EAD

EDUCAÇÃO A DISTÂNCIA

# UNIDADE II

## Linguagem de Programação de Banco de Dados

Profa. Dra. Vanessa Lessa

# Consulta de dados

- O resultado de muitas consultas é uma pequena proporção dos registros. Não é produtivo para o sistema ler cada registro e verificar se o campo ou campos de condições são válidos. O ideal é que o sistema consiga localizar os registros rapidamente.

- Um índice em arquivo funciona da mesma forma que o índice em um livro. Caso você precise localizar um determinado assunto em particular, procurando uma palavra ou uma frase, pode buscar o assunto no índice, verificar a página em que o assunto ocorre e depois ler as páginas indicadas para encontrar o assunto que você procura.

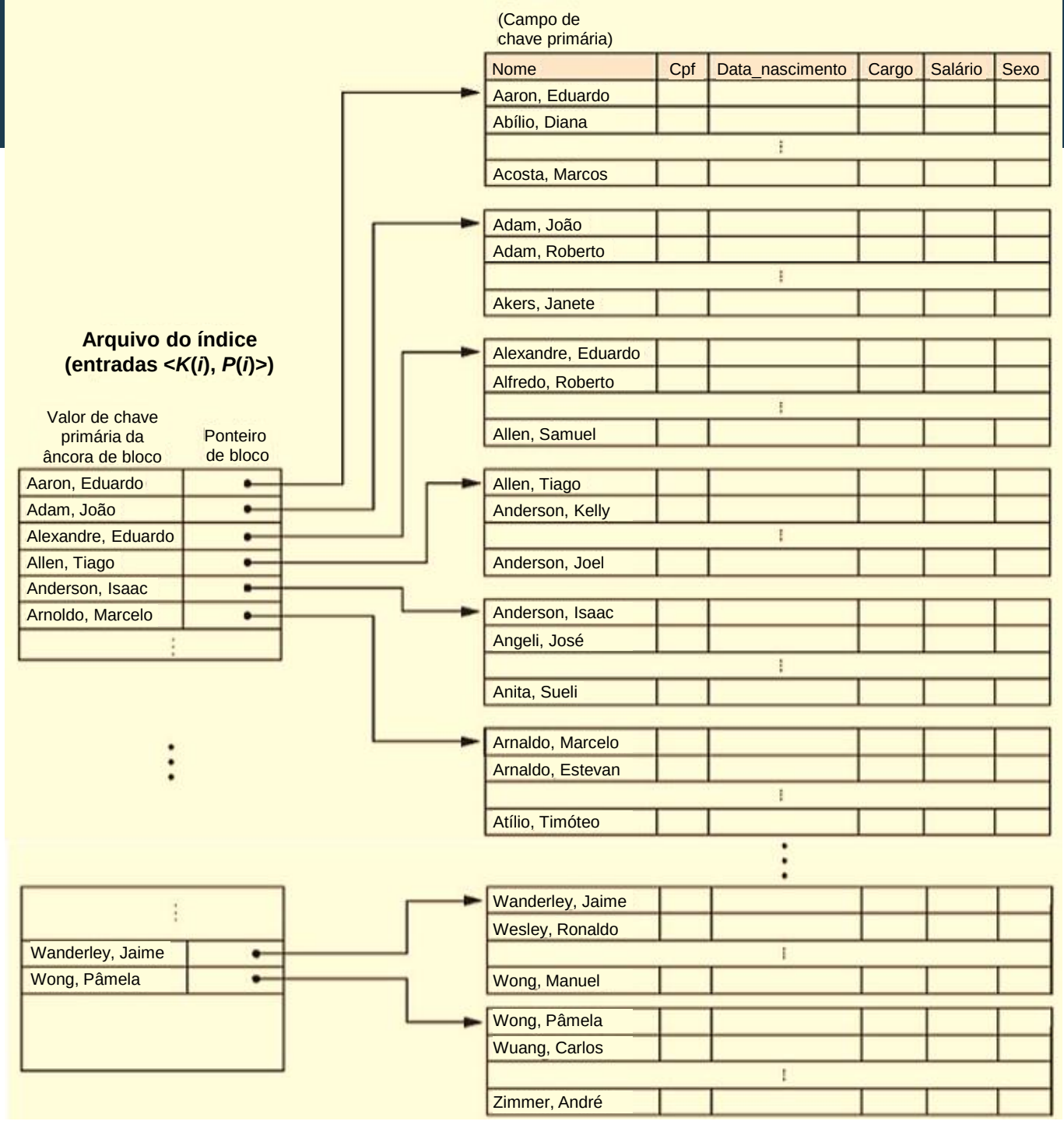
Existem dois tipos básicos de índices:

- Índices ordenados – utilizam uma ordem classificada dos valores;
- Índices hash – utilizam uma distribuição uniforme de valores por um intervalo de buckets (balde – o bucket utiliza uma função de hash para distribuir os valores).

# Índices ordenados

- Existem diversos tipos de índices ordenados. Um índice primário é definido no campo-chave de ordenação de um arquivo ordenado de registros.

Fonte: Adaptado de: ELMASRI, R.; NAVATHE, S. B. (2011, p. 426).



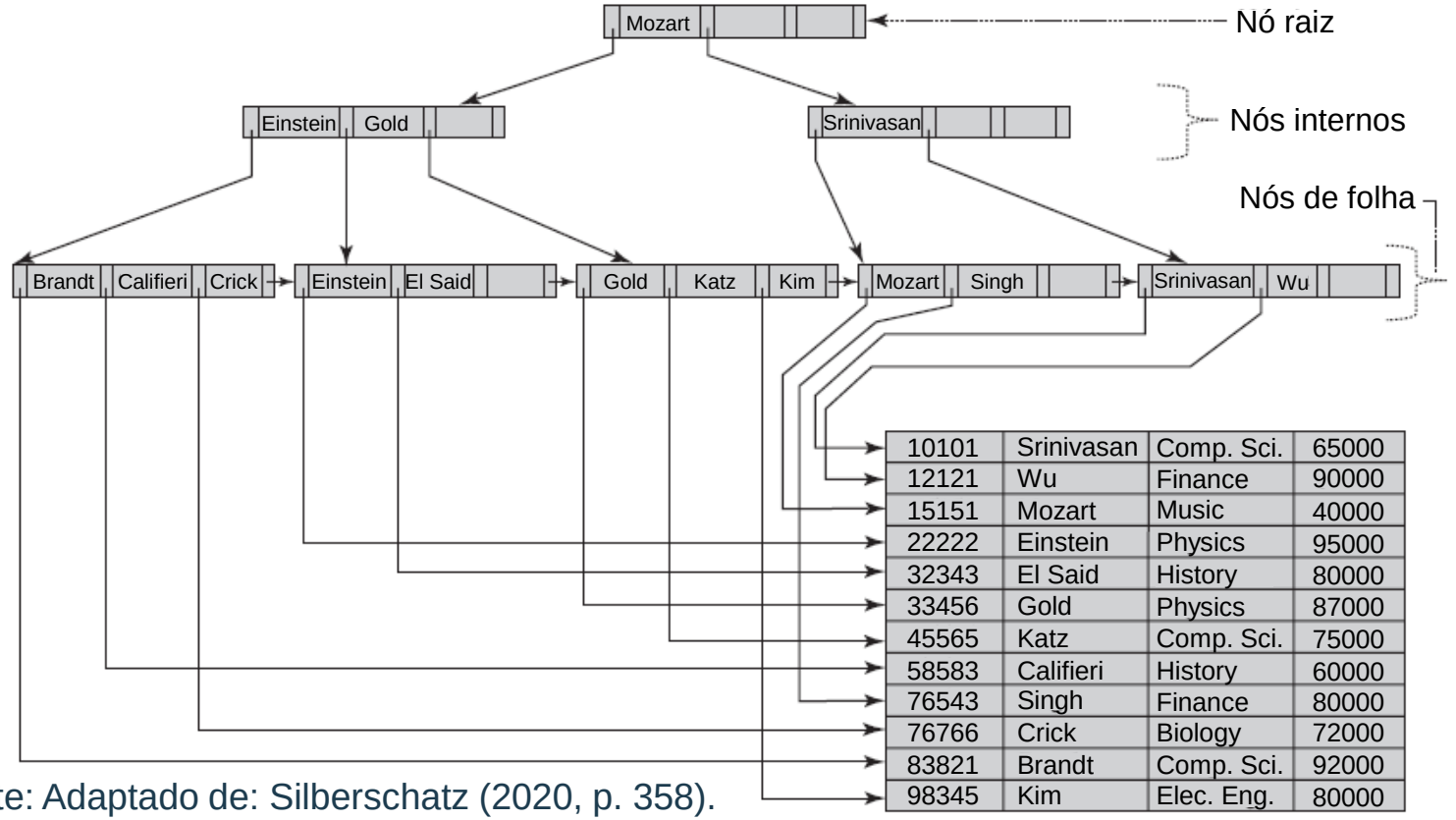
# Índices ordenados

- Os índices ordenados podem ser classificados como densos ou esparsos. Um índice denso tem uma entrada de índice para cada valor de chave no arquivo de dados. Um índice esparso (não denso) tem entradas de índice considerando apenas alguns valores de pesquisa, dessa forma, possui menos entradas do que o registro de arquivos como, por exemplo, o índice primário.

  - Quando criamos uma tabela, geralmente, não declaramos NONCLUSTERED em uma chave primária. A chave primária assume o valor de CLUSTERED, e, automaticamente, são criados os índices agrupados. Cada tabela deverá ter um e somente um índice de agrupamento (uma primary key). Quando criamos um índice de agrupamento, será necessário espaço em disco (aproximadamente, 1,2 vezes o tamanho atual da tabela). Depois desta operação, o espaço em disco é restaurado automaticamente.

# Arquivos de índice de árvore B+

- Na ciência da computação, uma árvore B+ é uma estrutura de dados do tipo árvore. A estrutura de índice de árvore B+ é a mais utilizada, pois mantém sua eficiência independentemente da inserção e exclusão dos dados. Possui a propriedade de uma árvore balanceada em que cada caminho raiz da árvore até uma folha da árvore possui o mesmo caminho, essa propriedade que assegura a boa *performance* para pesquisa, inserção e exclusão.




Fonte: Adaptado de: Silberschatz (2020, p. 358).

# Arquivos de índice de árvore B+

- Consulta em árvore B+

```
function find(v)
/* Assumindo que não há duplicatas, retorna o ponteiro para o registro com o
 * valor de chave de busca v se tal registro existir; se não, retorna nulo */
    Defina C = nó raiz
    while (C não é um nó de folha) begin
        Seja i = menor número tal que v ≤ C·K_i
        if esse número i não existir then begin
            Seja P_m = último ponteiro não nulo no nó
            Atribua C = C·P_m
        end
        else if (v = C·K_i) then Defina C = C·P_{i+1}
        else Defina C = C·P_i /* v < C·K_i */
    end
    /* C é o nó de folha */
    if para algum i, K_i = v
        then return P_i
        else return null; /* Não existe um registro com valor de chave v*/
```

Fonte: Silberschatz (2020, p. 360).

# Arquivos de índice de árvore B+

- Inserção em árvore B+

```
procedure insert (valor K, ponteiro P)
    if (árvore está vazia) crie um nó de folha L vazio, que também é a raiz
    else Encontre o nó de folha L que deve conter o valor de chave K
    if (L tem menos que n – 1 valores de chave)
        then insert_in_leaf (L, K, P)
        else begin /* L já tem n – 1 valores de chave, divida-o */
            Crie o nó L'
            Copie L·P1 ... L.Kn–1 para um bloco de memória T que possa
                armazenar n pares (ponteiro, valor-chave)
            insert_in_leaf (T, K, P)
            Atribua L'·Pn = L·Pn; Atribua L·Pn = L'
            Apague L·P1 até L.Kn–1 de L
            Copie T·P1 até T·K⌊n/2⌋ de T em L começando em L·P1
            Copie T·K⌊n/2⌋+1 até T·Kn de T em L' começando em L'·P1
            Seja K' o menor valor de chave em L'
            insert_in_parent(L, K', L')
    end
```

```
procedure insert_in_leaf (nó L, valor K, ponteiro P)
    if (K < L·K1)
        then insira P, K em L imediatamente antes de L·P1
        else begin
            Seja Ki o valor mais alto em L menor ou igual a K
            Insira P, K em L imediatamente após L·Ki
        end
procedure insert_in_parent(nó N, valor K', nó K')
    if (N é a raiz da árvore)
        then begin
            Crie um novo nó R contendo N, K', N'    /* N e N' são ponteiros*/
            Torne R a raiz da árvore
            return
        end
    Seja P = parent (N)
    if (P tem menos de n ponteiros)
        then insira (K', N') em P imediatamente após N
        else begin /* Divida P */
            Copie P para um bloco de memória T que possa armazenar P e (K', N')
            Insira (K', N') em T imediatamente após N
            Apague todas as entradas de P; Crie o nó P'
            Copie T·P1 ... T·P⌊(n+1)/2⌋ para P
            Defina K'' = T·K⌊(n+1)/2⌋
            Copie T·P⌊(n+1)/2⌋+1 ... T·Pn+1 para P'
            insert_in_parent(P, X'', P')
    end
```

Fonte: Silberschatz (2020, p. 363).

# Arquivos de índice de árvore B+

- Exclusão em árvore B+

```
procedure delete(valor K, ponteiro P)
    encontre o nó de folha L que contém (K, P)
    delete_entry(L, K, P)
```

```
procedure delete_entry(nó N, valor K, ponteiro P)
    delete (K, P) de N
    if (N é a raiz and N só tem um filho restante)
    then torne o filho de N a nova raiz da árvore e exclua N
    else if (N tem número insuficiente de valores/ponteiros) then begin
        Seja N' o filho anterior ou seguinte de parent(N)
        Seja K' o valor entre os ponteiros N e N' em parent(N)
        if (as entradas em N e N' couberem em um único nó)
            then begin  /* Una os nós */
                if (N é antecessor de N') then swap_variables(N, N')
                if (N não é uma folha)
                    then anexe K' e todos os ponteiros e valores existentes em N a N'
                    else anexe todos os pares (K_i, P_i) existentes em N a N'; atribua N'·P_n = N·P_n
                delete_entry(parent(N), K', N); exclua o nó N
            end
        else begin /* A redistribuição toma emprestada uma entrada de N' */
            if (N' é antecessor de N) then begin
                if (N é um nó não folha) then begin
                    seja m tal que N'.P_m seja o último ponteiro em N'
                    remova (N'·K_{m-1}, N'·P_m) de N'
                    insira (N'·P_m, K') como o primeiro ponteiro e valor em N,
                        deslocando outros ponteiros e valores para a direita
                    substitua K' em parent(N) por N'·K_{m-1}
                end
                else begin
                    seja m tal que (N'·P_m, N'·K_m) seja o último par de ponteiro/valor em N'
                    remova (N'·P_m, N'·K_m) de N'
                    insira (N'·P_m, N'·K_m) como o primeiro ponteiro e valor em N,
                    deslocando outros ponteiros e valores para a direita
                    substitua K' em parent(N) por N'·K_m
                end
            end
            else ... simétrico ao caso then ...
        end
    end
```

Fonte: Silberschatz (2020, p. 366).

# Interatividade

Analise as afirmações:

I. Índices ordenados utilizam uma distribuição uniforme de valores por um intervalo de buckets.

II. Índices hash utilizam uma ordem classificada dos valores.

III. Uma árvore B+ possui a propriedade de uma árvore balanceada.

IV. Os índices ordenados podem ser classificados como densos ou esparsos.

V. Um índice denso tem uma entrada de índice para cada valor de chave no arquivo de dados.

Está(ão) correta(s):

a) I, II e III.

b) III e IV.

c) I, II e V.

d) III, IV e V.

e) Todas as afirmações.

# Resposta

Analise as afirmações:

I. Índices ordenados utilizam uma distribuição uniforme de valores por um intervalo de buckets.

II. Índices hash utilizam uma ordem classificada dos valores.

III. Uma árvore B+ possui a propriedade de uma árvore balanceada.

IV. Os índices ordenados podem ser classificados como densos ou esparsos.

V. Um índice denso tem uma entrada de índice para cada valor de chave no arquivo de dados.

Está(ão) correta(s):

a) I, II e III.

b) III e IV.

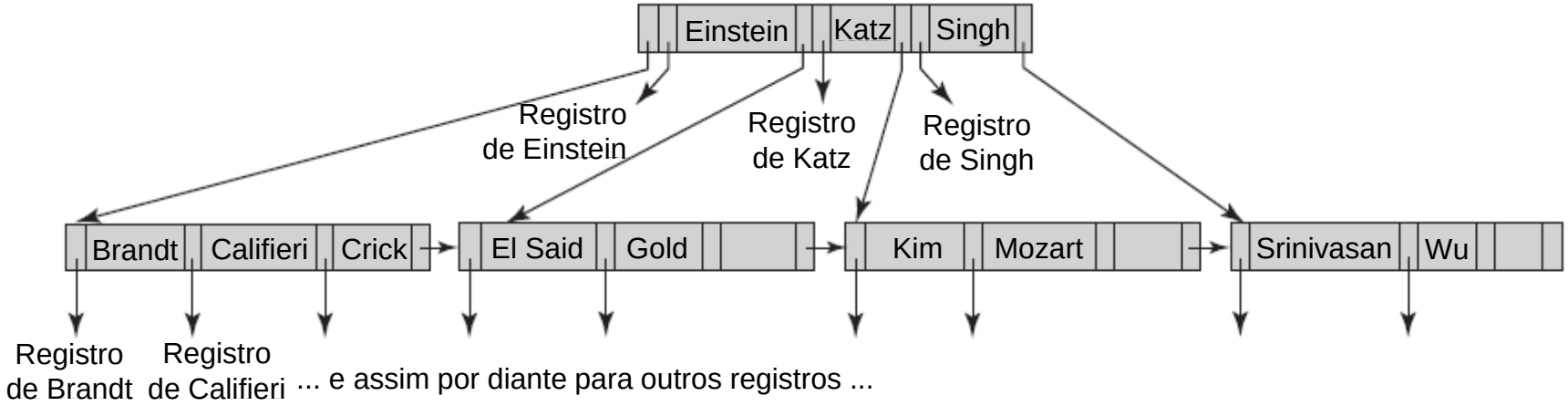
c) I, II e V.

d) III, IV e V.

e) Todas as afirmações.

# Arquivos de índice de árvore B

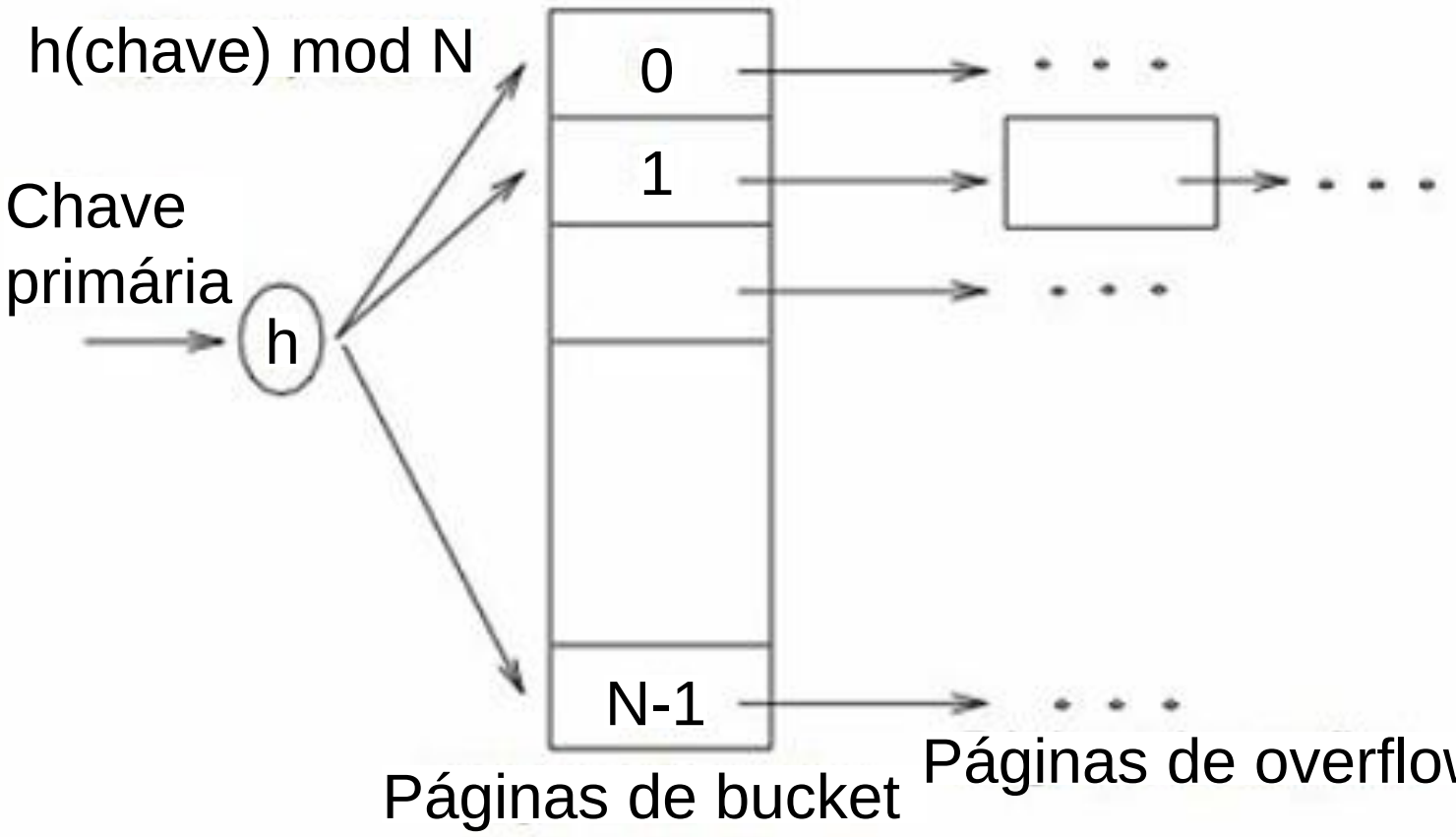
- Na ciência da computação, uma árvore B é uma estrutura de dados semelhante a uma árvore balanceada que armazena dados ordenados e permite a pesquisa. Uma árvore B é adequada para sistemas de armazenamento que leem e gravam blocos de dados relativamente grandes. A ideia básica por trás das árvores B é trabalhar com dispositivos de armazenamento secundários, quanto menos espaço em disco uma estrutura de dados fornecer, melhor será o desempenho do sistema na recuperação de dados manipulados.




Fonte: Adaptado de: Silberschatz (2020, p. 370).

# Hashing estático

- As páginas que contêm dados podem ser observadas como uma coleção buckets, com uma página primária e páginas de overflow adicionadas. O arquivo consiste em 0 até N-1 buckets e inicialmente uma página principal por bucket.




Fonte: Adaptado de: RAMAKRISHNAN (2008, p. 335).

# Hashing estático

- Para consulta de uma entrada de dados, utilizamos uma função hash h para localizar o bucket ao qual ela pertence e, em seguida, pesquisamos este bucket. Para melhorar a consulta de um bucket, podemos utilizar entradas de dados ordenadas pelo valor da chave de pesquisa.

- Para inserir uma entrada de dados, utilizamos a função hash para localizar o bucket certo e depois adicionamos a entrada de dados. Caso o espaço não seja suficiente para esta entrada de dados, alocamos uma nova página de overflow e adicionamos a entrada de dados na nova página e a página à cadeia de overflow do bucket.

- Para excluir uma entrada de dados, utilizamos a função de hashing para localizar o bucket correto, identificamos a entrada de dados consultando o bucket e o removemos. Caso essa entrada de dados seja a última em uma página de overflow, esta página de overflow é removida da cadeia de overflow do bucket e inserida em uma lista de páginas livres.

# Hashing estático

- A função hash é um item muito importante da abordagem hashing. Ele deve distribuir valores no domínio do campo de pesquisa uniformemente sobre uma coleção de buckets. Se tivermos N buckets, numerados de 0 até́ N – 1, uma função hash h da forma h(valor)=(a * valor + b) funciona bem na prática. (O bucket identificado é h(valor) mod N.) As constantes a e b podem ser selecionadas para corrigir a função hash, consultamos todas as páginas na sua cadeia de overflow, é fácil prever como o desempenho pode se deteriorar. Mantendo primeiro 80% das páginas cheias, podemos evitar páginas de overflow caso o arquivo não cresça muito, mas, geralmente, a única maneira de se libertar de cadeias de overflow é criando um novo arquivo com mais buckets.

  - O maior problema com o hashing estático é que o número de buckets é fixo. Caso um arquivo diminua bastante, teremos muito espaço inútil. Agora, se um arquivo aumentar bastante, longas cadeias de overflow se desenvolvem, resultando em baixo desempenho.

# Hashing dinâmico

A maior parte dos bancos de dados utilizados cresce muito ao longo do tempo. Se pretendemos usar hashing estático nesses bancos de dados, temos três classes de opções:

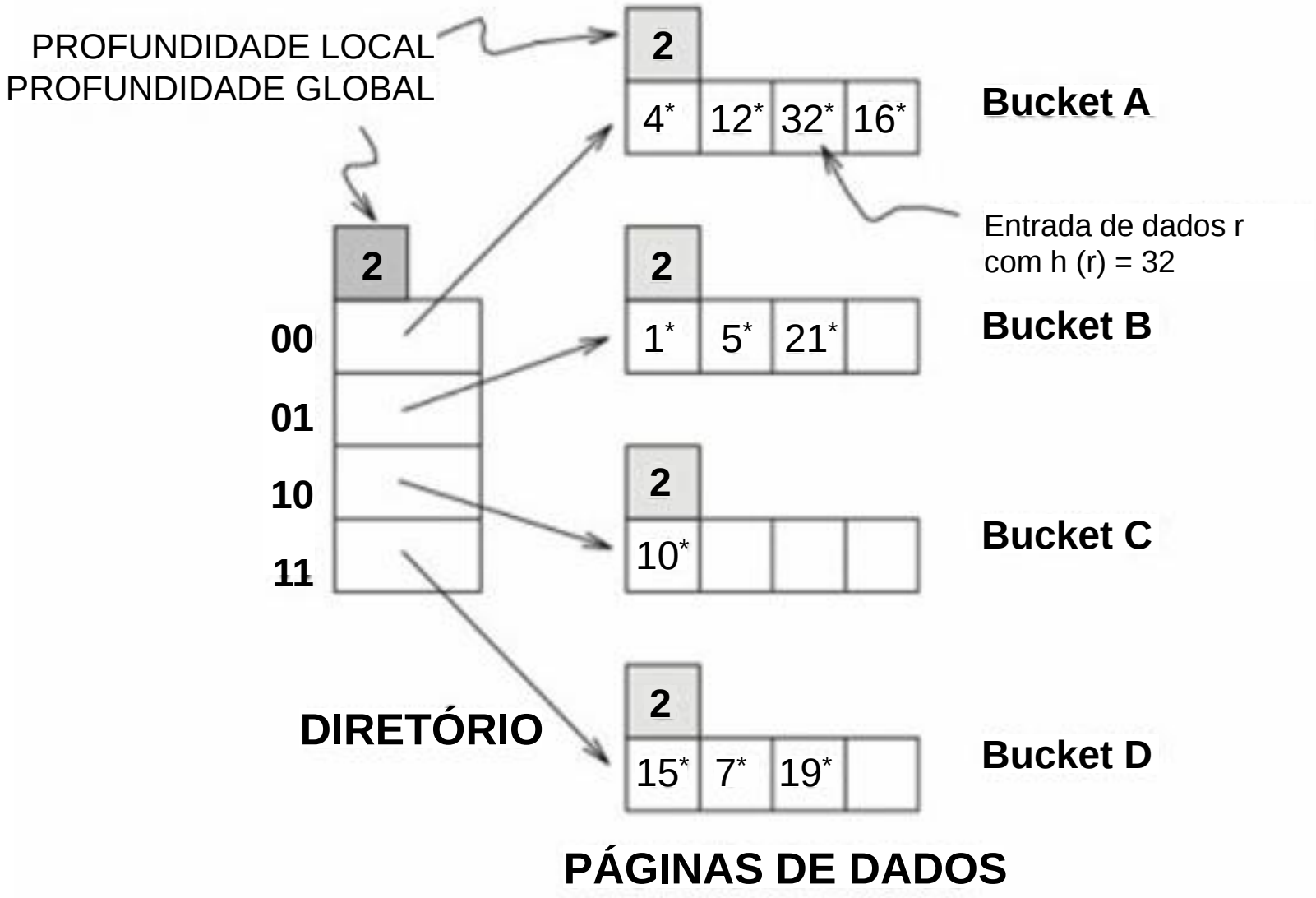
1. Escolha uma função hash baseada no tamanho atual do arquivo. Essa opção diminui o desempenho à medida que o banco de dados cresce.
2. Escolha uma função de hash com base no tamanho esperado do arquivo no futuro. Embora a degradação do desempenho seja evitada, uma quantidade significativa de espaço pode ser perdida inicialmente.
3. Reorganize periodicamente a estrutura de hash à medida que o arquivo cresce. Essa reorganização envolve escolher uma nova função de hash, recalcular a função de hash para cada registro no arquivo e criar definições de grupo. Essa reorganização é uma operação em larga escala e demorada. Além disso, é necessário negar o acesso ao arquivo durante a reorganização.
   - Várias técnicas de hash dinâmico permitem que a função de hash seja alterada dinamicamente para acomodar a expansão ou contração do banco de dados.

# Hashing dinâmico

- Hashing extensível
- Analisar um arquivo de hash estático para entender melhor hash extensível. Caso precise adicionar uma entrada de dados em um bucket carregado, vamos precisar inserir uma página de overflow. Caso não deseje adicionar mais páginas de overflow, será necessário a reorganização do arquivo neste ponto dobrando o número de buckets e redistribuindo as entradas no novo conjunto de buckets. Todo o arquivo precisa ser lido, e duas vezes mais páginas terão que ser gravadas para se conseguir a reorganização, causando assim uma deficiência. Este problema pode ser resolvido com uma solução simples: podemos usar um diretório de ponteiros para buckets, e duplique o tamanho do número de buckets duplicando assim apenas o diretório e dividindo apenas o bucket que sofreu o overflow.

# Hashing dinâmico

- Hashing extensível.




Fonte: Adaptado de: RAMAKRISHNAN (2008, p. 336).

# Interatividade

Analise as afirmações:

I. Uma árvore B é adequada para sistemas de armazenamento que leem e gravam blocos de dados relativamente grandes.

II. Para melhorar a consulta de um bucket, podemos utilizar entradas de dados ordenadas pelo valor da chave de pesquisa.

III. As técnicas de hash dinâmico não permitem que a função de hash seja alterada dinamicamente para acomodar a expansão ou contração do banco de dados.

IV. O maior problema com o hashing estático é que o número de buckets pode variar.

V. No hashing extensível, podemos usar um diretório de ponteiros para buckets e duplicar o tamanho do número de buckets.

Está(ão) correta(s):

a) I, II e III.

b) I, II e IV.

c) I, II e V.

d) I, III, IV e V.

e) Todas as afirmações.

# Resposta

Analise as afirmações:

I. Uma árvore B é adequada para sistemas de armazenamento que leem e gravam blocos de dados relativamente grandes.

II. Para melhorar a consulta de um bucket, podemos utilizar entradas de dados ordenadas pelo valor da chave de pesquisa.

III. As técnicas de hash dinâmico não permitem que a função de hash seja alterada dinamicamente para acomodar a expansão ou contração do banco de dados.

IV. O maior problema com o hashing estático é que o número de buckets pode variar.

V. No hashing extensível, podemos usar um diretório de ponteiros para buckets e duplicar o tamanho do número de buckets.

Está(ão) correta(s):

a) I, II e III.

b) I, II e IV.

c) I, II e V.

d) I, III, IV e V.

e) Todas as afirmações.

# Otimização da consulta

- Otimização da consulta é o método de selecionar o melhor plano de avaliação para consultas mais eficientes considerando as diversas estratégias geralmente possíveis para o processamento de uma consulta, especialmente se ela for complexa. Não queremos que os usuários digitem suas consultas para processá-las com eficiência. Entretanto, contamos com o sistema para produzir um plano de avaliação da pesquisa que minimize o custo da avaliação da pesquisa. Nesse momento, é que a otimização de consultas se torna necessária.
    - Uma característica da otimização acontece no nível da álgebra relacional, em que o sistema tenta identificar uma expressão que seja equivalente a uma dada expressão, mas que seja mais eficiente de executar. Outra característica é a escolha de uma estratégia detalhada para processar a consulta, como a decisão pelo algoritmo a ser utilizado para realizar a operação, a escolha dos índices específicos a serem utilizados etc.

# Processamento da consulta

- O processamento de consultas refere-se ao conjunto de operações envolvidas na recuperação de dados de um banco de dados. Os recursos incluem a tradução de consultas em linguagens de banco de dados de alto nível em expressões que podem ser usadas no nível do sistema de arquivos físico, transformações de otimização de pesquisa e avaliação de consultas em tempo real.

As fases no processamento de uma consulta são:

1. Análise e tradução.
2. Otimização.
3. Avaliação.

- Antes de processar a solicitação, o sistema deve traduzir a solicitação em um formato útil. A linguagem SQL é boa para uso humano, mas não para consulta no sistema. Uma representação interna mais útil é aquela baseada em álgebra relacional.

# Processamento da consulta

- Depois que uma pesquisa é realizada, geralmente, há vários métodos para calcular a resposta. Por exemplo, uma consulta pode ser expressa de diferentes maneiras no SQL. Qualquer consulta SQL pode ser traduzida em expressões de álgebra relacional de várias maneiras. Além disso, a representação algébrica relacional de uma seleção determina apenas parcialmente como a seleção é avaliada; geralmente, há várias maneiras de avaliar expressões de álgebra de relação. Por exemplo, considere o comando SELECT:

```
SELECT salario
FROM Funcionario
WHERE salario < 7000
```

Podemos traduzir essa consulta para uma das seguintes expressões da álgebra relacional:

$$\sigma_{salario} < 7000 (\pi_{salario} (funcionario))$$

$$\pi \text{ salario } ( \sigma \text{ salario} < 7000 (funcionario))$$

# Medidas de custo da consulta

- A consulta tem vários planos de avaliação possíveis, sendo importante poder comparar as opções de acordo com os seus custos previstos e escolher o melhor plano. Para isso, precisamos estimar os custos de cada atividade individualmente e adicioná-los aos custos do plano de avaliação da consulta.

- O custo de avaliação de uma consulta pode ser estimado analisando vários recursos diferentes, como o uso de disco, o tempo de CPU necessário para executar a consulta e custos de comunicação em um sistema de banco de dados distribuído paralelo.

# Medidas de custo da consulta

Se os dados estiverem na memória ou em SSDs, os custos de E/S não afetam o custo total, portanto, precisamos considerar o custo da CPU ao calcular o custo de avaliação da consulta. Podem estimar utilizando estimativas simples, como:

1. Custo de CPU por tupla;
2. Custo de CPU para processar cada entrada de índice (além do custo de E/S);
3. Custo de CPU por operador ou função (como operadores aritméticos, operadores de comparação e funções relacionadas).

- O banco de dados possui valores determinados para cada custo, que são multiplicados pelo número de tuplas processadas, o número de itens de índice processados e o número de operadores e operações realizadas.

# Medidas de custo da consulta

- Ao avaliar o custo de um plano de implementação, usamos o número de transferências em bloco do local de armazenamento e o número de ocorrências de E/S aleatórias, cada uma exigindo uma busca em disco do meio de armazenamento magnético, como dois fatores importantes. Se um subsistema de disco leva uma média de tT segundos para mover um bloco de dados e tem um tempo médio de acesso ao bloco (tempo de busca do disco mais latência rotacional) de tS segundos, então uma função que move b blocos e executa S ocorrências de E/S aleatórios utilizará b × tT + S × tS segundos.

# Medidas de custo da consulta

O tempo necessário para executar um plano de avaliação de consultas, supondo que não haja outras operações no computador, refletiria todos esses custos e poderia ser usado como uma medida do custo do plano.  Infelizmente, é muito difícil estimar o tempo de resposta de um plano sem executá-lo por dois motivos:

1. O tempo de resposta vai depender do conteúdo do buffer quando a consulta iniciar a execução; esse dado não está à disposição quando a consulta é otimizada, e é complicado de ser considerada mesmo que ela esteja disponível.

2. Em um sistema com diversos discos, o tempo de resposta é baseado em como os acessos são distribuídos entre os discos, para isso, precisamos conhecer o *layout* dos dados no disco.

# Interatividade

Analise as afirmações:

I. Otimização da consulta é o método de selecionar o melhor plano de avaliação para consultas mais eficientes considerando as diversas estratégias.

II. O processamento de consultas refere-se ao conjunto de operações envolvidas na recuperação de dados de um banco de dados.

III. Qualquer consulta SQL pode ser traduzida em expressões de álgebra relacional de várias maneiras.

IV. Se os dados estiverem na memória ou em SSDs, os custos de E/S não afetam o custo total.

V. As fases no processamento de uma consulta são: análise e tradução, otimização e avaliação.

Está(ão) correta(s):

a) I, II e III.

b) I, II e IV.

c) II, III e V.

d) III, IV e V.

e) Todas as afirmações.

# Resposta

Analise as afirmações:

I. Otimização da consulta é o método de selecionar o melhor plano de avaliação para consultas mais eficientes considerando as diversas estratégias.
II. O processamento de consultas refere-se ao conjunto de operações envolvidas na recuperação de dados de um banco de dados.
III. Qualquer consulta SQL pode ser traduzida em expressões de álgebra relacional de várias maneiras.
IV. Se os dados estiverem na memória ou em SSDs, os custos de E/S não afetam o custo total.
V. As fases no processamento de uma consulta são: análise e tradução, otimização e avaliação.

Está(ão) correta(s):

a) I, II e III.
b) I, II e IV.
c) II, III e V.
d) III, IV e V.
e) Todas as afirmações.

# Operação de seleção

- Ao processar uma consulta, uma varredura de arquivo é o operador de processamento de dados de nível mais baixo. As varreduras de arquivos são algoritmos de pesquisa que procuram e recuperam registros que correspondem a uma condição de seleção. Em sistemas relacionais, a varredura de arquivos permite que toda a relação seja lida se a relação estiver armazenada em um arquivo separado.

# Operação de seleção

## Seleções usando varreduras de arquivo e índices

Considere uma operação de seleção para uma tabela cujos registros são armazenados em um único arquivo. O algoritmo de varredura mais óbvio para implementar a função de seleção é:

- A1 (busca linear) – Em uma busca linear, o sistema varre cada bloco do arquivo e testa todos os registros para ver se eles correspondem aos critérios de seleção. A primeira busca é necessária para acessar o primeiro bloco do arquivo. Buscas adicionais podem ser necessárias se os blocos de arquivos não estiverem armazenados próximos uns dos outros, mas ignoramos esse efeito para facilitar.

    - Ainda que a seleção possa ser mais lenta para aplicar do que outros algoritmos, o algoritmo de busca linear pode ser aplicado a qualquer arquivo, independentemente da classificação do arquivo ou da disponibilidade de índices ou da natureza da operação de seleção.

# Operação de seleção

Os algoritmos de pesquisa que usam um índice são chamados de varreduras de índice. Usamos o predicado de seleção para nos orientar na melhor escolha do índice a ser utilizado ao processar a consulta. Os algoritmos de pesquisa que utilizam o índice são:

- A2 (índice agrupado, igualdade sobre chave). Para uma comparação de igualdade entre um atributo de chave e um índice primário, podemos utilizar o índice para pegar um único registro que atenda à condição de igualdade. As estimativas de custo estão na figura 40. Para moldar a situação comum em que os nós internos do índice estão no buffer na memória, hi é definido como um.

# Operação de seleção

- A3 (índice agrupado, igualdade sobre não chave). Podemos separar diversos registros utilizando um índice primário quando a condição da seleção especifica uma validação de igualdade sobre um atributo não chave, A. A única diferença de A2 é que diversos registros podem ser separados. Contudo, os registros seriam guardados de forma sequencial no arquivo, pois o arquivo utiliza uma chave de busca para classificação.

- A4 (índice secundário, igualdade). As seleções que especificam uma condição de igualdade podem utilizar um índice secundário. Podemos separar um único registro se a condição de igualdade for sobre uma chave; diversas tuplas podem ser separadas se o campo de indexação não for uma chave.

# Operação de junção

- Para calcular a junção (r ⋈θ s) de duas relações r e s. Denominamos de algoritmo de junção por loop aninhado, pois consiste em um par de loops for aninhado. Denominamos a relação r de relação externa e a relação s de relação interna da junção, uma vez que, observando o algoritmo, o loop para a relação r encobre o loop para s. O algoritmo usa a notação para as tuplas tr e ts e, para indicar a tupla resultante da concatenação dos atributos, utiliza tr · ts.

**for each** tupla $t_r$ **in** $r$ **do begin**
  **for each** tupla $t_s$ **in** $s$ **do begin**
    testar par ($t_r$, $t_s$) para ver se satisfaz a condição da junção θ
    se satisfizer, acrescentar $t_r \cdot t_s$ ao resultado;
  **end**
**end**

Fonte: Silberschatz (2020, p. 399).

# Operação de junção

## Junção por loop aninhado em bloco

- Caso o buffer seja pequeno para receber qualquer relação completa na memória, ainda podemos obter economias significativas no acesso ao bloco, manipulando as relações bloco a bloco, em vez de tupla.

```
for each bloco B_r of r do begin
    for each bloco B_s of s do begin
        for each tupla t_r in B_r do begin
            for each tupla t_s in B_s do begin
                testar par (t_r, t_s) para ver se satisfaz a condição de junção
                se satisfizer, acrescente t_r · t_s ao resultado;
            end
        end
    end
end
```

Fonte: Silberschatz (2020, p. 399).

# Plano de avaliação de consulta

O plano de avaliação de consulta constitui-se de uma árvore de álgebra relacional estendida, com anotações adicionais em cada nó informando os métodos de acesso a serem utilizados por cada tabela e o método de execução de cada operador relacional. Vamos, considere a seguinte consulta SQL:

```
SELECT  M.nome-marin
FROM    Reservas R, Marinheiros M
WHERE   R.id-marin = M.id-marin
        AND  R.id_barco = 100 AND M.avaliação > 5
```

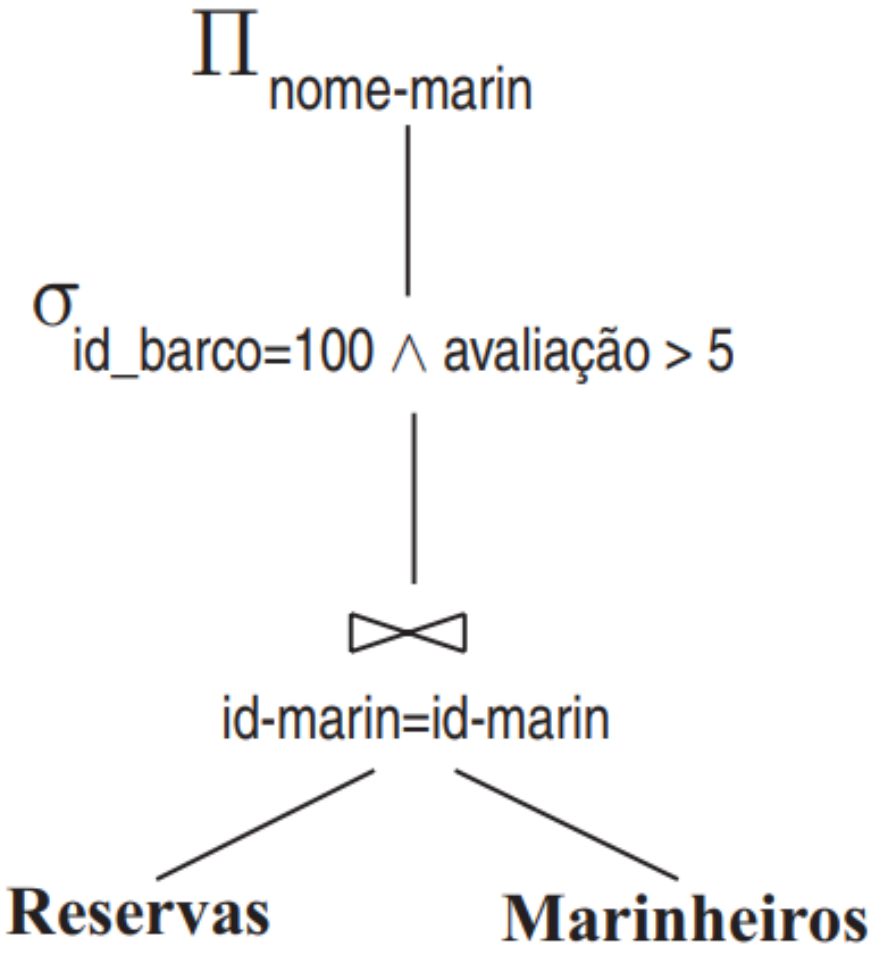
Podemos expressar a consulta utilizando álgebra relacional:

$$\pi_{nome\text{-}marin}(\sigma_{id\text{-}barco=100 \wedge avaliação>5}(Reservas \bowtie_{id\text{-}marin=id\text{-}marin} Marinheiros))$$

Fonte: RAMAKRISHNAN (2008, p. 362).

# Plano de avaliação de consulta

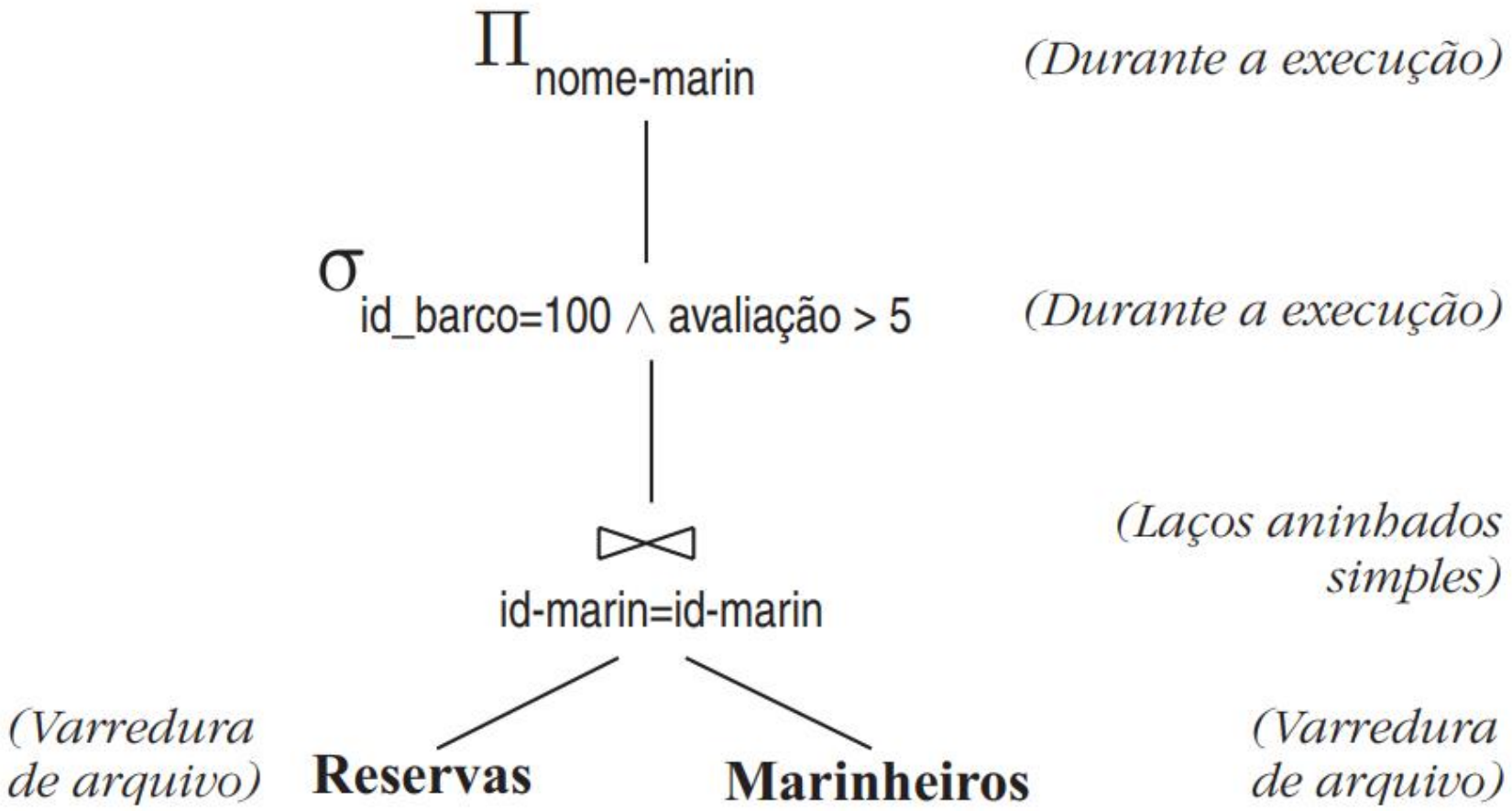
- Consulta expressa como uma árvore de álgebra relacional.




Fonte: RAMAKRISHNAN (2008, p. 362).

# Plano de avaliação de consulta

- Plano de avaliação de consultas para consulta exemplo.




Fonte: RAMAKRISHNAN (2008, p. 362).

# Interatividade

Analise as afirmações:

I. As varreduras de arquivos são algoritmos de pesquisa que procuram e recuperam registros que não correspondem a uma condição de seleção.
II. Em uma busca linear, o sistema varre cada bloco do arquivo e testa todos os registros para ver se eles são diferentes dos critérios de seleção.
III. Para uma comparação de igualdade entre um atributo de chave e um índice primário, podemos utilizar o índice para pegar um único registro que atenda à condição de igualdade.
IV. O plano de avaliação de consultas constitui-se de uma árvore de álgebra relacional estendida.
V. Caso o buffer seja pequeno para receber qualquer relação completa na memória, ainda podemos obter economias significativas no acesso ao bloco, manipulando as relações bloco a bloco, em vez de tupla.

Está(ão) correta(s):

a) I, II e III.
b) I, II e IV.
c) I, II e V.
d) III, IV e V.
e) Todas as afirmações.

# Resposta

Analise as afirmações:

I. As varreduras de arquivos são algoritmos de pesquisa que procuram e recuperam registros que não correspondem a uma condição de seleção.
II. Em uma busca linear, o sistema varre cada bloco do arquivo e testa todos os registros para ver se eles são diferentes dos critérios de seleção.
III. Para uma comparação de igualdade entre um atributo de chave e um índice primário, podemos utilizar o índice para pegar um único registro que atenda à condição de igualdade.
IV. O plano de avaliação de consultas constitui-se de uma árvore de álgebra relacional estendida.
V. Caso o buffer seja pequeno para receber qualquer relação completa na memória, ainda podemos obter economias significativas no acesso ao bloco, manipulando as relações bloco a bloco, em vez de tupla.

Está(ão) correta(s):

a) I, II e III.
b) I, II e IV.
c) I, II e V.
d) III, IV e V.
e) Todas as afirmações.

# Referências


- ELMASRI, R.; NAVATHE, S. B. *Sistemas de banco e dados*. 6 ed. São Paulo: Pearson, 2011.
- RAMAKRISHNAN, Raghu; Johannes Gehrke. *Sistemas de gerenciamento de bancos de dados*. 3 ed. New York: McGraw-Hill, 2008.
- SILBERSCHATZ, A; SUDARSHAN, H. F. *Sistema de banco de dados*. Rio de Janeiro: Elsevier, 2020.

# ATÉ A PRÓXIMA!