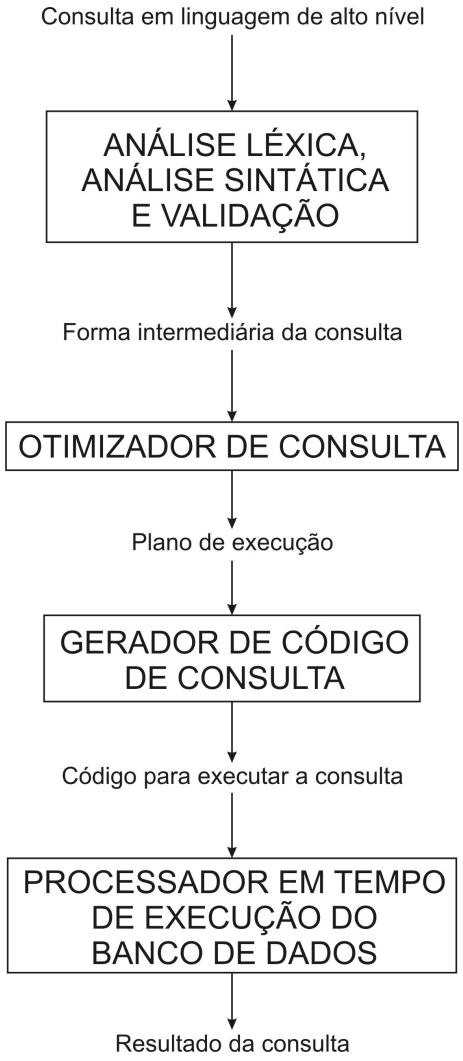
Tópicos do curso

- Introdução – conceitos básicos e arquitetura de SGBD
- Modelagem de Dados (MER)
- Modelo Relacional
- Linguagens de Manipulação de Dados
- Dependências Funcionais e normalização
- Processamento e Otimização de Consultas
- Processamento de Transações, Controle de Concorrência e Recuperação de Dados

Agenda

- Introdução ao processamento de consultas
- Otimização de consultas baseada em heurísticas
- Otimização de consultas baseada em estimativas de custos

Processamento de Consultas



- O processamento de consulta visa processar, otimizar e executar consultas de alto nível

Processo de Tradução

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > ( SELECT    MAX (SALARY)
                     FROM      EMPLOYEE
                     WHERE     DNO = 5);
```

```
SELECT    LNAME, FNAME
FROM      EMPLOYEE
WHERE     SALARY > C
```

```
SELECT    MAX (SALARY)
FROM      EMPLOYEE
WHERE     DNO = 5
```

$\pi_{\text{LNAME, FNAME}}(\sigma_{\text{SALARY>C}}(\text{EMPLOYEE}))$

$\mathcal{F}_{\text{MAX SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$

Árvore (Algébrica) da Consulta

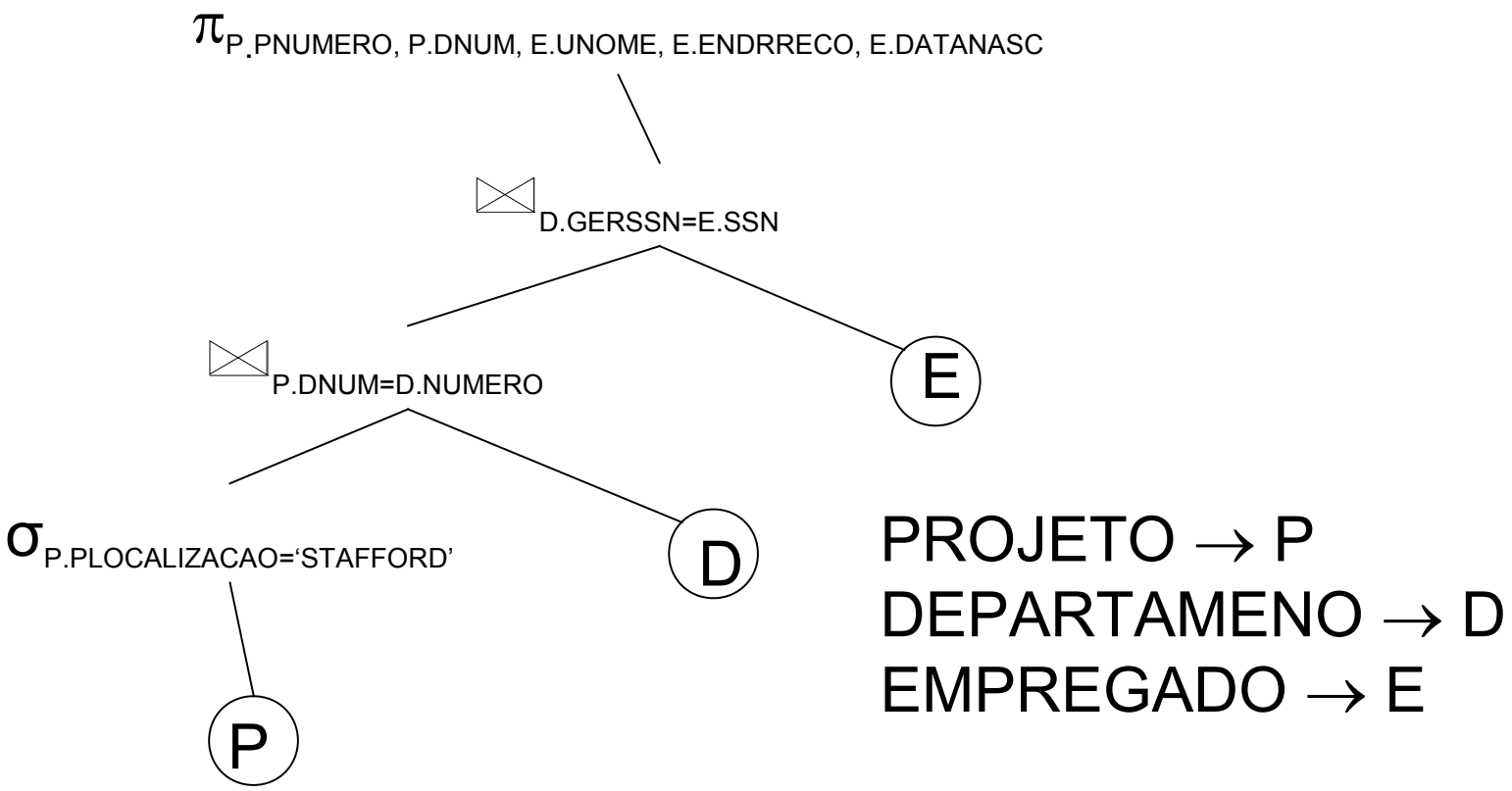
- Estrutura que representa o mapeamento da consulta para a álgebra relacional
  - Expressão da álgebra relacional "estendida"
  - Nós folha: relações do BD
  - Nós internos: operações da álgebra

Árvore (Algébrica) da Consulta

- Processamento da árvore
  - Nós internos:
    - Executados quando seus operandos estão disponíveis
    - São substituídos pela relação resultante
  - Nó raiz: último a ser executado

Árvores de Consulta

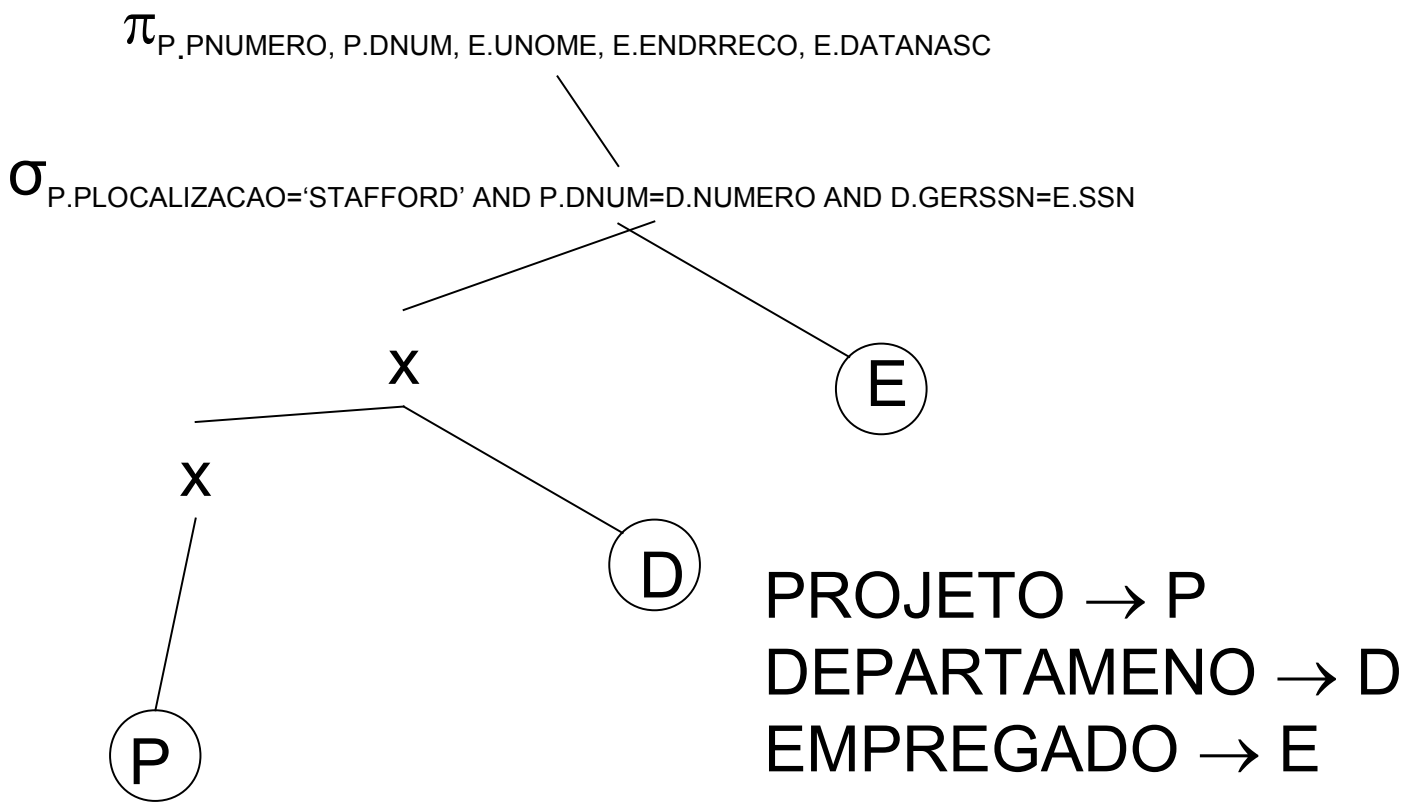
$\pi_{PNUMERO, DNUM, UNOME, ENDRRECO, DATANASC}$ $(((\sigma_{PLOCALIZACAO='STAFFORD'}(PROJETO))$
$\bowtie_{DNUM=DNUMERO}(DEPARTAMENTO))$ $\bowtie_{GERSSN=SSN}(EMPREGADO))$



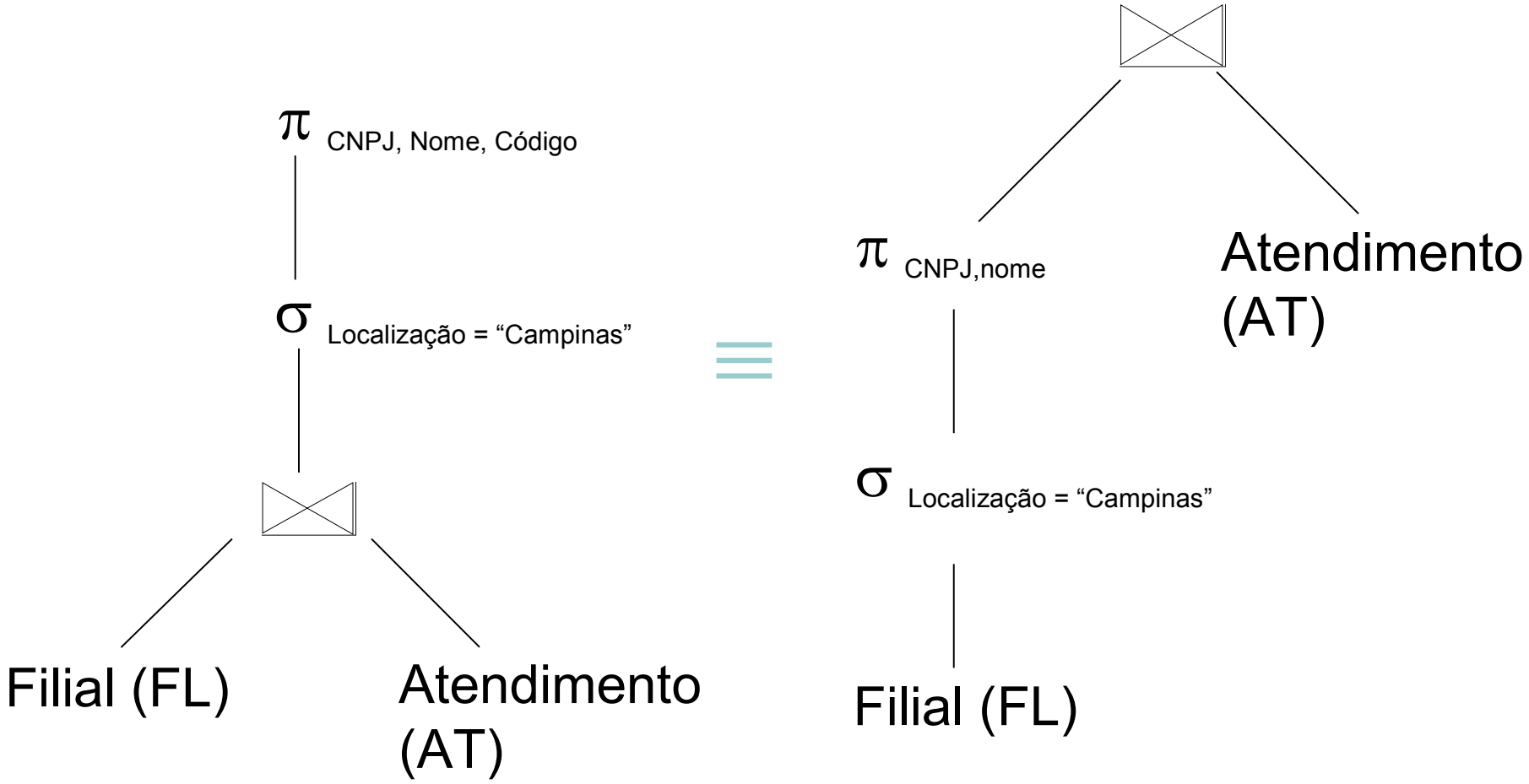
PROJETO → P
DEPARTAMENO → D
EMPREGADO → E

Árvore de Consulta Canônica

$\pi_{\text{PNUMERO, DNUM, UNOME, ENDRRECO, DATANASC}}$ $(((\sigma_{\text{PLOCALIZACAO='STAFFORD'}}(\text{PROJETO}))$
$\bowtie_{\text{DNUM=DNUMERO}}(\text{DEPARTAMENTO}))$ $\bowtie_{\text{GERSSN=SSN}}(\text{EMPREGADO}))$



Árvores Equivalentes



Otimização de consultas

"Otimização" – determinação do plano de execução

Otimização de consultas

- Processo de escolha de uma estratégia de execução para o processamento de consulta

Otimização de consultas

- Principais técnicas de otimização de consultas
  - **Baseada em regras heurísticas**
  - **Baseada em estimativas de custo**

Otimização de consultas

- Principais técnicas de otimização de consultas
  - **Baseada em regras heurísticas**
  - **Baseada em estimativas de custo**

Otimização baseada em heurísticas

- Utilizam **regras heurísticas** para modificar a estrutura interna (árvore) da consulta
- Uma mesma consulta $\rightarrow$ **várias** árvores
- Encontrar a árvore de consulta **mais eficiente**

Otimização baseada em heurísticas

- **Processo para otimização heurística**
  1. O parser de uma linguagem de consulta de alto nível gera uma representação interna inicial
  2. Regras heurísticas são aplicadas para otimizar a representação interna
  3. Um plano de execução é gerado para executar grupos de operações baseados em estruturas de acesso existentes para os arquivos envolvidos na consulta

Otimização de Consulta

- Baseada nas regras de equivalência algébrica
  - Reordenação da árvore a partir das regras de equivalência
  - O resultado das árvores deve ser o mesmo
    - Otimização válida
- Algoritmo de otimização algébrica
  - Indica a ordem de aplicação das regras e de outros processamentos de otimização

Regras de Equivalência Algébrica

- Cascata de Seleções
  - .

$$\sigma_{c1 \wedge c2 \wedge \ldots cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))))$$

- Comutatividade de Seleções
  - .

$$\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$$

- Cascata de Projeções
  - .

$$\pi_{at1,\, at2,\ldots,atn}(R) \equiv \pi_{at1}(\pi_{at2}(\ldots(\pi_{atn}(R))))$$

Regras de Equivalência Algébrica

- Comutatividade de Seleções e Projeções
  - . $\pi_{a1,\, a2\, ,\, ...,\, an}(\sigma_c\,(R)) \equiv \sigma_c\,(\pi_{a1,\, a2\, ,\, ...,\, an}\,(R))$
- Comutatividade de Produto Cartesiano
  - . R “X” S ≡ S “X” R
  - Válida também para junções

Regras de Equivalência Algébrica

- Comutatividade de Seleções e Junções
  - .

$\sigma_c (R \text{ “X” } S) \equiv (\sigma_c (R)) \text{ “X” } S$

$\sigma_c (R \text{ “X” } S) \equiv (\sigma_{c1} (R)) \text{ “X” } (\sigma_{c2} (S))$, com c = c1, c2

Regras de Equivalência Algébrica

- Comutatividade de Projeções e Junções
  - . $\pi_{at1}(R \times S) \equiv (\pi_{at1}(R)) \times S$

    $\pi_{at1,at2}(R \times S) \equiv (\pi_{at1}(R)) \times (\pi_{at2}(S))$

    $\pi_{at1,at2,at3}(R \times S) \equiv \pi_{at1}((\pi_{at2}(R)) \times (\pi_{at3}(S)))$
- Comutatividade de Operações de Conjunto
  - . $R \cup S \equiv S \cup R$ $\quad\quad$ $R \cap S \equiv S \cap R$

Otimização baseada em heurísticas

- Intuitivamente
  - Principal heurística $\rightarrow$ aplicar **seleção** e **projeção** antes de aplicar **junção** ou outras operações binárias
  - Executar operações de seleção e projeção o antes $\rightarrow$ reduzir o número de tuplas e atributos
  - Operações de junção e seleção mais restritivas devem ser ser executadas antes

Otimização baseada em heurísticas

- Heurísticas
  1. Quebrar seleções com condições **conjuntivas**
  2. Mover seleção para **baixo** na árvore
  3. Trocar folhas $\rightarrow$ seleções **mais restritivas** são executadas primeiro
  4. Combinar produto cartesiano com seleção $\rightarrow$ junção
  5. Mover projeções para **baixo** na árvore $\rightarrow$ novas projeções
  6. Identificar operações que podem ser executadas em um único algoritmo

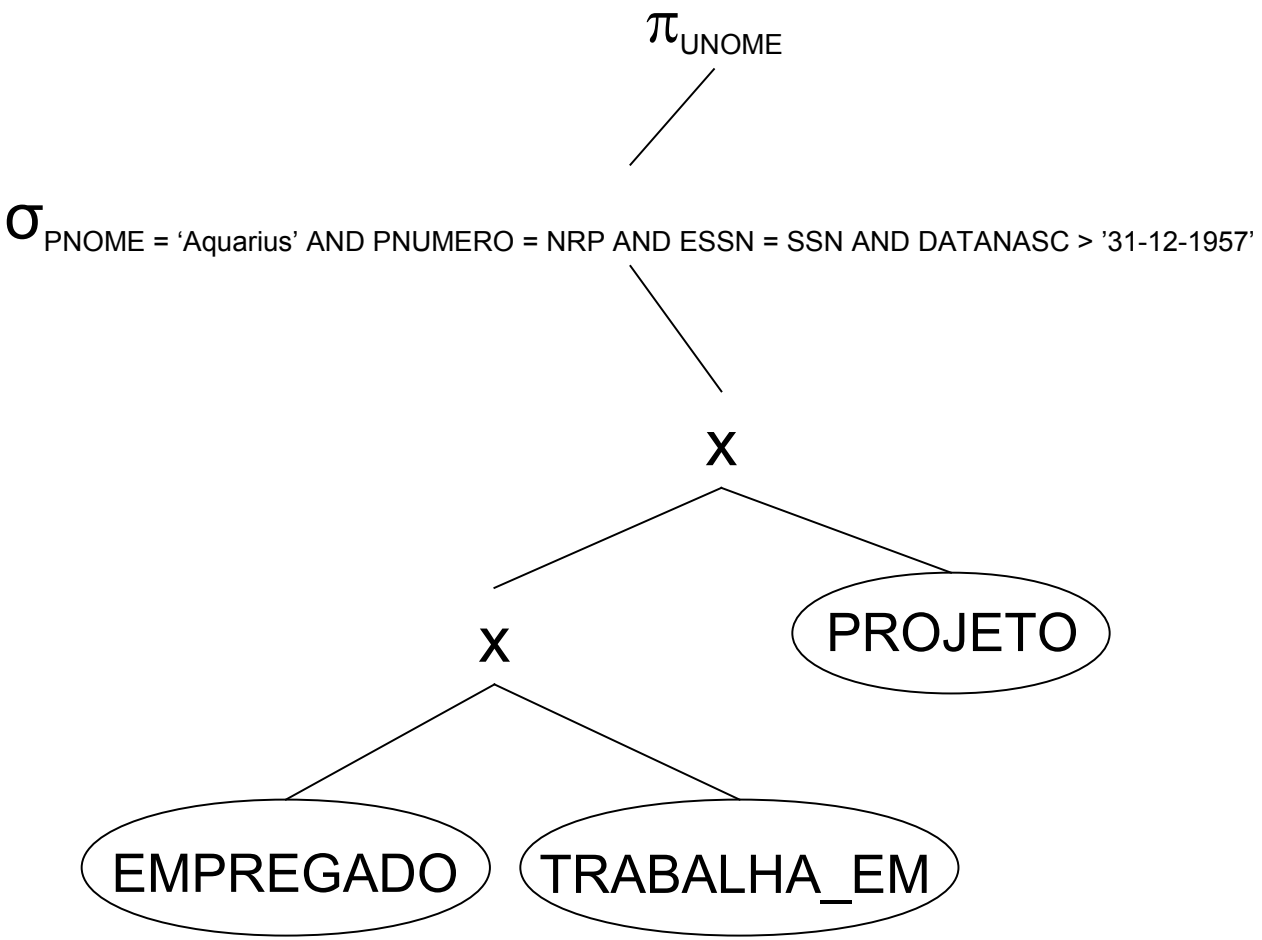
Otimização de consultas baseada em heurísticas

- Uma mesma consulta pode corresponder a muitas expressões em álgebra relacional — e daí a muitas árvore de consultas diferentes

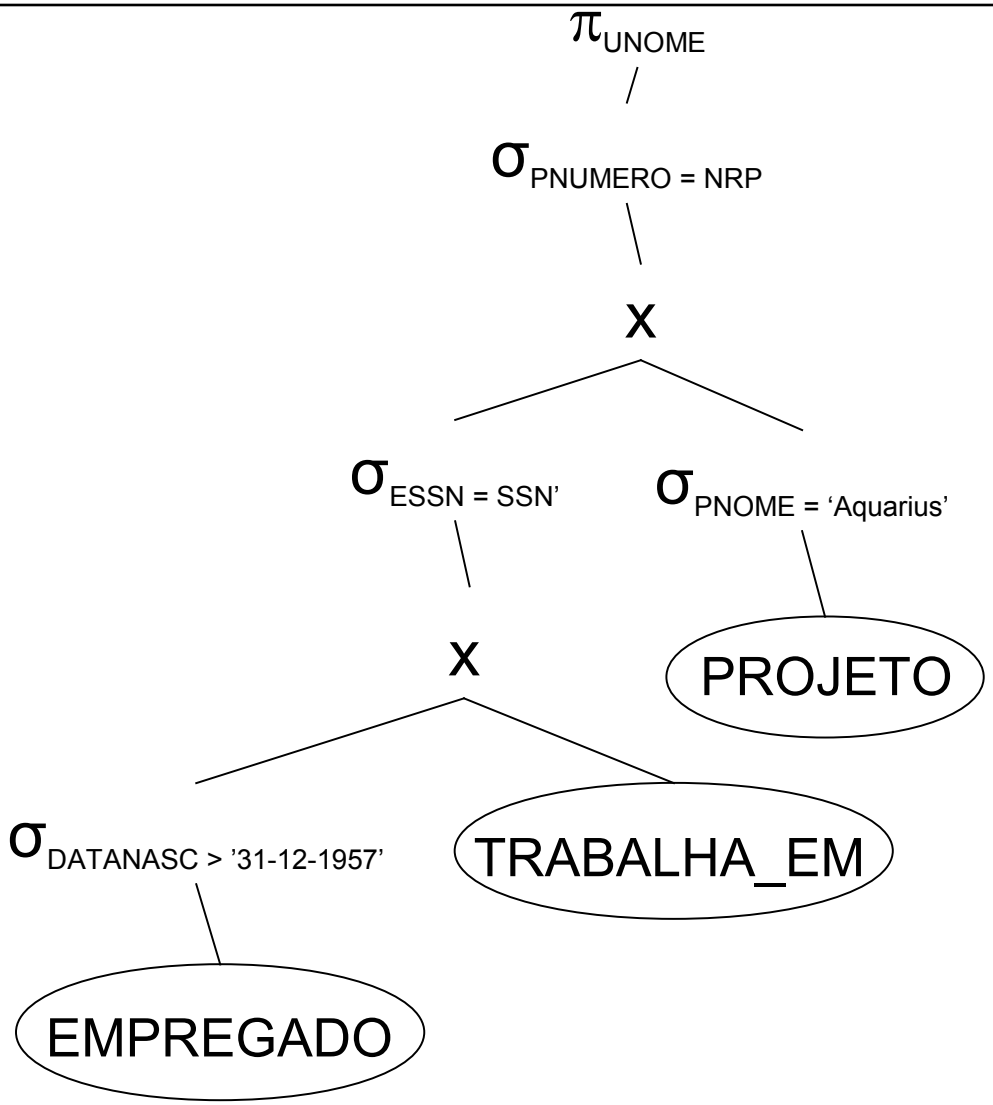
- O processo de otimização de consultas de árvores de consulta consiste em encontrar a árvore de consulta final que seja eficiente para executar

- **Exemplo**:
  Q: SELECT LNAME
  FROM EMPREGADO, TRABALHA_EM, PROJETO
  WHERE PNAME = 'AQUARIUS' AND PNMUBER=PNO
  AND ESSN=SSN AND BDATE > '1957-12-31';

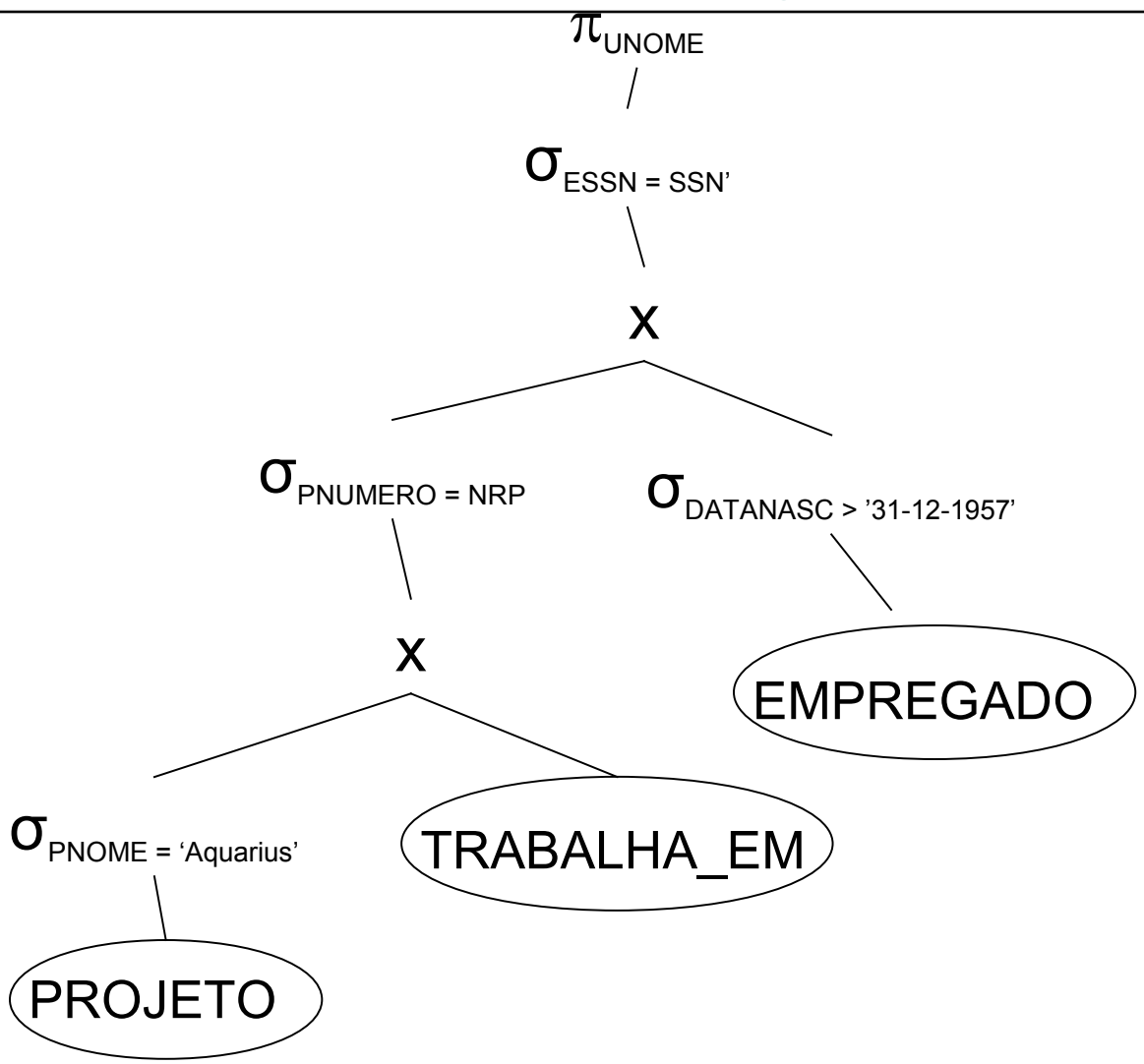
Exemplo de Otimização Heurística



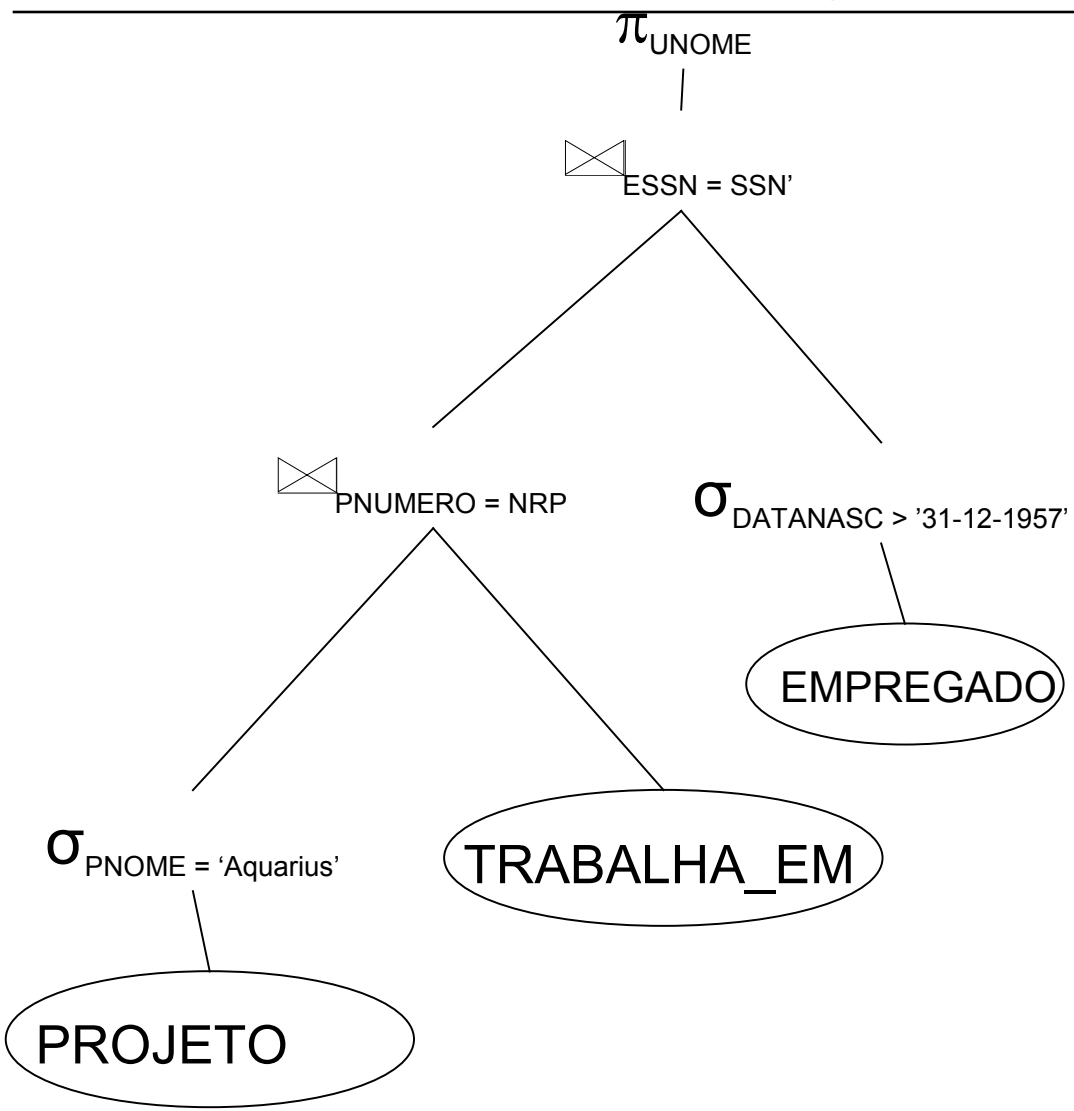
Exemplo de Otimização Heurística



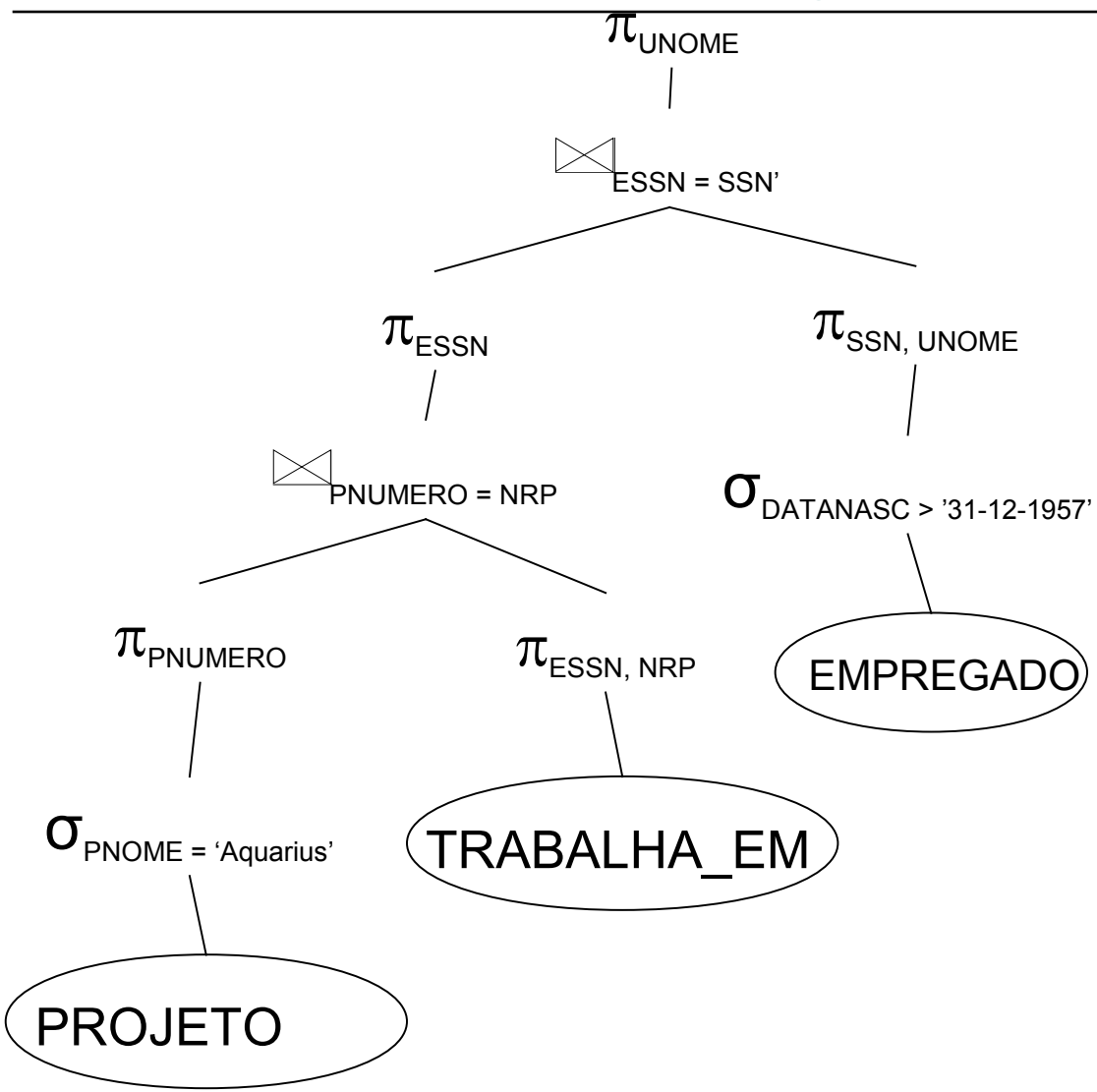
Exemplo de Otimização Heurística



Exemplo de Otimização Heurística



Exemplo de Otimização Heurística



Otimização de consultas baseada em heurísticas

1. A principal heurística é aplicar primeiro operações que reduzem o tamanho de resultados intermediários
2. Executar operações de seleção o mais cedo possível para reduzir o número de tuplas; executar operações de projeção o mais cedo possível para reduzir o número de atributos
3. As operações de junção e seleção mais restritivas devem ser executadas antes de operações similares

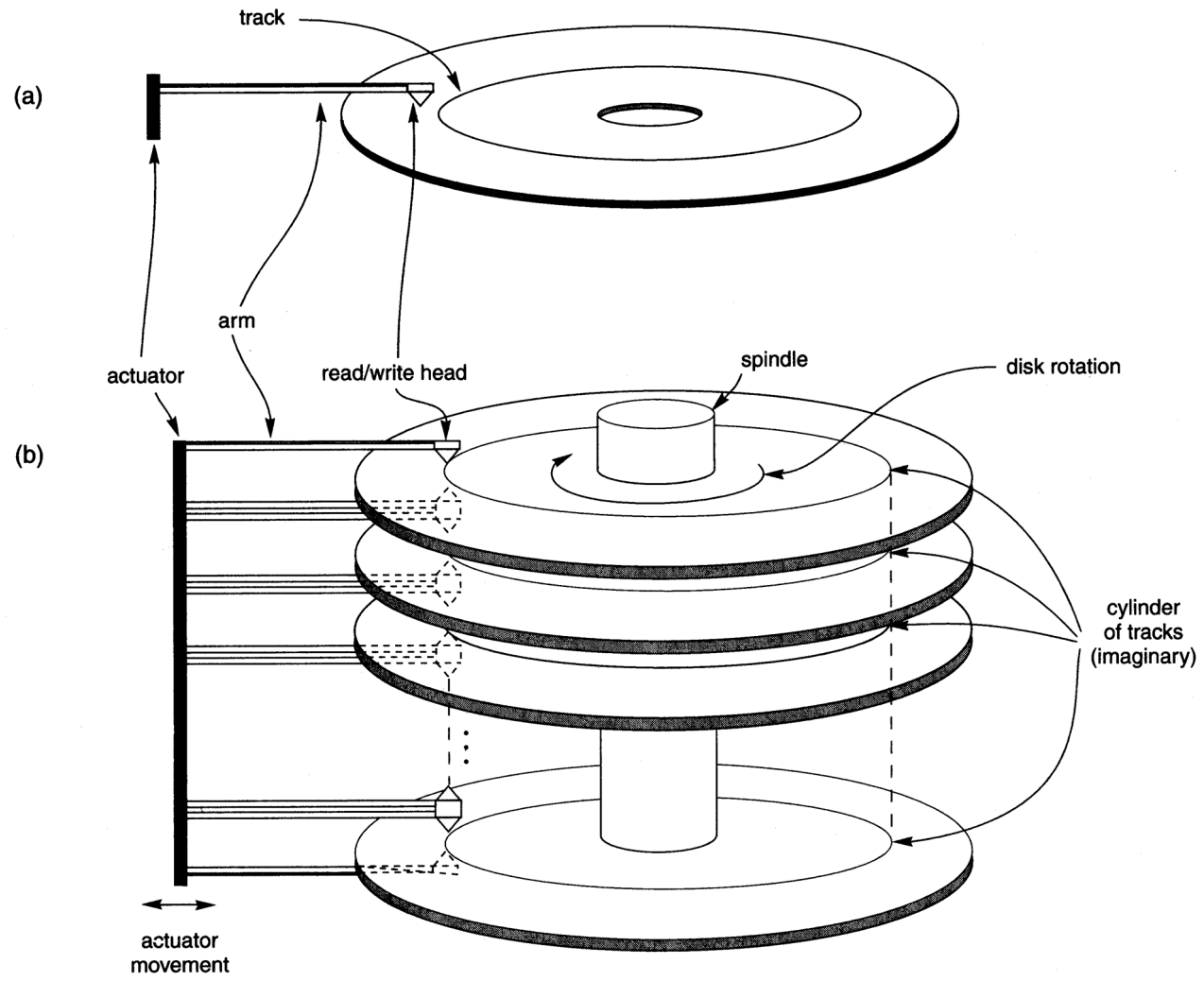
Otimização de consultas baseada em heurísticas

1. Quebre todas as operações de seleção com condições conjuntivas em uma cascata de operações de seleção
2. Mova cada operação de seleção o mais baixo possível na árvore de consulta
3. Rearrume os nós folhas da árvore de tal forma que as operações de seleção mais restritivas são executas primeiro na árvore de consulta
4. Combine a operação de produto cartesiano com uma operação de seleção subseqüente, formando uma operação de junção
5. Quebre e mova listas de projeção de atributos o mais baixo possível na árvore de consulta, criando novas operações de projeção quando necessário
6. Identifique subárvore que representa grupos de operações que podem ser executadas em um único algoritmo

Otimização de consultas

- Principais técnicas de otimização de consultas
  - **Baseada em regras heurísticas**
  - **Baseada em estimativas de custo**

Otimização de consultas baseada em estimativas de custo



Memória secundária

Custo de acesso a disco

Alguns conceitos importantes:

- Tempo de seek: é o tempo gasto para a cabeça de leitura/gravação se posicionar na trilha correta. Varia de 3 ms (para trilhas adjacentes) e até 100 ms (para trilhas que estão nos extremos do disco).
- Latência rotacional: é o tempo gasto para localizar o setor ao qual se quer ter acesso.
- Tempo de transferência: é o tempo gasto para a migração dos dados da memória secundária para a memória principal.

- Tempo de acesso: seek + latência + transferência

Otimização de consultas baseada em estimativas de custo

- Ordenação externa
  - Refer-se a algoritmos que são usados para ordenar arquivos de registros armazenados em disco; estes arquivos não cabem na memória
  - **Estratégia Sort-Merge**: pequenos subarquivos (runs) são ordenados; estes arquivos são intercalados, criando-se subarquivos maiores ordenados, …
    - Fase de ordenação: $n_R = \lceil (b/n_B) \rceil$
    - Fase de intercalação: $d_M = \text{Min}\ (n_B\text{-}1, n_R)$; $n_P = \lceil (\log_{dM}(n_R)) \rceil$

    n_R: número inicial de runs; b: número de blocos no arquivo;

    n_B: espaço de buffer disponível; d_M: grau de intercalação;

    n_P: número de passagens

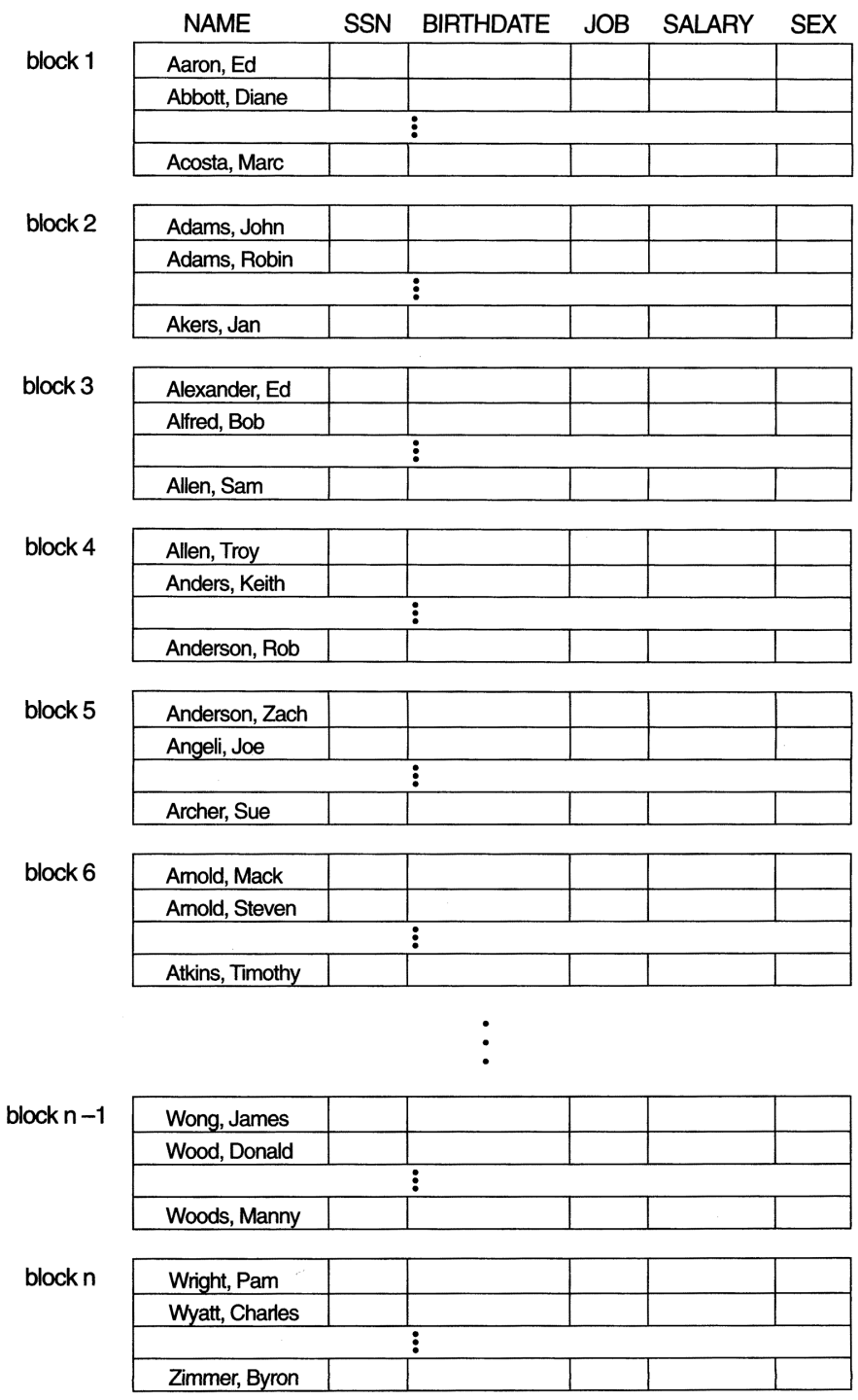
    Custo = $2b + 2\ (b \lceil (\log_{dM}(n_R)) \rceil)$

    Pior caso: 3 buffers

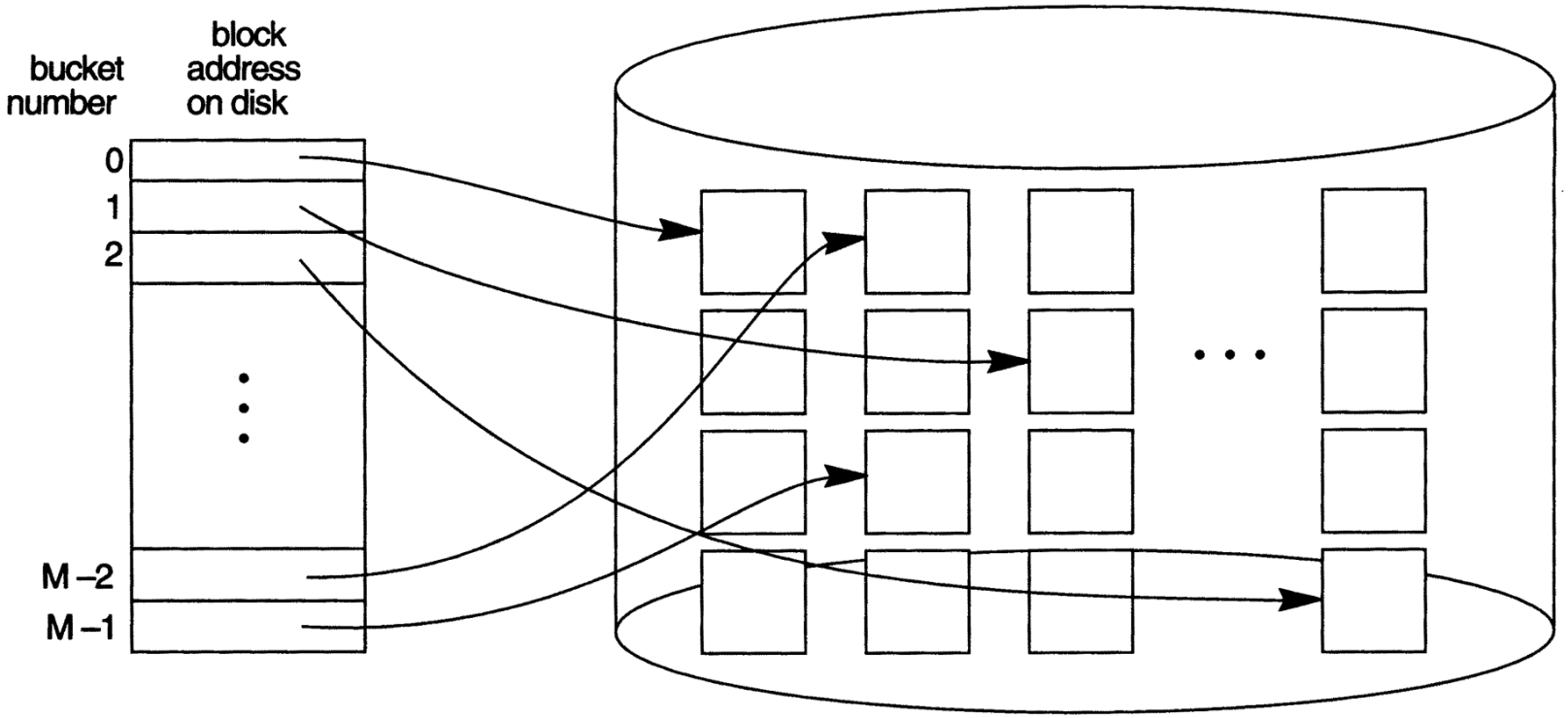
    Custo = $2b + 2\ (b \lceil (\log_2(n_R)) \rceil)$

```
set     i ← 1;
        j ← b;      {size of the file in blocks}
        k ← n_B;    {size of buffer in blocks}
        m ← ⌈(j/k)⌉;
{Sort Phase}
while (i <= m)
        do {
            read next k blocks of the file into the buffer or if there are less than k blocks remaining,
                 then read in the remaining blocks;
            sort the blocks in the buffer and write as a temporary subfile;
            i ← i + 1;
            }
{Merge Phase: merge subfiles until only 1 remains}
set     i ← 1;
        p ← log_{k-1} m;   {p is the number of passes for the merging phase}
        j ← m;
while (i <= p)
        do {
            n ← 1;
            q ← ⌈(j/ (k–1))⌉ ;   {number of subfiles to write in this pass}
            while (n <= q)
                 do {
                   read next k–1 subfiles or remaining subfiles (from previous pass) one block at a time;
                   merge and write as new subfile;
                   n ← n + 1;
                     }
            j ← q;
            i ← i + 1;
                     }
```

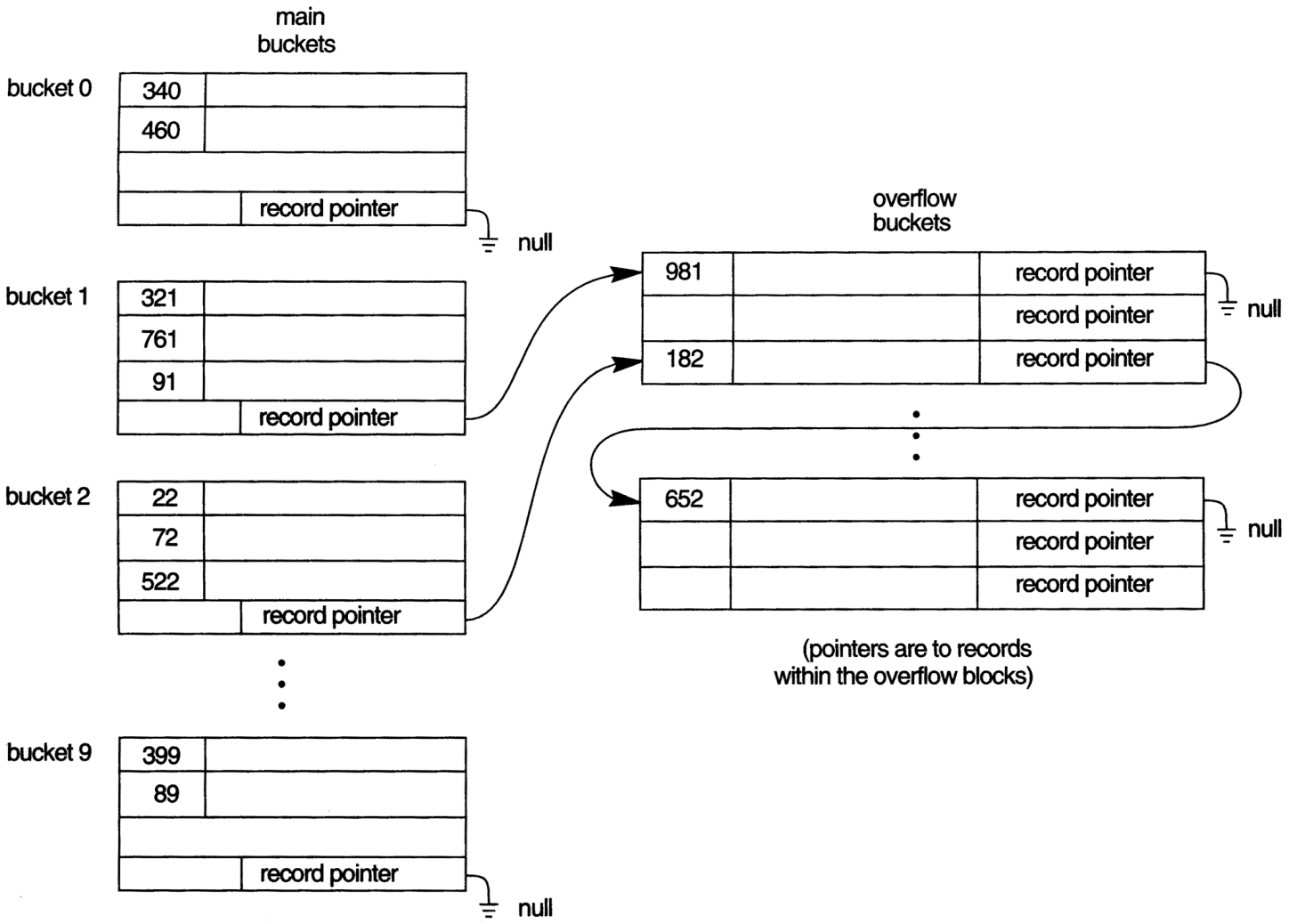
Otimiz seada em es

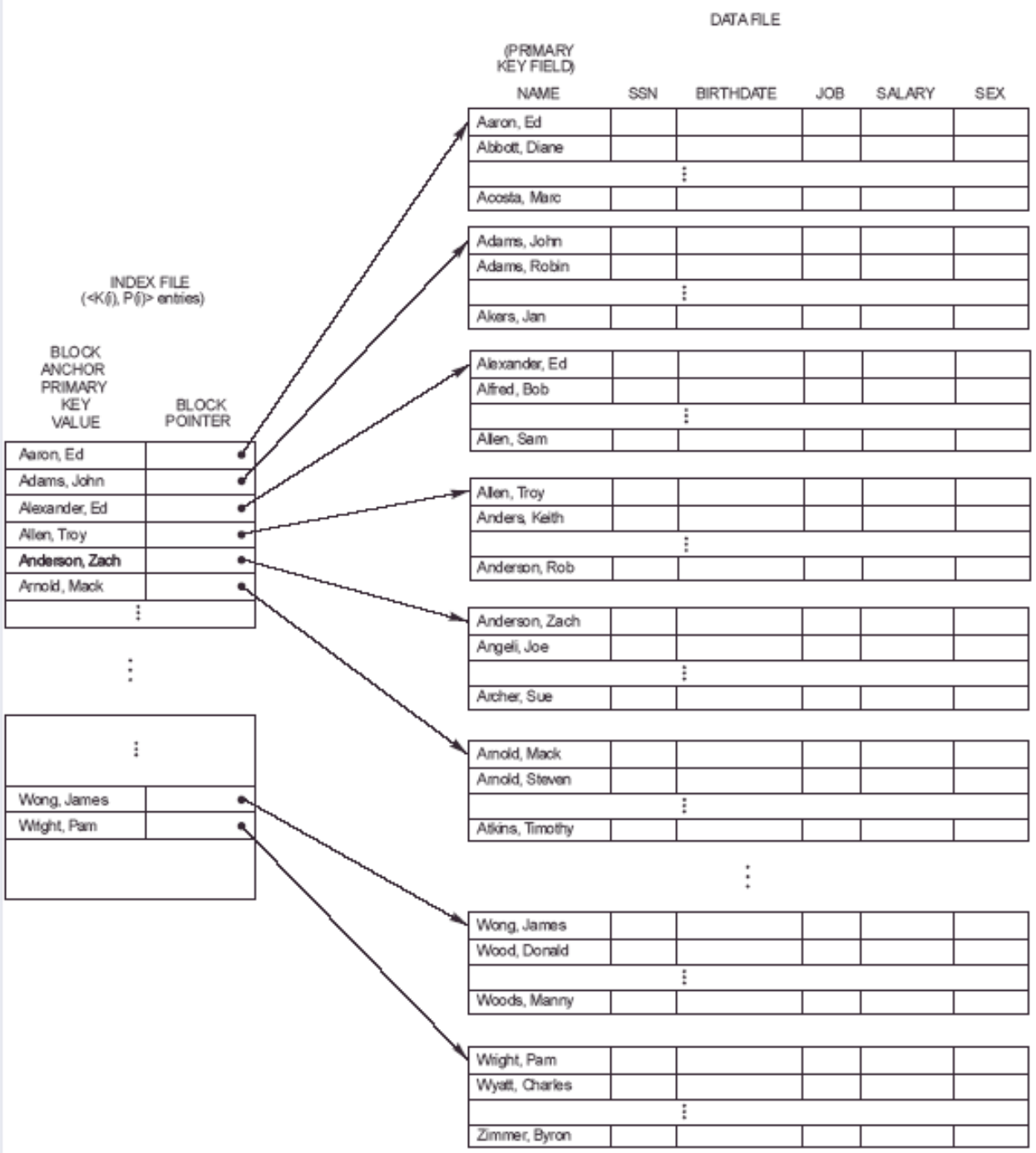


Otimização de consultas baseada em estimativas de custo

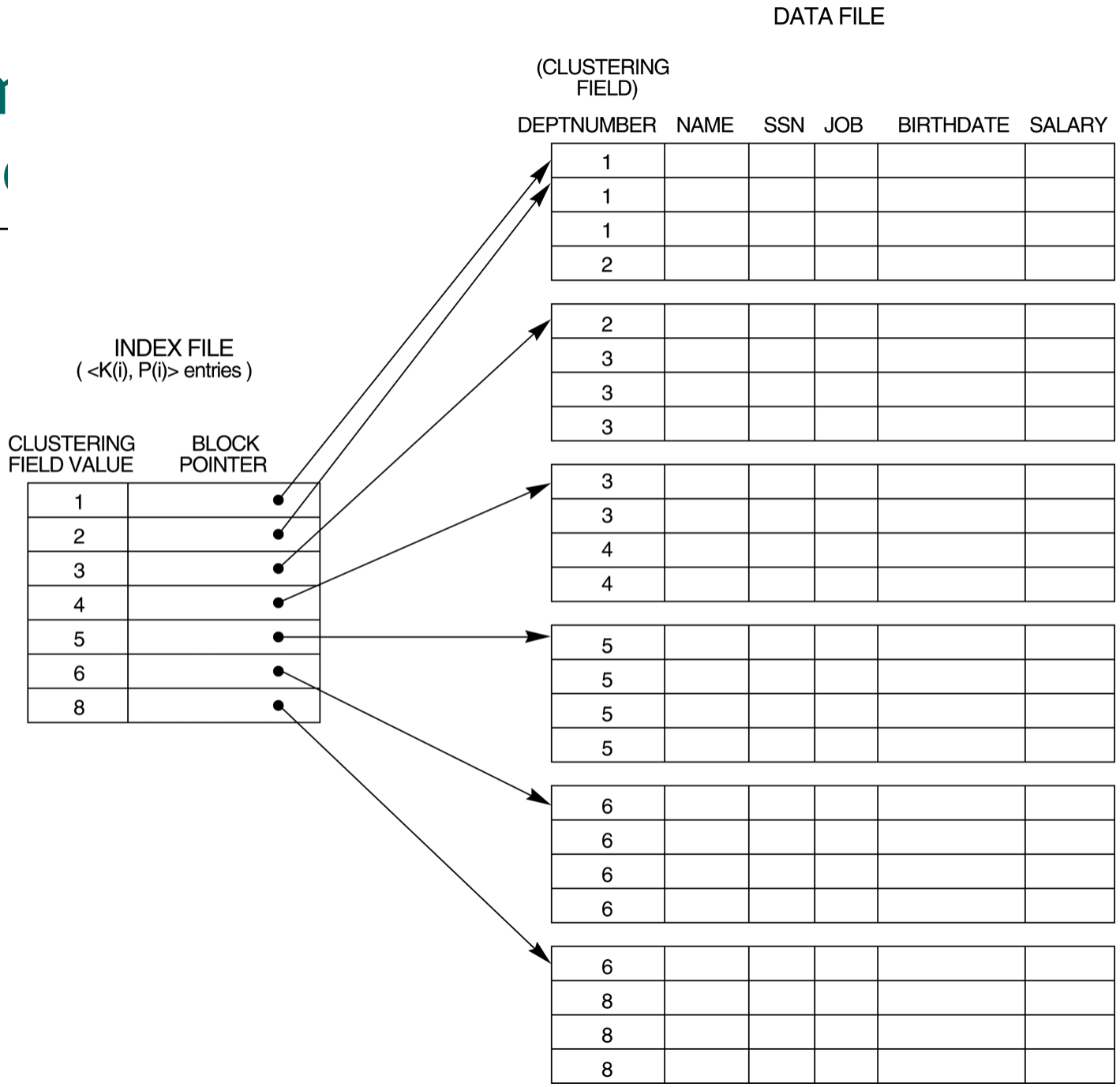


Otimização de consultas baseada em estimativas de custo

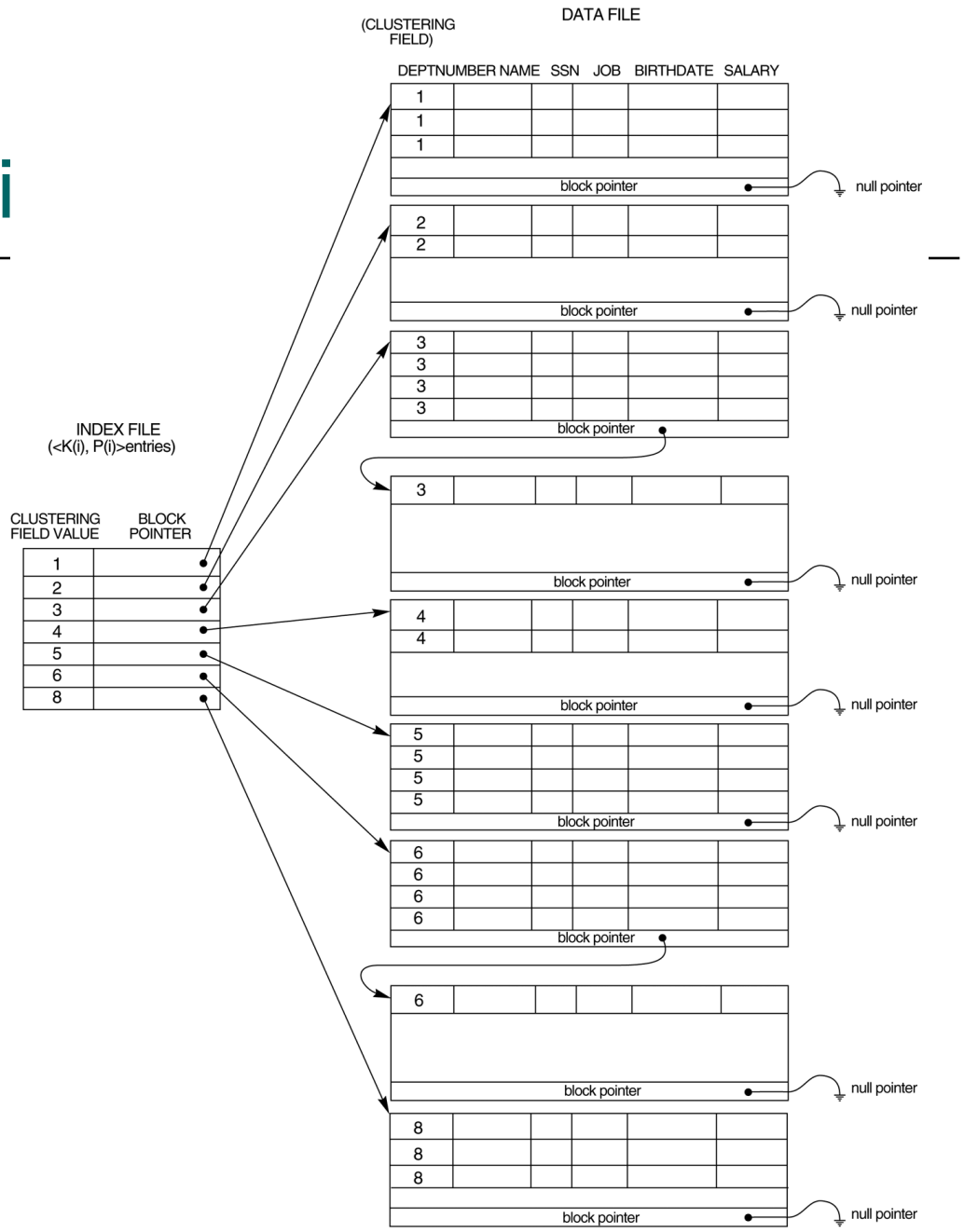



DATA FILE
(PRIMARY KEY FIELD)
NAME
SSN
BIRTHDATE
JOB
SALARY
SEX
Aaron, Ed
Abbott, Diane
Acosta, Marc
Adams, John
Adams, Robin
Akers, Jan
Alexander, Ed
Alfred, Bob
Allen, Sam
Allen, Troy
Anders, Keith
Anderson, Rob
Anderson, Zach
Angeli, Joe
Archer, Sue
Arnold, Mack
Arnold, Steven
Atkins, Timothy
Wong, James
Wood, Donald
Woods, Manny
Wright, Pam
Wyatt, Charles
Zimmer, Byron
INDEX FILE
(<K(i), P(i)> entries)
BLOCK ANCHOR PRIMARY KEY VALUE
BLOCK POINTER
Aaron, Ed
Adams, John
Alexander, Ed
Allen, Troy
Anderson, Zach
Arnold, Mack
Wong, James
Wright, Pam

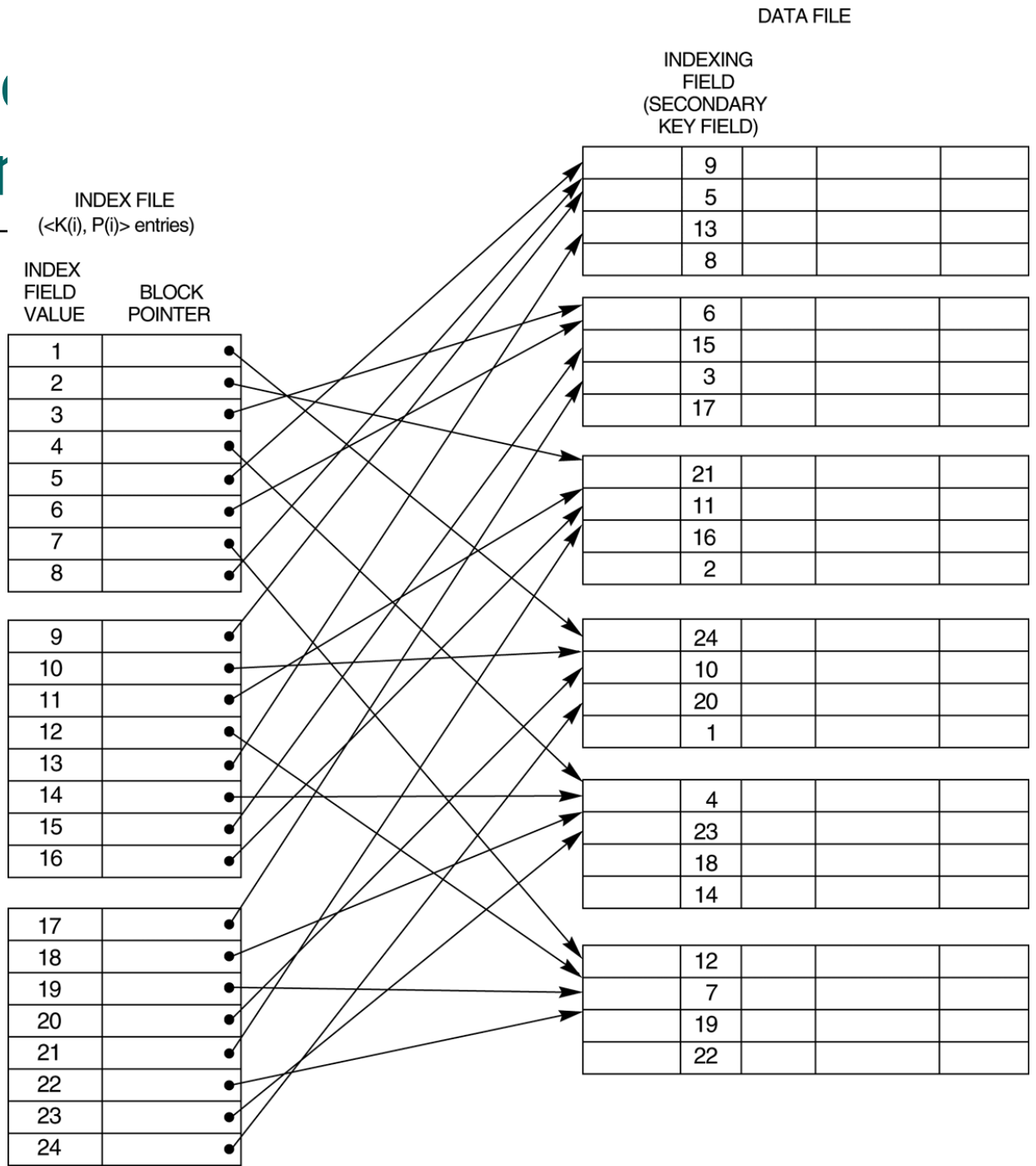
Otin
em


DATA FILE
(CLUSTERING FIELD)
DEPTNUMBER
NAME
SSN
JOB
BIRTHDATE
SALARY
INDEX FILE
(<K(i), P(i)> entries)
CLUSTERING FIELD VALUE
BLOCK POINTER
1
2
3
4
5
6
8

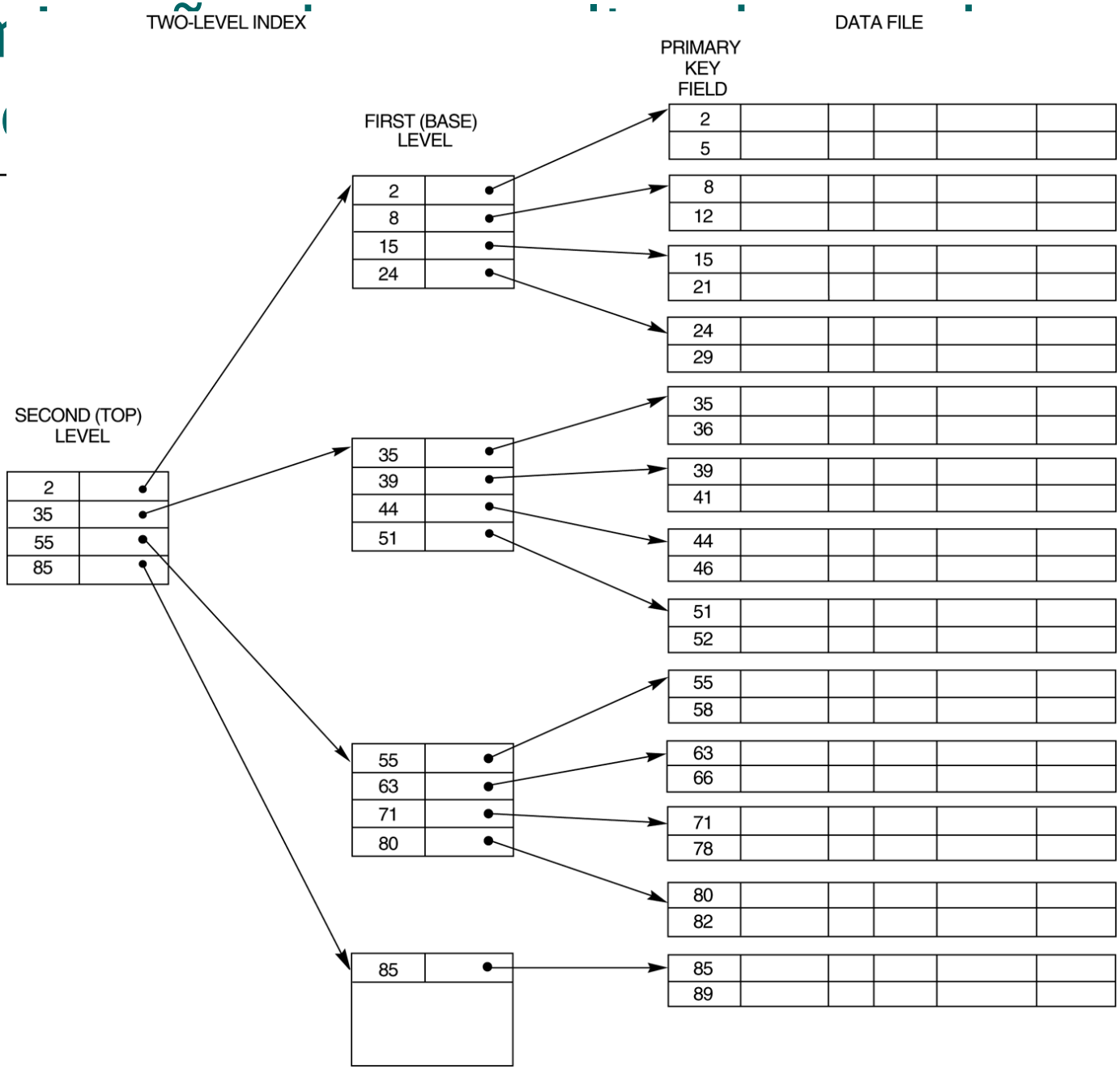
Otimização em estimati



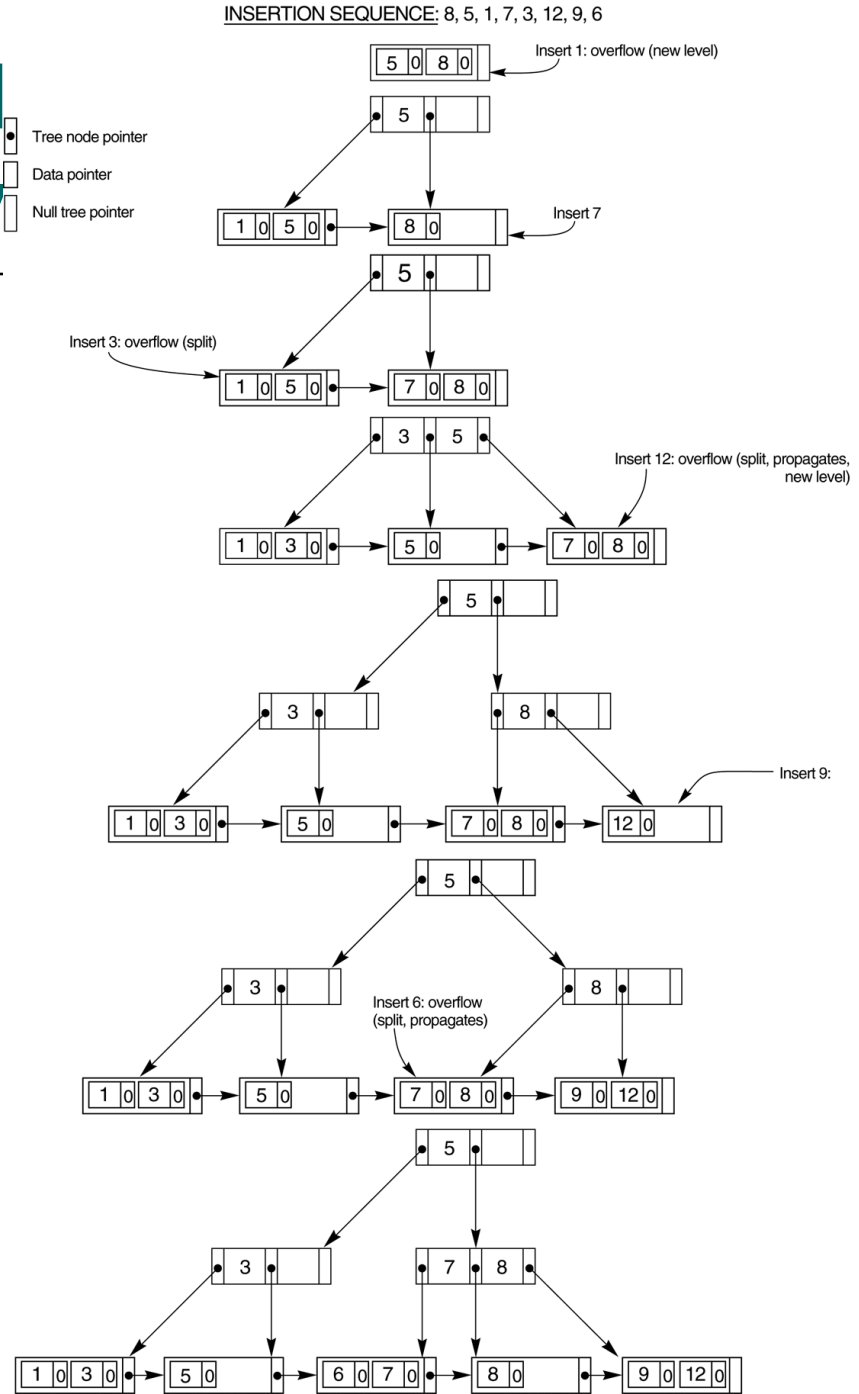
Otimiza
em estir



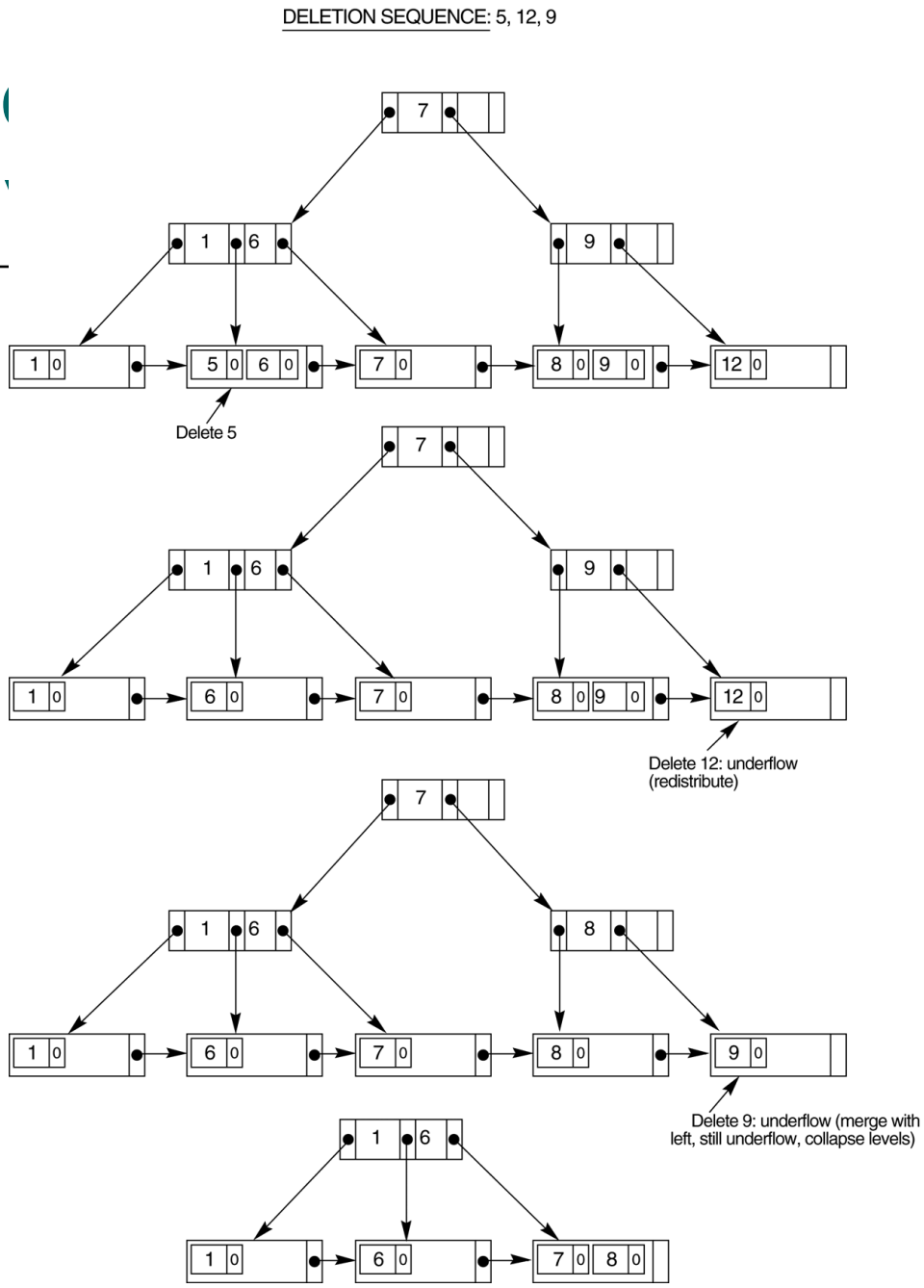
Otim
em


TWO-LEVEL INDEX
DATA FILE
PRIMARY KEY FIELD
FIRST (BASE) LEVEL
SECOND (TOP) LEVEL
2
35
55
85
2
8
15
24
35
39
44
51
55
63
71
80
85
2
5
8
12
15
21
24
29
35
36
39
41
44
46
51
52
55
58
63
66
71
78
80
82
85
89

Otimização d
em estimativ



Otimização
em estimati



Otimização de consultas baseada em estimativas de custo

- Operações de seleção
- Exemplos:

(OP1): $\sigma_{SSN='123456789'}(EMPLOYEE)$

(OP2): $\sigma_{DNUMBER>5}(DEPARTMENT)$

(OP3): $\sigma_{DNO=5}(EMPLOYEE)$

(OP4): $\sigma_{DNO=5\ AND\ SALARY>30000\ AND\ SEX=F}(EMPLOYEE)$

(OP5): $\sigma_{ESSN=123456789\ AND\ PNO=10}(WORKS\_ON)$

Otimização de consultas baseada em estimativas de custo

- S1. **Busca linear** (força bruta)
- S2. **Busca binária**
- S3. **Usando índice primário ou hash** aplicado a uma chave
- S4. **Usando índice primário** para recuperar múltiplos registros
- S5. **Usando um índice do tipo clustering** para recuperar múltiplos registros
- S6. **Usando um índice secundário (B+-tree)**
- S7. **Seleção conjuntiva**
- S8. **Seleção conjuntiva usando um índice composto**
- S9. **Seleção conjuntiva pela interseção de ponteiros de registros**

Otimização de consultas baseada em estimativas de custo

- Algoritmos para junção
  - J1. **Nested-loop join**
  - J2. **Single-loop join**
  - J3. **Sort-merge join**
  - J4. **Hash-join**

Sort-merge join

```
sort the tuples in R on attribute A;  (*assume R has n tuples (records) *)
sort the tuples in S on attribute B;  (*assume S has m tuples (records) *)
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do{     if R(i)[A] > S(j)[B]
              then      set j ← j+1
        elseif R(i)[A] < S(j)[B]
              then      set i ← i+1
              else  {   (* R(i)[A] < S(j)[B], so we output a matched tuple*)
                        output the combined tuple <R(i), S(j)> to T;
                              (*output other tuples that match R(i), if any*)
                              set l ← j+1;
                              while (l ≤ m) and (R(i)[A] < S(l)[B])
                              do    {       output the combined tuple <R(i), S(l)> to T;
                                            set l ← l+1
                                    }
                              (*output other tuples that match S(j), if any*)
                              set k ← i+1;
                              while (k ≤ n) and (R(k)[A] < S(j)[B])
                              do    {       output the combined tuple <R(k), S(j)> to T;
                                            set k ← k+1
                              }
                             set i ← k, j ← l
              }
}
```

Otimização de consultas baseada em estimativas de custo

- Fatores que interferem na junção:
  - Espaço de *buffer* disponível
  - Fator de seleção da junção
  - Escolha da relação interna e externa no loop aninhado

Otimização de consultas baseada em estimativas de custo

- **Projeção:**
  - π **<Lista de atributos>**(R)
    - A <lista de atributos> inclui a chave?
  - Métodos para remover tuplas duplicatas
    - Ordenação
    - Hashing

Projeção

```
create a tuple t[<attribute list>] in T' for each tuple t in R;
        (*T' contains the projection result before duplicate elimination*)
if <attribute list> includes a key of R
        the T ← T'
        else    {       sort the tuples in T';
                        set i ← 1, j ← 2;
                        while i ≤ n
                                do      {       output the tuple T'[i] to T;
                                                while T'[i] = T'[j]  and j ≤ n do j ← j+1; (*eliminate duplicates*)
                                                i ← j; j ← i+1
                                        }
                }
(* T contains the projection result after duplicate elimination *)
```

Otimização de consultas baseada em estimativas de custo

- **Operações de conjunto:**
  - Solução baseada em ordenação

União

```
sort the tuples in R and S using the same unique sort attributes;
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do    {       if R(i) > S(j)
              then    {       output S(j) to T;
                              set j ← j+1
                      }
              elseif R(i) > S(j)
              then    {       output R(i) to T;
                              set i ← i+1
                      }
              else set j ← j+1 (*R(i)=S(j), so we skip one of the duplicate tuples*)
      }
if (i ≤ n) then add tuples R(i) to R(n) to T;
if (j ≤ m) then add tuples S(j) to S(m) to T;
```

Otimização de consultas baseada em estimativas de custo

- **Processo de otimização de consulta:** Estima-se e compara-se o custo de deferentes estratégias de execução existentes e escolhe-se a estratégia com menor custo estimado

- Questões que devem ser consideradas
  - Função de custo
  - Número de estratégias de execução que devem ser consideradas

Otimização de consultas baseada em estimativas de custo

- Custos associados a execução de uma consulta
  1. Custo de acesso à memória secundária
  2. Custo de armazenamento
  3. Custo de computação
  4. Custo relacionado ao uso de memória
  5. Custo de comunicação

Otimização de consultas baseada em estimativas de custo

- **Informações mantidas em catálogo**
  - Informação sobre o tamanho de um arquivo
    - **Número de registros (tuplas) (r),**
    - **Tamanho do registro (R),**
    - **Número de blocos (b)**
    - **Fator de bloco (bfr)**
  - Informação sobre índices e atributos indexados de um arquivo
    - **Número de níveis (x)** de cada índice multinível
    - **Número de blocos do primeiro nível do índice (b_{I1})**
    - **Número de valores distintos (d)** de um atributo
    - **Seletividade (sl)** de um atributo
    - **Cardinalidade da seleção (s)** de um atributo.
      - (s = sl * r)

Otimização de consultas baseada em estimativas de custo

- **Exemplos de funções de custo para a operação de seleção**
- **S1. Busca linear**

  $C_{S1a} = b$;

  Para uma condição de igualdade na chave, $C_{S1a} = (b/2)$ se o registro é encontrado; caso contrário $C_{S1a} = b$.
- **S2. Busca binária**

  $C_{S2} = \log_2 b + \lceil (s/bfr) \rceil - 1$

  Para uma condição de igualdade em um atributo chave

  $C_{S2} = \log_2 b$
- **S3. Usando um índice primário (S3a) ou hash (S3b) para recuperar registros de um arquivo**

  $C_{S3a} = x + 1$; $C_{S3b} = 1$ para hash linear ou estático;

  $C_{S3b} = 1$ para hash extensível;

Otimização de consultas baseada em estimativas de custo

- **S4. Usando um índice para recuperar múltiplos registros:**

  Para uma condição aplicada a um campo chave $C_{S4} = x + (b/2)$

- **S5. Usando índice do tipo clustering para recuperar múltiplos registros**:

  $C_{S5} = x + \lceil (s/bfr) \rceil$

- **S6. Usando índice secundário (B^+-tree):**

  Para uma condição de igualdade, $C_{S6a} = x + s$;

  Para uma condição do tipo >, <, >=, ou <=,

  $C_{S6a} = x + (b_{I1}/2) + (r/2)$

Otimização de consultas baseada em estimativas de custo

- **S7. Seleção conjuntiva:**

  Use métodos S1 ou um dos métodos de S2 a S6

  Use uma condição para recuperar os registros e então verifique se os registros recuperados satisfazem as outras condições
- **S8. Seleção conjuntiva usando um índice composto**:

  Igual a S3a, S5 ou S6a, dependendo do tipo de índice

Otimização de consultas baseada em estimativas de custo

Exemplo de funções de custo para Junção

- **Seletividade de junção (js)**

  js = | (R $\bowtie_C$ S) | / | R x S | = | (R $\bowtie_C$ S) | / (|R| * |S |)

  Se condição não existe, js = 1;

  Se nenhuma tupla satisfaz a condição C, js = 0;

  Usualmente, **0 <= js <= 1**;

- **Tamanho do arquivo resultado depois da operação de junção** $\bowtie$

  | (R $\bowtie_C$ S) | = js * |R| * |S |

Otimização de consultas baseada em estimativas de custo

- **J1. Nested-loop join:**
  $C_{J1} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$
  (Uso de R para loop externo)
- **J2. Single-loop join** (usando uma estrutura de acesso para recuperar os registros corretos)
  Se um índice existe para um atributo de junção B de S com x_B níveis. Recupere cada registro s em R e então use o índice para recuperar todos os registros t de S que satisfazem t[B] = s[A].
  O custo depende do tipo de índice

Otimização de consultas baseada em estimativas de custo

- **J2. Single-loop join (cont.)**
  Para um índice secundário,
  $C_{J2a} = b_R + (|R| * (x_B + s_B)) + ((js* |R|* |S|)/bfr_{RS})$;
  Para um índice do tipo clustering,
  $C_{J2b} = b_R + (|R| * (x_B + (s_B/bfr_B))) + ((js* |R|* |S|)/bfr_{RS})$;
  Para um índice primário,
  $C_{J2c} = b_R + (|R| * (x_B + 1)) + ((js* |R|* |S|)/bfr_{RS})$;
  Se existe um hash para um dos dois atributos da junção — B de S
  $C_{J2d} = b_R + (|R| * h) + ((js* |R|* |S|)/bfr_{RS})$;
- **J3. Sort-merge join:**
  $C_{J3a} = C_S + b_R + b_S + ((js* |R|* |S|)/bfr_{RS})$;
  (C_S: Custo para ordenar os arquivos)

Processamento de Consultas

- Alternativas e estimativas de custo:
  - Processamento de operações de conjunto ($\cup$,—,$\cap$,x)
    - Operação de produto cartesiano:
      - Se R tem n tuplas e j atributos e S tem m tuplas e k atributos, a relação resultado terá n*m tuplas e j+k atributos
      - Operação muito cara computacionalmente, deve ser substituída por junção quando possível
    - Operação de união:
      - Deve-se organizar as duas relações pelos mesmos atributos e, em seguida, procurar e intercalar os arquivos organizados

Processamento de Consultas

- Alternativas e estimativas de custo:
  - Processamento de operações de conjunto (∪,—,∩,x)
    - Operação de interseção:
      - Devem-se organizar as duas relações pelos mesmos atributos e, em seguida, procurar e intercalar os arquivos organizados
      - Devem-se manter no resultado apenas as tuplas que aparecem em ambas as relações
    - Operação de diferença de conjuntos:
      - Devem-se organizar as duas relações R e S pelos mesmos atributos e, em seguida, procurar e intercalar os arquivos organizados
      - Devem-se manter no resultado apenas as tuplas que aparecem em R mas não aparecem em S

Interseção e Diferença

```
sort the tuples in R and S using the same unique sort attributes;
set i←1, j←1;
while (i ≤ n) and (j ≤ m)
do    {       if R(i) > S(j)
              then    {       output S(j) to T;
                              set j←j+1
                      }
              elseif R(i) > S(j)
              then    {       output R(i) to T;
                              set i←i+1
                      }
              else set j←j+1 (*R(i)=S(j), so we skip one of the duplicate tuples*)
      }
if (i ≤ n) then add tuples R(i) to R(n) to T;
if (j ≤ m) then add tuples S(j) to S(m) to T;

sort the tuples in R and R using the same unique sort attributes;
set i←1, j←1;
while (i ≤ n) and (j ≤ m)
do    {       if R(i) > S(j)
              then            set j←j+1
              elseif R(i) < S(j)
              then            set i← i+1
              else    {       output R(i) to T; (* R(i)=S(j), so we output the tuple *)
                              set i←i+1, j←j+1
                      }
      }
```

Problema 1

- Empregado $|X|_{DNO=DNUMERO}$ Departamento
- Número de buffers disponíveis: $n_B = 7$
- Departamento: $r_D = 50$ registros em $b_D = 10$ blocos
- Empregado: $r_E = 6000$ registros em $b_E = 2000$ blocos
- Loops aninhados:
  - Ter na memória quantos blocos sejam possíveis do arquivo sendo utlizado no loop externo (n_B-2)
  - Ler um bloco por vez do arquivo usado no loop interno
  - Mais um bloco é utilizado para manter o resultado da junção

- Qual relação deve ser utilizada no loop externo?

Problema 1

- Número total de blocos acessados para arquivo externo: b_E
- Número de vezes que (n_b-2) blocos de arquivo externo são carregados: floor(b_E/(n_B-2))
- Número total de blocos acessados para o bloco interno: b_D* floor(b_E/(n_B-2))

- Empregado no loop externo:
  - Total = 2000 + 10*floor(2000/5) = 6000 I/Os
- Departamento no loop externo:
  - Total = 10 + 2000*(10/5) = 4010 I/Os

Problema 2

- Op1: Empregado $|X|_{DNO=DNUMERO}$ Departamento
- Op2: Departamento $|X|_{GERSSN=SSN}$ Empregado
- Dados:
  - Há 3 buffers disponíveis para fazer a junção
  - Empregado: r_E = 6000 registros em b_E = 2000 blocos
  - Departamento: r_D = 125 registros armazenados em b_D = 13 blocos
  - Existe um índice primário para DNUMERO de Departamaneto com $x_{DNUMERO}$ = 1 nível
  - Existe um índice secundário para GERSSN de Departamento com cardinalidade de seleção s_{GERSSN} = 1 e x_{GERSSN} = 2
  - Seletividade js da junção Op1: js = 1 / 125 (DNUMERO é chave de Departamento)
  - Fator de divisão do bloco para o arquivo de junção resultante: bfr_{ED} = 4 registros por bloco

Problema 2

- Qual o custo para a operação Op1 usando os métodos aplicáveis?
  - A) Dois loops aninhados com Empregado como laço externo
  - B) Dois loops aninhados com Departamento como laço externo
    - $C_{J1} = b_R + (b_R * b_S) + ((js * |R| * |S|)/bfr_{RS})$
  - C) Laço único com Empregado usando estrutura de indexação para acessar Departamento
  - D) Laço único com Departamento usando estrutura de indexação para acessar Empregado
    - Para um índice secundário,
      - $C_{J2a} = b_R + (|R| * (x_B + s_B)) + ((js * |R| * |S|)/bfr_{RS})$;
    - Para um índice primário
      - $C_{J2c} = b_R + (|R| * (x_B + 1)) + ((js * |R| * |S|)/bfr_{\ \ })$;

Problema 2

- Op1
- A) 30500
- B) 28513
- C) 24500
- D) 12763

- Op2
- A)
- B)
- C)
- D)

Problema 3

- Supondo que há 15 buffers disponíveis, qual seria o melhor plano de execução?