1.1 Estruturas de Índices

Índices são estruturas auxiliares que têm por objetivo aumentar a eficiência na recuperação dos registros.

Os índices podem ser agrupados em dois tipos básicos:

- índices ordenados: índices que se baseiam na ordenação de valores;
- índices hash: que se baseiam na distribuição uniforme dos valores por meio de uma faixa de *buckets.*

Os índices ordenados podem ser classificados em:

- índice primário
- índice secundário
- árvores B (B-Tree)

1.1.1 Índice primário

Representam um dos mais antigos esquemas de índices utilizados em sistemas de banco de dados. São projetados para aplicações que requerem tanto o processamento seqüencial de um arquivo inteiro quanto o acesso aleatório a registros individuais. É dito primário por estar baseado na chave de ordenação do arquivo.

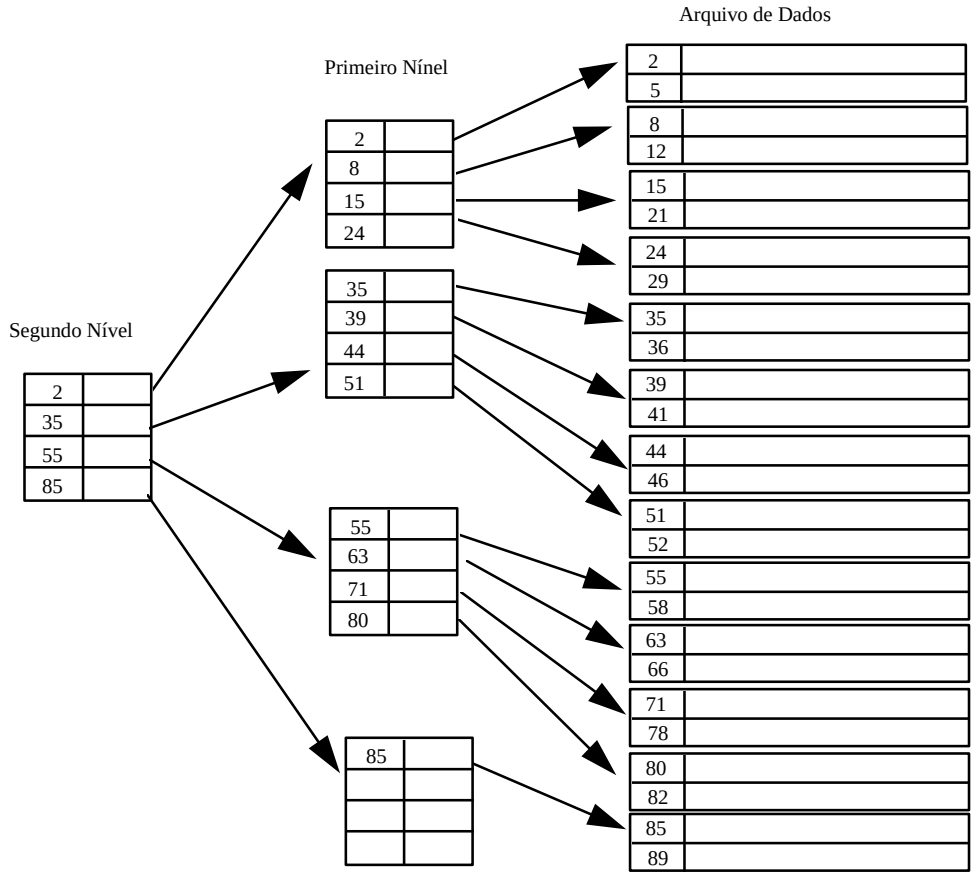
Um índice ordenado pode ser:

- *denso*: um registro de índice aparece para cada valor de chave de procura no arquivo;
- *esparso*: um registro de índice é criado apenas para alguns dos valores.

1.1.2 Índices de Níveis Múltiplos

Mesmo utilizando um índice esparso, o próprio índice pode se tornar muito grande para ser processado eficientemente. O objetivo da utilização de um índice multi-nível é reduzir a parte do índice em que a consulta será realizada.

Exemplo:



1.1.3 Índices secundários

Um índice secundário em uma chave candidata parece-se com um índice denso, exceto pelo fato de que os registros apontados por valores sucessivos no índice não estão armazenados seqüencialmente.

Os índices secundários melhoram o desempenho das consultas que usam chaves diferentes da chave de procura do índice primário. Entretanto, eles impõem uma sobrecarga significativa na atualização do banco de dados.

O projetista do banco de dados decide quais índices secundários são desejáveis baseado na estimativa da freqüência de consultas e atualizações.

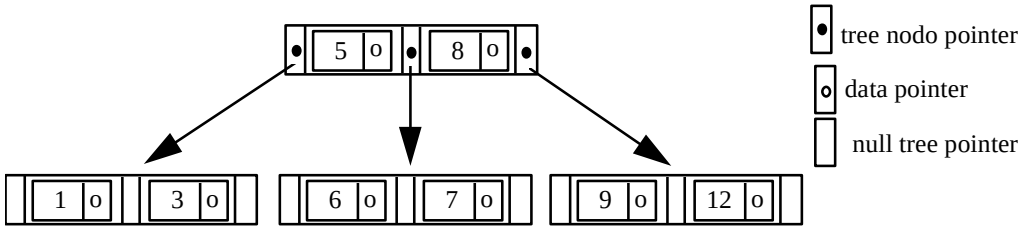
1.2 Árvores B (B-Tree)

As estruturas de índices apresentadas anteriormente não apresentam bom desempenho. Uma das estruturas mais utilizadas são as B-Trees, pois as mesmas possuem um bom desempenho mesmo no caso de muitas inserções e remoções. Uma B-Tree é uma estrutura de índice multi-nível, na forma de uma árvore balaceada. Numa árovore balanceada., todos os caminhos da raiz até a folha têm o mesmo comprimento.

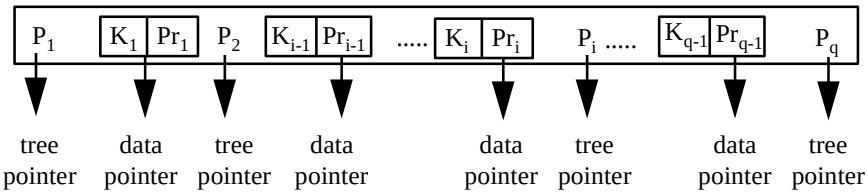
Algumas premissas:

- todos os nodos têm o mesmo número de níveis;
- uma B-tree de ordem *p* tem *p* ponteiros e *p-1* chaves em cada nodo;
- Cada nodo tem no máximo p pointeiros para sub-árvores
- Cada nodo exceto a raiz e as folhas, tem pelo menos $\lceil p/2 \rceil$ ponteiros para sub-árvores. A raiz deve possuir pelo menos dois ponterios para sub-árvore, a menos que seja o único nodo na árvore;
- Todo o nodo exceto a raiz possui ocupação mínima igual a $\lceil p/2 \rceil$ - 1 chaves;
- Um nodo com q tree-pointers, q <=p, tem q-1 valores de chave de pesquisa;

B-tree de ordem 3 com os valores inseridos na ordem: 8,5,1,7,3,12,9,6



Estrutura do nodo de uma B-Tree



1.2.1 Algoritmo de inserção em B-tree:

1. A partir da raiz, procura o nodo onde deve ser inserida a chave
2. Verifica se existe e espaço livre, no nodo, para inserção
2.1 SIM - INSERE ORDENADAMENTE
2.2 NÃO
2.2.1 Forma um conjunto composto pelas chaves contidas no nodo, mais a chave que está sendo inserida
2.2.2 Ordena o conjunto
2.2.3 Escolhe o elemento central do conjunto
2.2.4 Cria um novo nodo e limpa o nodo atual
2.2.5 Insere os elementos do conjunto que são menores que o elemento central no nodo atual

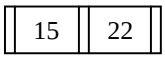
2.2.6 Insere os elementos do conjunto que são maiores que o elemento central no novo nodo

2.2.7 Insere o elemento central no nodo ascendente, seguindo o mesmo algoritmo. Se não houver ascendente, criar o nodo e inserir a chave (caso específico da raiz)

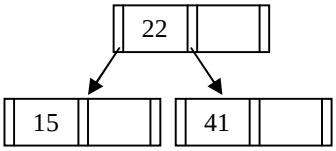
2.2.8 Faz o encadeamento, colocando o nodo atual à esquerda do elemento central do conjunto e o nodo novo à direita

Exemplo: 22,15, 41,33,59,63,95 (2 valores de chave e 3 ponteiros)

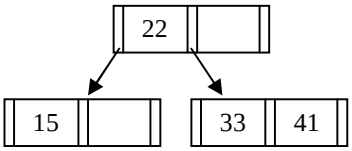
Resultado da inserção das chaves: 22, 15



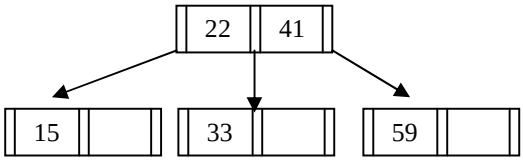
Inserção da chave:41
Conjunto: < 15,22, 41>
Elemento Central: 22
Resultado da inserção da chave: 41



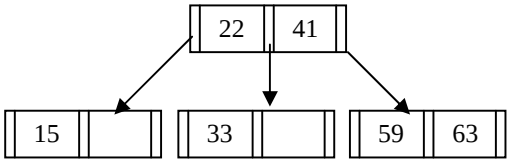
Inserção do 33:



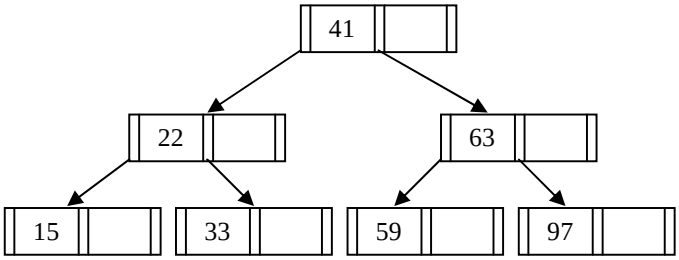
Inserção do 59:
Conjunto: < 33, 41, 59>
Elemento Central: 41
Resultado da inserção da chave: 59



Inserção do 63:



Inserção do 97:
Conjunto: < 59, 63,97>
Elemento Central: 63
Resultado da inserção da chave: 97



1.3 Organizações Hash (Arquivos Diretos)

1.3.1 Hashing Estático

- A técnica de hashing evita a utilização de índice já que existe uma função que determina o endereço onde o registro está localizado;
- Considere um arquivo na memória principal organizado em um array de registro, o índice começa em 0 e vai até M-1. Existem M entradas (endereços) correspondente ao array de registros.
- Uma das funções hash mais conhecidas é aquela que transforma o valor de um campo em um endereço entre 0 e M-1:

h(K) = K mod M. O valor de h é utilizado para indexar o registro no array

- O problema de colisões é resolvido através do encadeamento dos registros;

	Codf	Nome	Idade	Salário	p. overflow
0	100				M
1	101				M + 1
				.....	
M-1					-1
M	200				-1
M+1	201				-1

Para organizar os arquivos no disco através de uma técnica hash é necessário fazer algumas modificações já que os registros são armazenados em blocos (determinar o endereço para o bloco);
- O problema de colisões é menor neste caso (link em cada bloco);
- Ao invés de especificar um endereço para cada registro é determinado um

- Exemplo:
h(K) = K mod B, onde B = 10;

bucket 0

340	
460	
	record pointer

bucket 1

321	
761	
	record pointer

.....

bucket 9

399	
89	
	record pointer

Overfolw

981	
	record pointer

- Se o número de buckets for pequeno o número de colisões será maior;
- escolher uma função hash baseada no tamanho antecipado do arquivo em algum ponto no futuro. Embora a degradação do desempenho seja evitada, uma quantidade significativa de espaço é desperdiçada inicialmente (número de buckets muito grande).
- reorganizar periodicamente a estrutura hash em resposta ao crescimento do arquivo (escolha de uma nova função hash);
- Diversas técnicas de hashing permitem a função hash ser modificada dinamicamente a fim de acomodar o crescimento ou encolhimento do banco de dados. Estas técnicas são chamadas de função hash dinâmicas.

1.3.2 Hashing Dinâmico

- Um dos tipos de hashing dinâmico é chamado de hashing extensível. O hashing extensível permite mudanças no banco de dados dividindo e agrupando buckets à medida que o banco de dados cresce e diminui;
- Exemplo: bucket com dois registros;

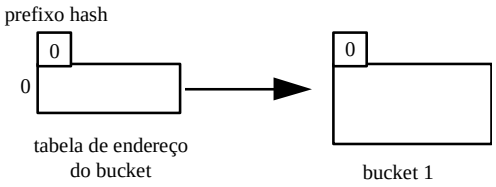
- Assumimos inicialmente que o arquivo está vazio (c) e vamos inserir um registro de cada vez. O valor da função hash aplicada sobre o atributo nome e que resulta em 32 bits é mostrado na letra (b).
Seqüência de inserções:

1. Patrícia - a tabela de endereço contém um pointer para um bucket e um registro é inserido.
2. Vitória - o bucket ainda possui um espaço disponível e o registro é inserido.
3. Cristina - o bucket está cheio. Uma vez que o prefixo hash da tabela de endereços é igual ao prefixo hash do bucket precisamos aumentar o número de bits usados no hash. Usamos agora 2 bits permitindo 2^1=2 buckets. Isso requer duplicar o tamanho da tabela de endereços de bucket para duas entradas. Dividimos o bucket, colocando esses registros cujas chaves de pesquisas tenham o hash iniciado por 1 em um novo bucket, e deixando os outros registros no bucket original (d).
4. Antônio - o primeiro bit de h(Antônio) = 1, assim precisamos inserir este registro no bucket apontado pela entrada 1 na tabela de endereço do bucket. O bucket está cheio e o número de bits da tabela de endereços é igual ao prefixo hash do bucket. Aumentamos o número de bits que usamos no hash para 2. Assim deve-se aumentar a tabela de endereços do bucket para quatro entradas. Uma vez que o bucket com prefixo 0 não foi dividido as duas entradas da tabela de endereço 00 e 01 apontam para o mesmo bucket. Para cada registro do bucket para o prefixo 1, examinamos os dois primeiros bits do hash para determinar qual bucket da nova estrutura deveria mantê-lo. Continuamos desta maneira até inserirmos todos os registros da tabela empregado.

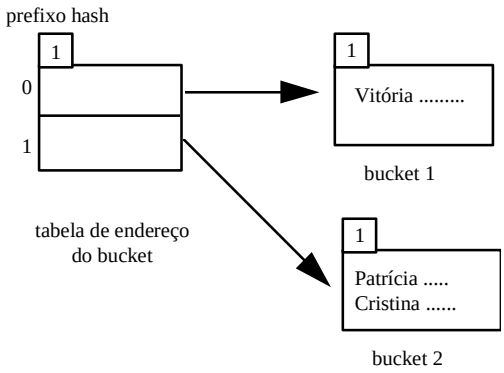
b) Função hash para nome do empregado

Nome	h(nome)
Ricardo	0010 1101 .....
Alberto	1101 0101 .....
Cristina	1010 0011 .....
Ana Paula	1000 0111 .....
Patrícia	1111 0001 .....
Antônio	1011 0101 .....
Vitória	0101 1000 .....

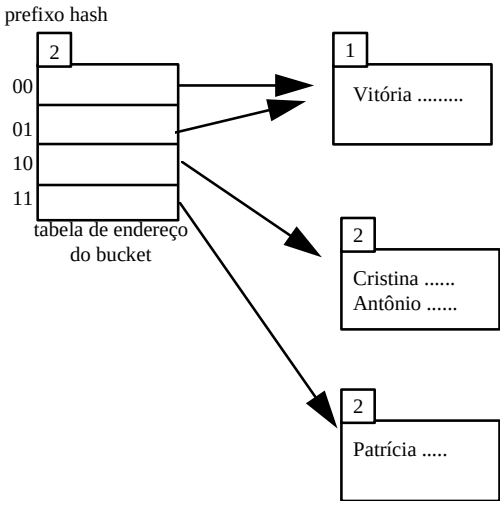
c) Estrutura Hash Extensível Inicial



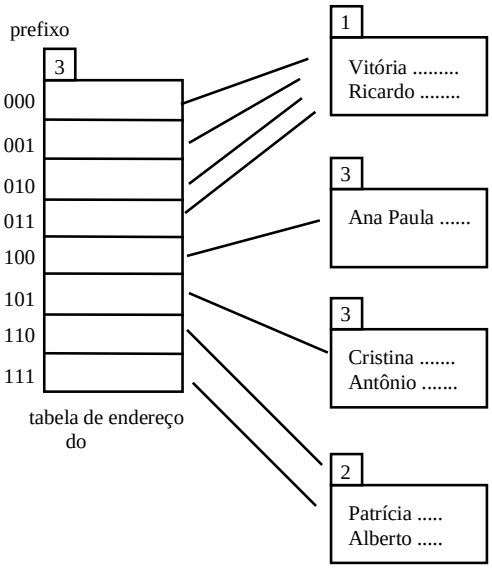
d) Estrutura Hash após três inserções



e) Estrutura após quatro inserções



f) Estrutura hash extensível para o arquivo empregado



Vantagens na utilização do hash extensível

1. O desempenho não degrada a medida que o arquivo cresce. Além do mais, existe um mínimo de espaço improdutivo;

2. Não é preciso reservar buckets para crescimentos futuros, os buckets por ser alocados dinamicamente.

3. Mesmo existindo a necessidade de reorganização, esta é feita em pequemas porções e não degrada a performance;

Desvantagens na utilização do hash extensível

Uma desvantagem para utilização desta técnica é que as pesquisas envolvem um nível adicional de descaminho, uma vez que precisamos acessar a tabela de endereços de bucket antes de acessar o próprio bucket. Mas isto tem um impacto pequeno no desempenho.

1.4 Exemplo de Cálculos de Número de Acessos

Cálculo de número de acessos:

* Arquivo seqüencial sem índice:
  Número de registros do arquivo: r = 30000
  Tamanho do bloco: B = 1024 bytes
  Tamanho dos registros do arquivo: R = 100 bytes
  Fator de Bloco: bfr = B/r = 1024/100 = 10 registros por bloco.
  Total de blocos para o arquivo: b = r/bfr = 30000/10 = 3000 blocos.
  Pesquisa binária = $Log_2 b$ = 12 acessos a bloco
* Arquivo seqüencial com índice:
  Tamanho do campo chave de ordenação do arq.: V= 9 bytes
  Tamanho do endereço do bloco: P = 6 bytes
  Tamanho de cada entrada no índice: ri = (9 + 6) = 15 bytes
  Fator de Bloco do índice: bfri = b/ri = 1024/15 = 68 entradas por bloco
  Total de blocos necessários para o índice: bi = 3000/68 = 45 blocos
  Pesquisa binária sobre o índice: $Log_2 b = log_2 45$ = 6 acessos a bloco
  Pesquisa do registro utilizando o índice: 6 + 1 = 7 acessos a bloco

Exemplo3: O índice secundário denso do exemplo 2 é convertido em um índice multi-nível.

fator de bloco: bfri = 68 entradas de índice por bloco
num. de blocos do primeiro nível: b1 = 442 blocos
num. de blocos no segundo nível: b2 = 442/68 = 7 blocos
num. de blocos no terceiro nível: b3 = 7/68 = 1 bloco
Para pesquisar um determinado registro são necessários 3 + 1 = 4 acessos