

Ciência da Computação - Banco de Dados II - IFC Videira

Indexação

Apresentação por Gabriel, Gabrielle e Guilherme





O que é indexação?

- Estrutura auxiliar de acesso;
- Arquivos adicionais que oferecem **formas alternativas** de acessar registros.

Conceito

- Agiliza a recuperação de registros e resposta a **pesquisas**;
- Usa uma base de condição, levando a um ponteiro de blocos;
- SQL Server cria uma estrutura de dados com as informações que fazem parte do índice;
- **INSERTs**, **DELETEs** e **UPDATEs** deixam os índices menos eficientes.




Para indexação eficientes:

**As melhores colunas são do tipo inteiro,
exclusivas e não nulas.**

**Índices filtrados para colunas com muitos
valores nulos e/ou subconjuntos definidos
de dados.**

**Tabelas muito pequenas podem não trazer
benefícios.**



Ordenados de único nível

- Lista termos importantes ao final;
- Ordem alfabética com uma lista de números;
- Buscando certo termo encontramos uma lista de endereços, usando-a para localizar as tabelas especificadas;
- **Única indicação exata;**
- Armazena-se cada valor do seu campo com uma lista de ponteiros para todos os blocos com registros;
- São ordenados de modo que é possível realizar uma pesquisa binária no índice;

Índices primários

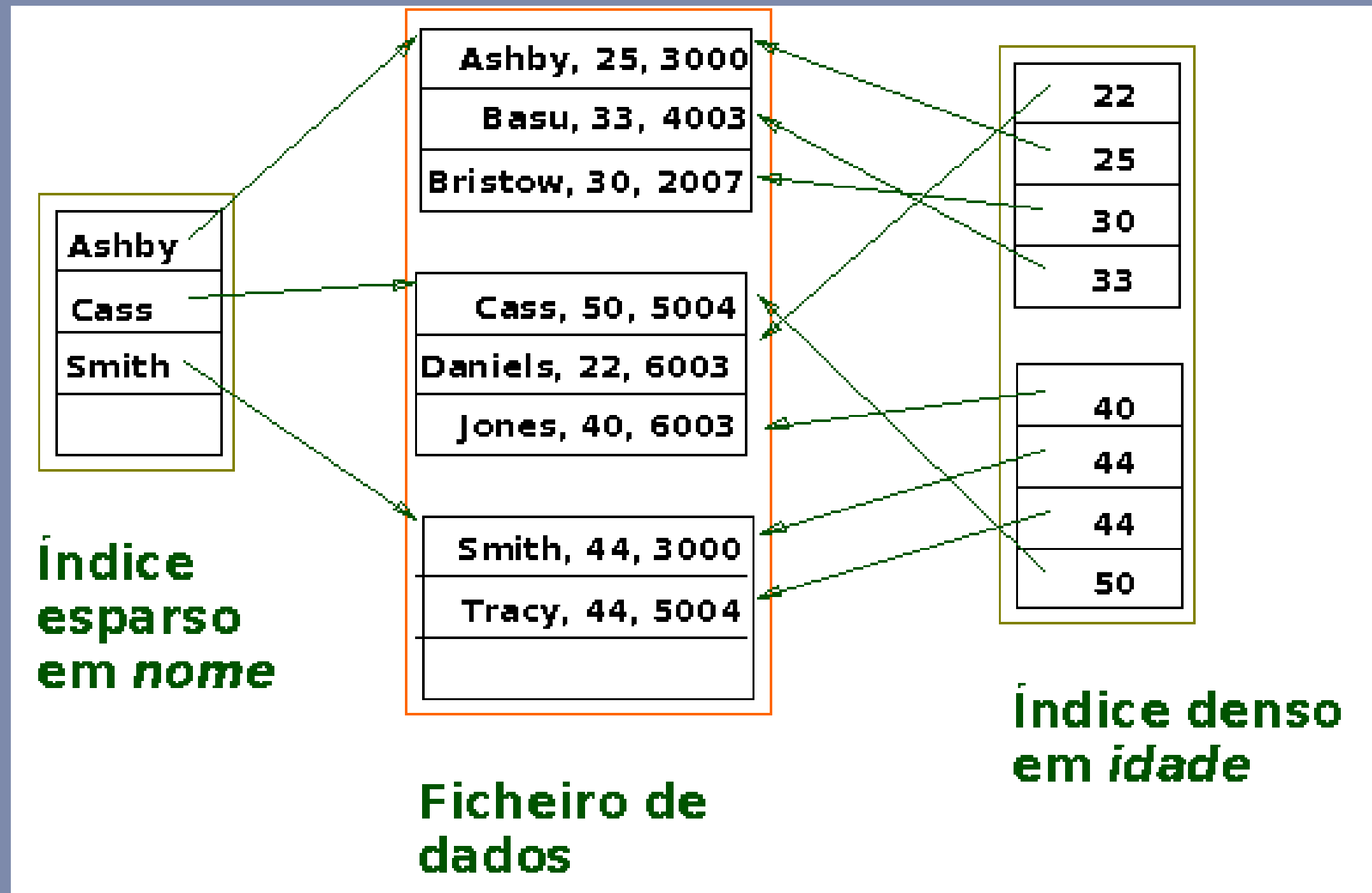
- Possuem tamanho fixo com **dois campos**:
 - *chave primária e ponteiro.*
- Estrutura de acesso para procurar e acessar registros de dados;
- **Problema importante**, é a inserção e exclusão de registros;
- Usar um arquivo de overflow desordenado pode resolver esse tipo de questão;
- Índice esparsos;

Índices Densos

- Uma entrada para cada valor de chave de pesquisa;
- Sua principal vantagem é a rapidez

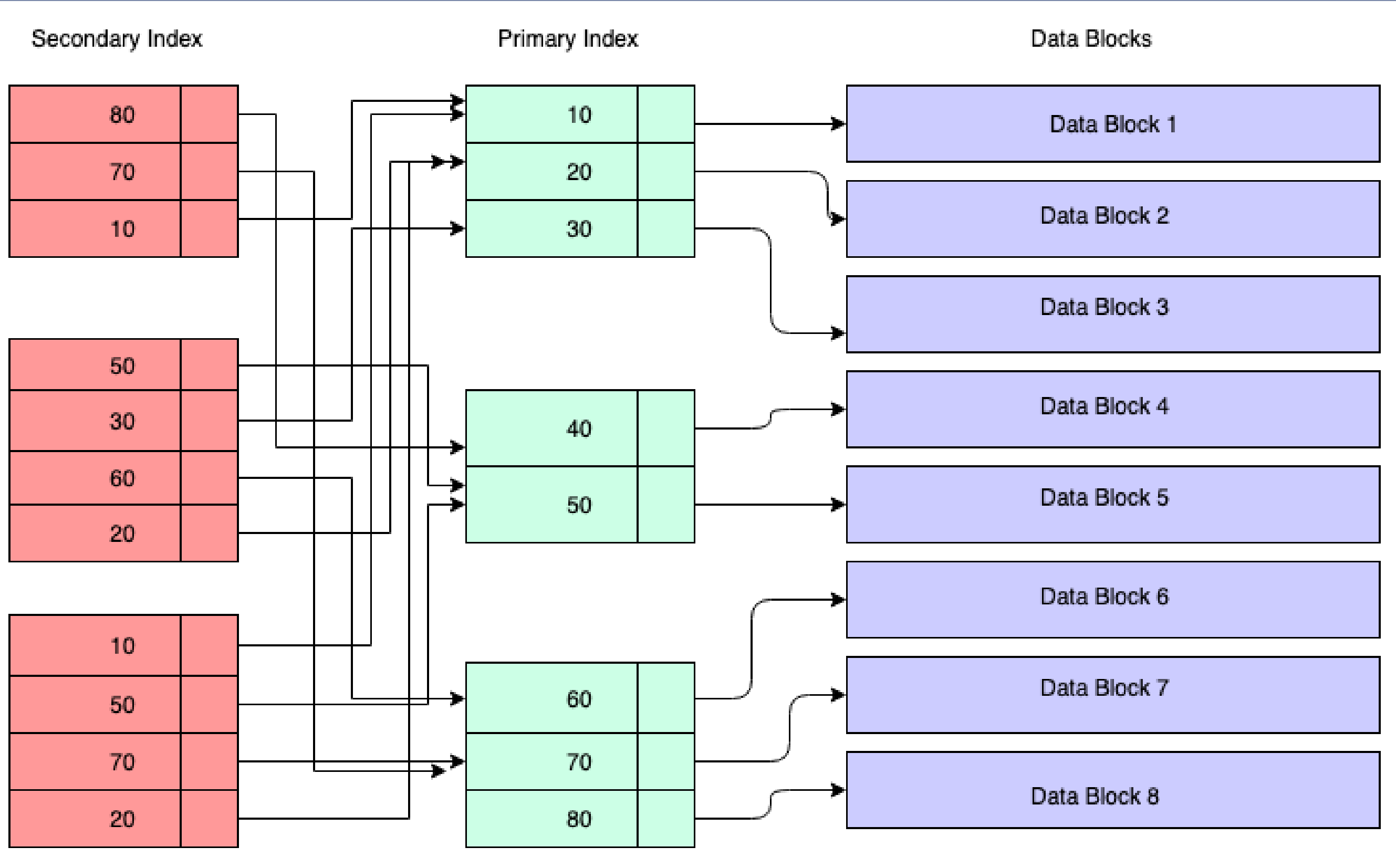
Índices Esparsos

- Possui entradas para somente alguns dos valores;
- Menor espaço e a menor sobrecarga de manutenção de INSERTs e DELETEs;



Índices secundários

- Meio secundário de acesso a arquivo, no qual há um acesso primário já existente;
- Ordenado com **dois campos**:
 - *mesmo tipo de dado de algum campo não ordenado*;
 - *ponteiro de bloco ou registro*;
- Podem ser criados para o mesmo arquivo;
- Índice denso;
- Ordenação lógica;
- Precisa de armazenamento e tempo de busca maiores;



Índices de agrupamento

- Arquivos fisicamente ordenados em um campo não chave, os quais não possuem valor distinto para cada registro;
- Podemos criar um índice de agrupamento, agilizando a recuperação de registros que **têm o mesmo valor**;
- Esparso;
- Possui valores duplicados.
- Índice primário pode ser chamando de índice de agrupamento;

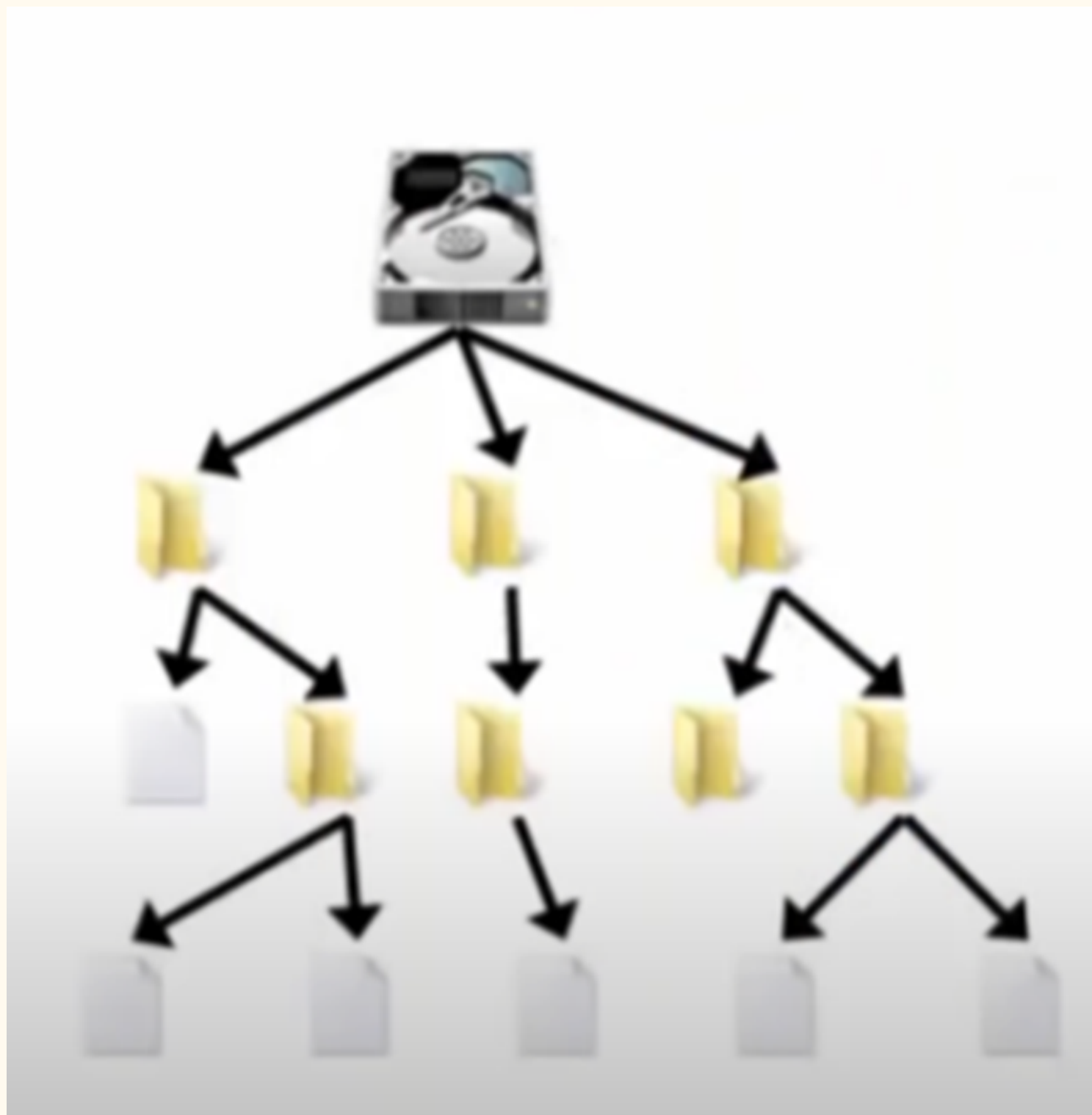
Um arquivo pode possuir



**Máximo um índice
primário ou um índice
de agrupamento.**



**Vários índices
secundários.**



Árvores Binárias

Aplicações práticas

01 Relação de descendência;

02 Relação hierárquica.

Pai

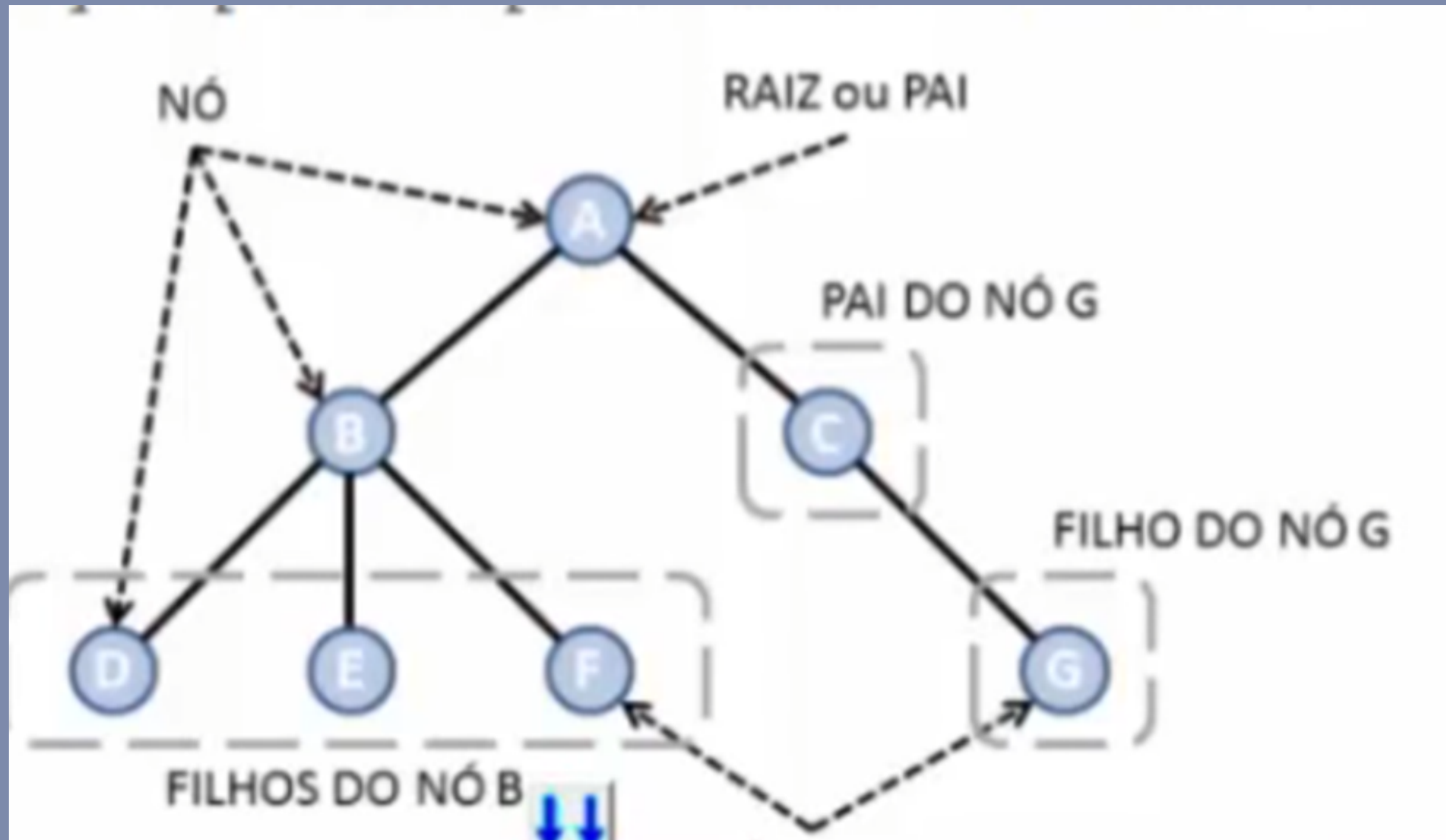
Filho

Raiz

Folha

Propriedades

Propriedades



Implementação da árvore

**Criação da
árvore**

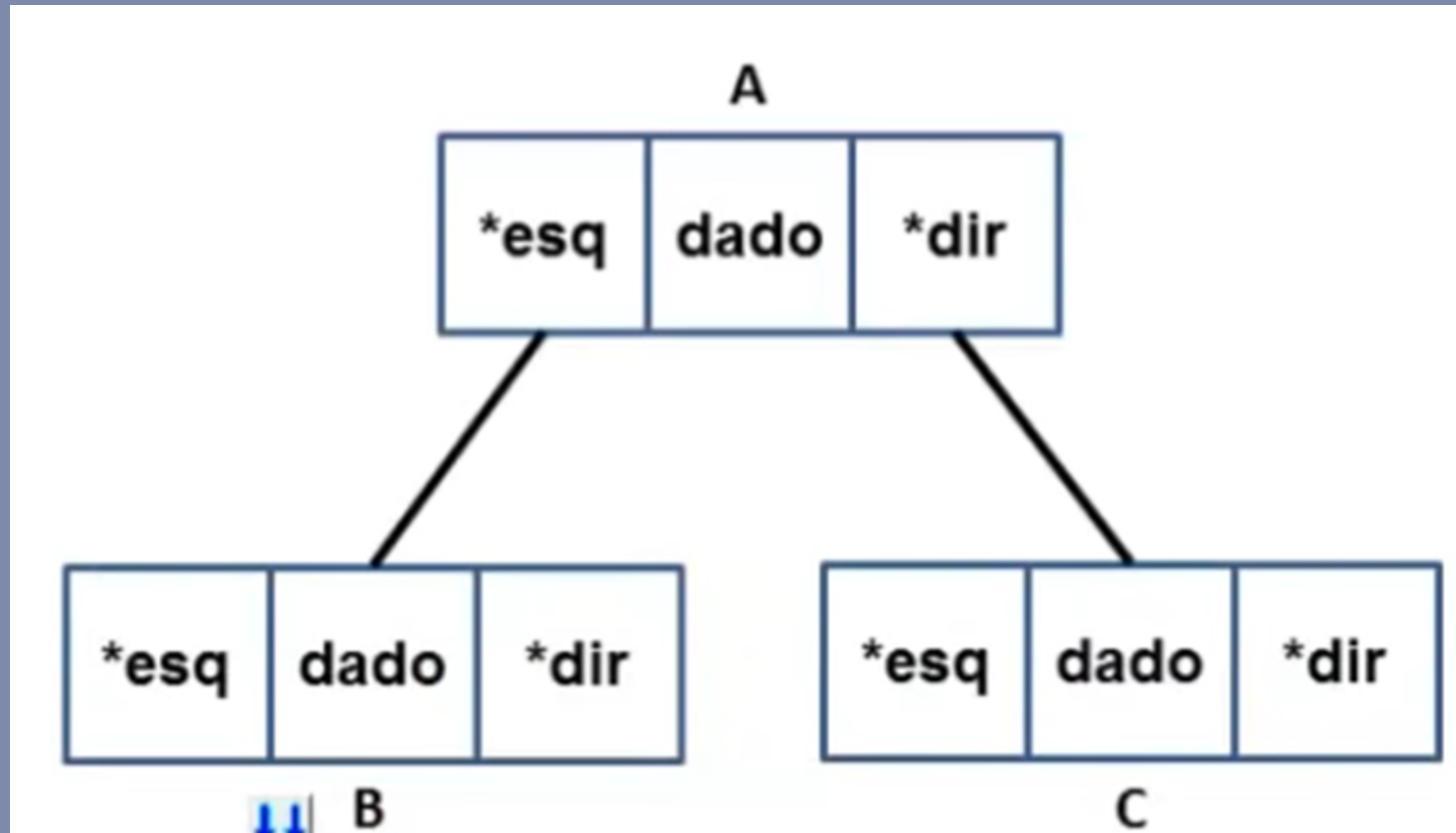
**Inserção de
um
elemento**

**Remoção de
um elemento**

**Acesso a um
elemento**

**Destruição
da árvore**

Alocação de memória



Implementação básica

```
struct NO{  
    int info;  
    struct NO *esq;  
    struct NO *dir;  
}
```

```
ArvBin* cria_ArvBin(){  
    ArvBin* raiz = (ArvBin*) malloc(sizeof (ArvBin)) ;  
  
    if(raiz != NULL)  
        *raiz = NULL;  
    return raiz;  
}
```

Adição de um nó

```
int insere_ArvBin(ArvBin* raiz, int valor){
    if (raiz == NULL)
        return 0;    // árvore vazia
    struct NO* novo;
    novo = (struct NO*) malloc(sizeof (struct NO));

    if (novo == NULL)
        return 0;    // nó vazio
    novo->info = valor;
    novo->dir = NULL;
    novo->esq = NULL;

    if (*raiz == NULL)
        *raiz = novo;    // erro no ponteiro
```

```
else {
    struct NO* atual = *raiz;
    struct NO* ant = NULL;
    while (atual != NULL){
        ant = atual;
        if (valor == atual->info){
            free(novo);
            return 0;    // elemento já existe
        }
        if (valor > atual->info)
            atual = atual->dir;
        else    // define por qual caminho ele irá
            atual = atual->esq;
    }
    if (valor > ant->info)
        ant->dir = novo;    // define o caminho
    else
        ant->esq = novo;
}
return 1;
```

Remoção de um nó

```
struct NO* remove_atual(struct NO* atual){  
    struct NO *no1, *no2;  
    if (atual->esq == NULL){  
        no2 = atual->dir;  
        free (atual) ;  
        return no2;  
    }  
    no1 = atual;  
    no2 = atual->esq; // nó 1 será o elemento que  
// queremos remover, já o nó 2 será o filho do 1
```

```
while (no2->dir != NULL){  
    no1 = no2; // percorre a árvore  
    no2 = no2->dir;  
}  
// no2 é o nó anterior a subraiz que  
// queremos remover na ordem esq - raiz - dir  
// no1 é o pai de no2  
  
if (no1 != atual){  
    no1->dir = no2->esq;  
    no2->esq = atual->esq;  
}  
no2->dir = atual->dir;  
free (atual);  
return no2;  
}
```

Implementação da destruição da árvore

```
void libera_NO(struct NO* no){  
    if (no == NULL)  
        return;  
    libera_NO (no->esq);  
    libera_NO(no->dir) ;  
    free(no) ;  
    no = NULL;  
}
```

```
void libera_ArvBin (ArvBin* raiz){  
    if (raiz == NULL)  
        return;  
    libera_NO(*raiz);           //libera cada nó  
    free(raiz);                 //libera a raiz
```

Características da árvore

```
int estaVazia_ArvBin(ArvBin *raiz){  
    if(raiz == NULL)  
        return 1;  
    if(*raiz == NULL)  
        return 1;  
    return 0;  
}
```

```
int totalNO_ArvBin(ArvBin *raiz){  
    if(raiz == NULL)  
        return 0;  
    if(*raiz == NULL)  
        return 0;  
    int alt_esq = totalNO_ArvBin(&((*raiz)->esq ));  
    int alt_dir = totalNO_ArvBin(&((*raiz)->dir ));  
    return (alt_esq + alt_dir + 1);  
}
```

```
int altura_ArvBin(ArvBin *raiz){  
    if (raiz == NULL)  
        return 0;  
    if (*raiz == NULL)  
        return 0;  
    int alt_esq = altura_ArvBin((( *raiz)->esq));  
    int alt_dir = altura_ArvBin(&(( *raiz)->dir));  
    if (alt_esq > alt_dir)  
        return (alt_esq + 1);  
    else  
        return(alt_dir + 1);  
}
```



Árvores B

Conceito

- São árvores de pesquisa balanceadas;
- Muitos SGDB usam árvores B ou variações de árvores B para armazenar informações;

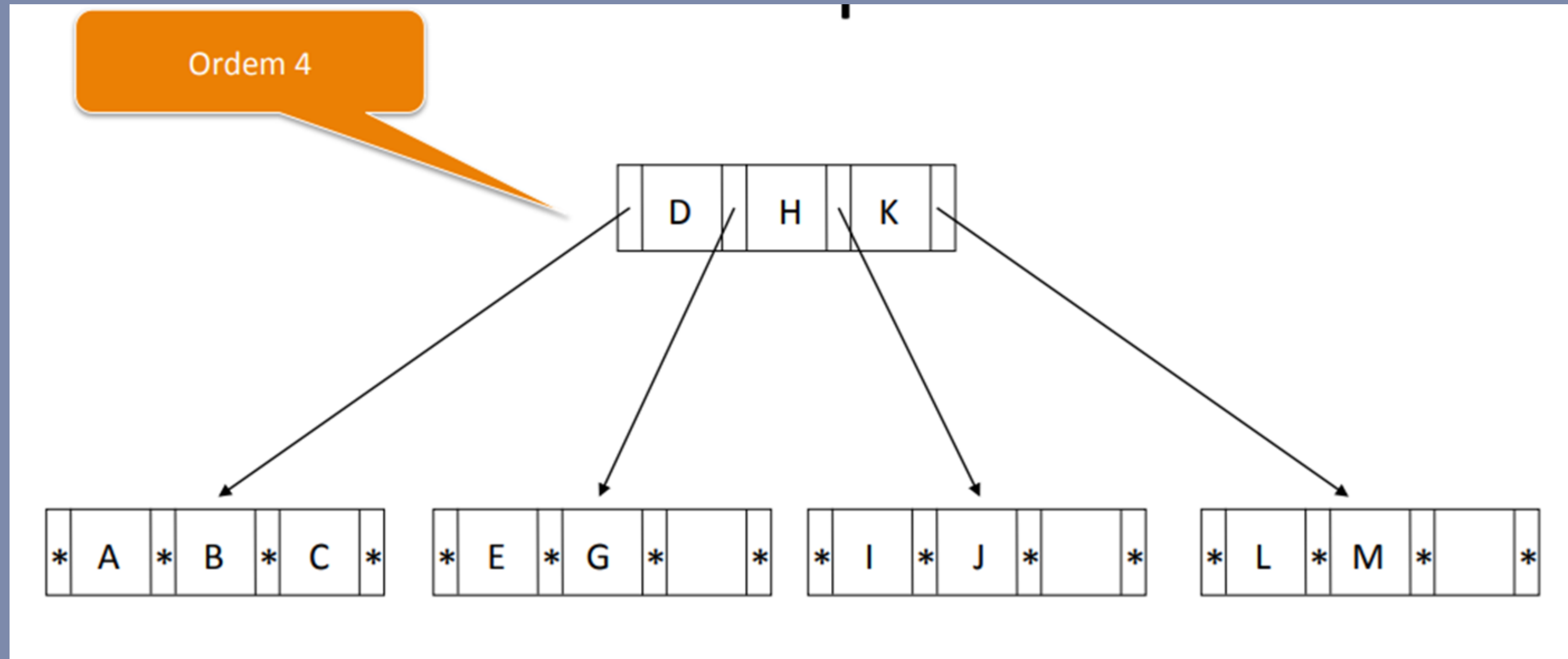
Vantagens de uso

- Voltada para arquivos volumosos;
- Proporciona rápido acesso aos dados.

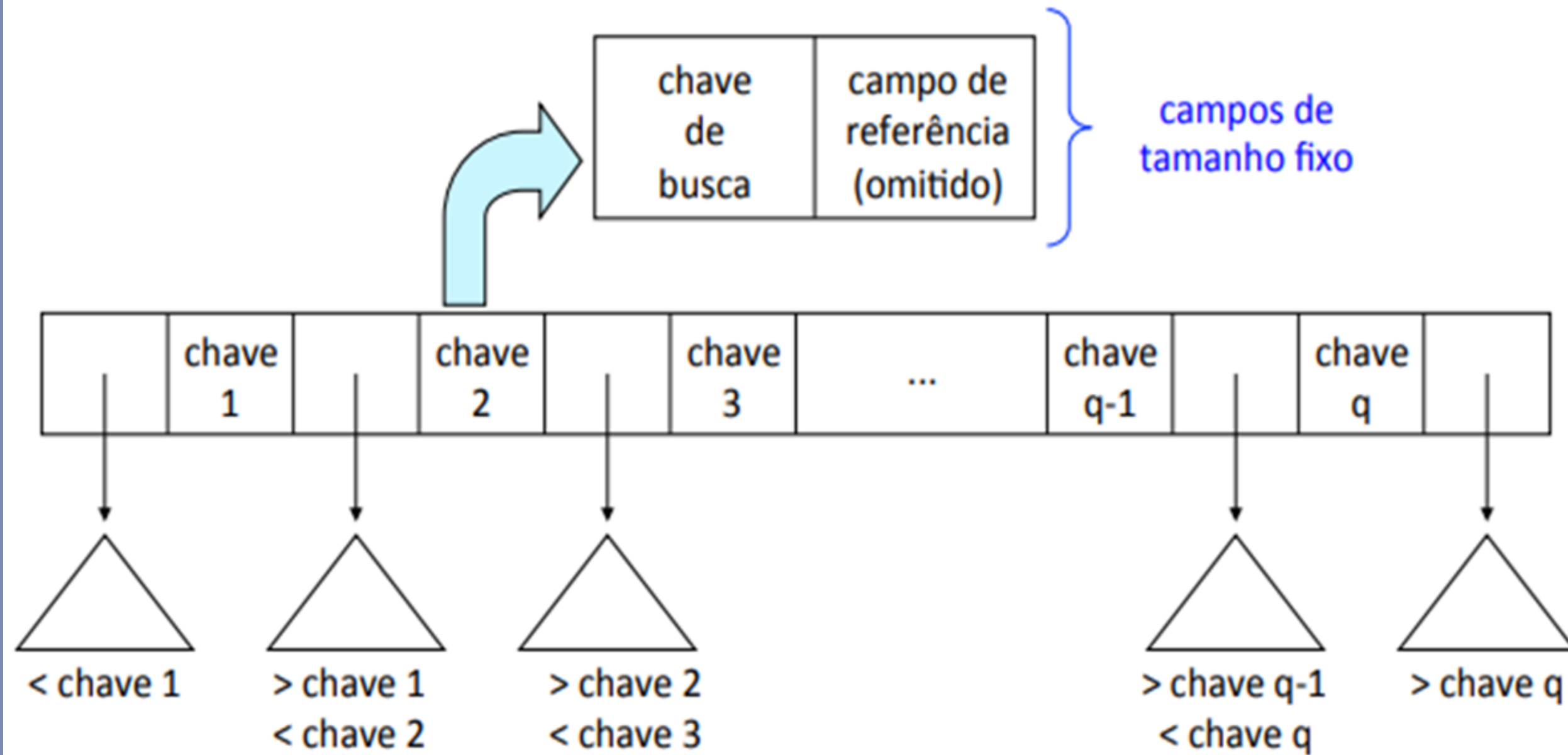
Característica

- Balanceada.

Exemplo de árvore B



Estrutura Lógica de um Nó



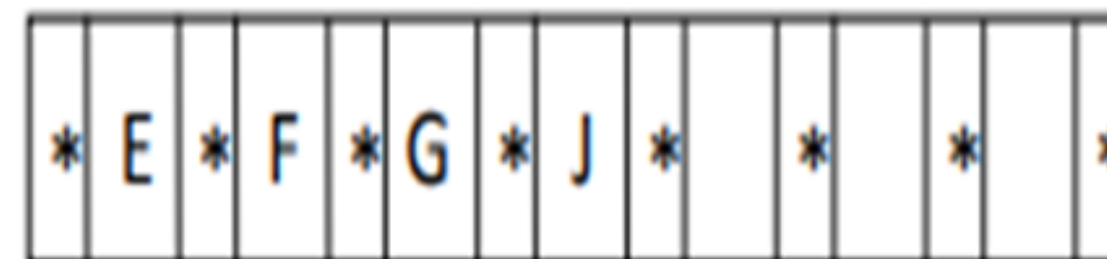
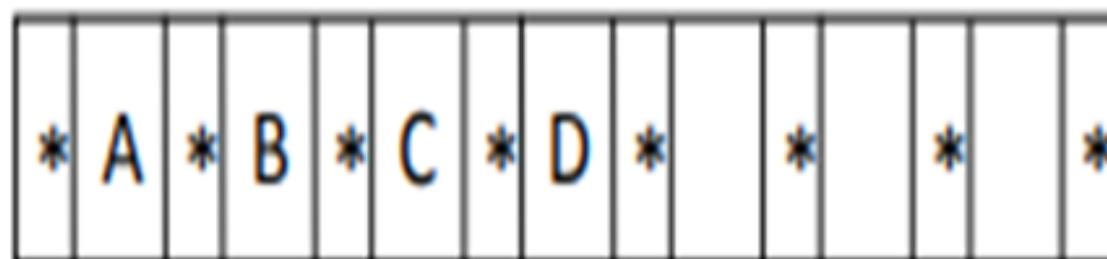
Árvore B - Inserção

Nó com capacidade para 7 chaves - Ordem 8



Inserção de J

- **Passo 1:** particionamento do nó (Split);
 - Nó original -> nó original + novo nó;
 - Split 1-to-2;
 - As chaves são distribuídas uniformemente nos dois nós;
 - chave do nó original + novo chave



Referências

- 01** ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de banco de dados**. 6. ed. São Paulo: Pearson Addison Wesley, 2010;
- 02** NUNES, Fátima L. S. **Indexação e Hashing**. Disponível em: <https://bityli.com/dzGvO>. Acesso em: 1 abr. 2022;
- 03** MONTEIRO, Danielle. **8 dicas para criar índices mais eficientes**, 2018. Disponível em: <https://bityli.com/qYRZI>. Acesso em: 25 mar. 2022;
- 04** Microsoft. **Criar Índices Filtrados**, 2022. Disponível em: <https://bityli.com/kGaYP>. Acesso em: 25 mar. 2022;
- 05** CIFERRI, Cristina. **Tipos de Índices**. Disponível em: <https://bityli.com/VwmMj>. Acesso em: 18 mar. 2022.