Indexação e hashing

Capítulo 12: Indexação e hashing

- Hashing estático
- Hashing dinâmico

Hashing estático

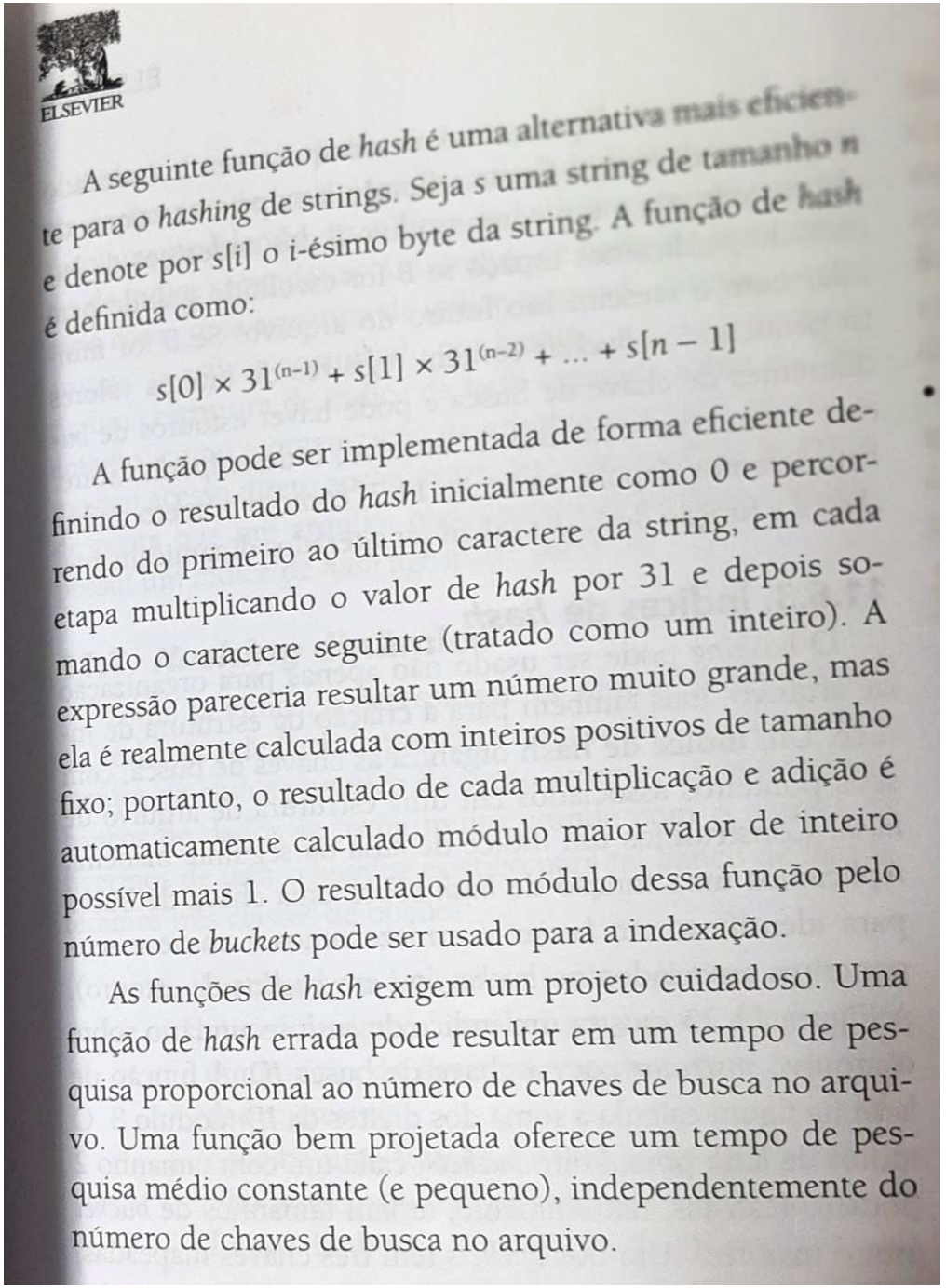
- Um bucket é uma unidade de armazenamento contendo um ou mais registros (um bucket normalmente é um bloco de disco).
- Em uma organização de arquivo de hash, obtemos o bucket de um registro diretamente por seu valor de chave de busca, usando uma função de hash.
- A função de hash *h* é uma função do conjunto de todos os valores de chave de busca *K* para o conjunto de todos os endereços de bucket *B*.
- A função de hash é usada para localizar registros para acesso, inserção e também exclusão.
- Os registros com diferentes valores de chave de busca podem ser mapeados para o mesmo bucket; assim, o bucket inteiro precisa ser pesquisado sequencialmente, para localizar um registro.

Exemplo de organização de arquivo de hash

- Organização de arquivo de hash do arquivo *conta*, usando *nome-agência* como chave (ver figura no slide seguinte).
- Existem 10 buckets
- A representação binária do *i*-ésimo caractere é considerada como o inteiro *i*
- A função de hash retorna a soma das representações binárias dos caracteres módulo 10
  - Por exemplo: h(Perryridge) = 5 h(Round Hill) = 3 h(Brighton) = 3

Hash de strings

- Silbershatz, Korth, Sudarshan. Sistema de Banco de Dados, tradução 6ª edição
  - Página 321, Capitulo 11 Indexação e hashing



ELSEVIER

A seguinte função de *hash* é uma alternativa mais eficiente para o *hashing* de strings. Seja s uma string de tamanho *n* e denote por s[i] o *i*-ésimo byte da string. A função de *hash* é definida como:

$$s[0] \times 31^{(n-1)} + s[1] \times 31^{(n-2)} + \ldots + s[n-1]$$

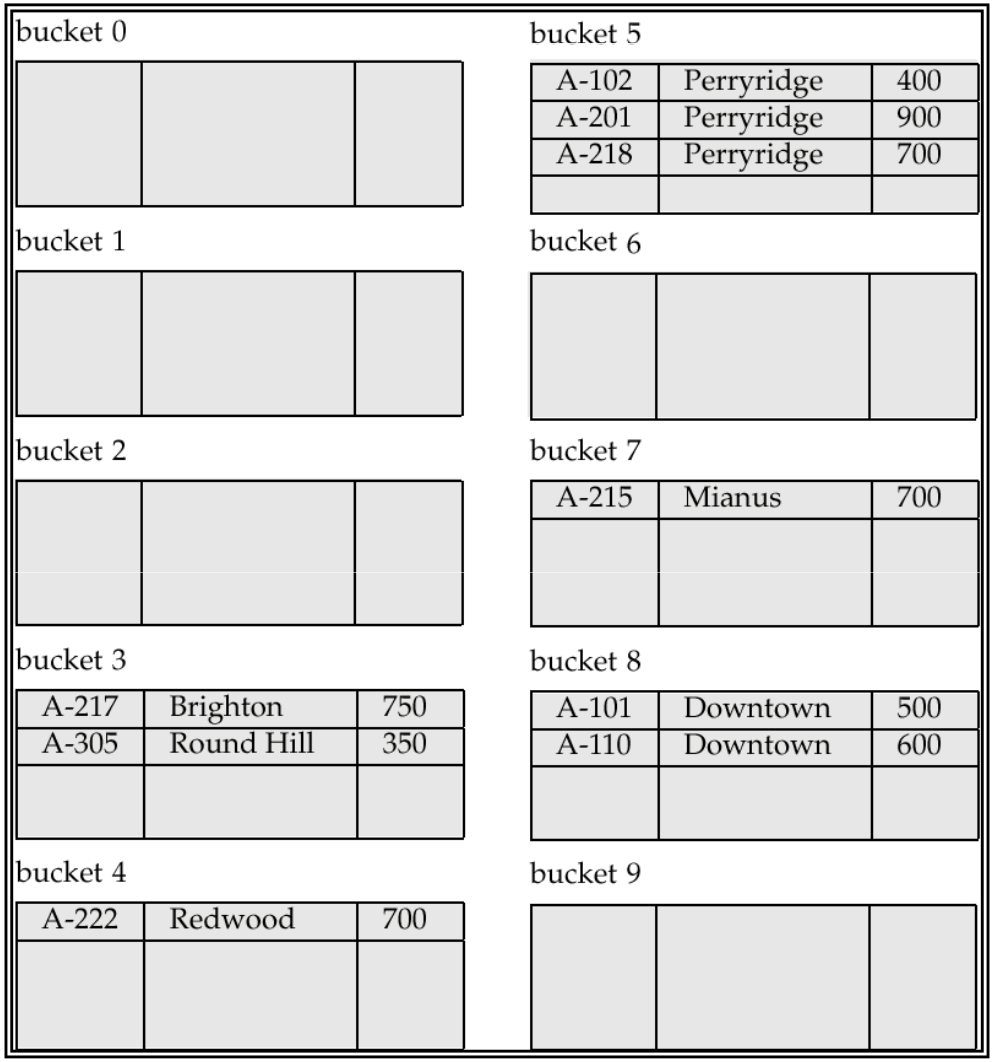
A função pode ser implementada de forma eficiente definindo o resultado do *hash* inicialmente como 0 e percorrendo do primeiro ao último caractere da string, em cada etapa multiplicando o valor de *hash* por 31 e depois somando o caractere seguinte (tratado como um inteiro). A expressão pareceria resultar um número muito grande, mas ela é realmente calculada com inteiros positivos de tamanho fixo; portanto, o resultado de cada multiplicação e adição é automaticamente calculado módulo maior valor de inteiro possível mais 1. O resultado do módulo dessa função pelo número de *buckets* pode ser usado para a indexação.

As funções de *hash* exigem um projeto cuidadoso. Uma função de *hash* errada pode resultar em um tempo de pesquisa proporcional ao número de chaves de busca no arquivo. Uma função bem projetada oferece um tempo de pesquisa médio constante (e pequeno), independentemente do número de chaves de busca no arquivo.

Exemplo de organização de arquivo de hash (cont.)

- Organização de arquivo de hash do arquivo *conta*, usando *nome-agência* como chave (detalhes no slide anterior).

- Por exemplo:
  - h(Perryridge) = 5
  - h(Round Hill) = 3
  - h(Brighton) = 3



Funções de hash

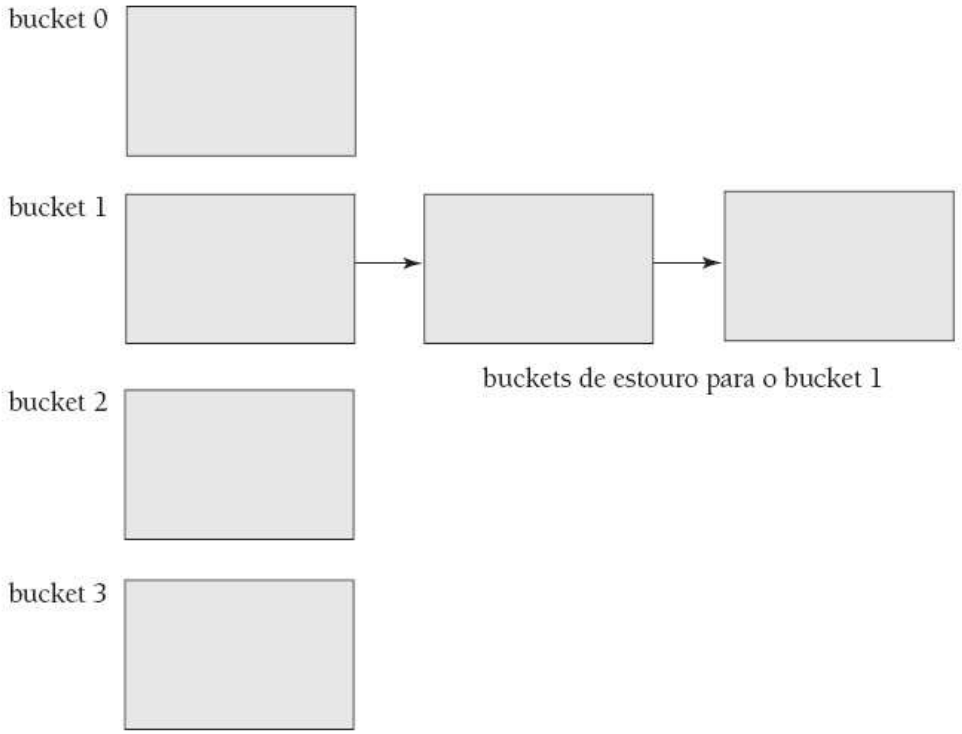
- A pior função de hash mapeia todos os valores de chave de busca para o mesmo bucket; isso torna o tempo de acesso proporcional ao número de valores de chave de busca no arquivo.
- Uma função de hash ideal é uniforme, ou seja, cada bucket recebe o mesmo número de valores de chave de busca do conjunto de *todos* os valores possíveis.
- A função de hash ideal é aleatória, de modo que cada bucket terá o mesmo número de registros atribuídos a ele, independente da *distribuição real* dos valores de chave de busca no arquivo.
- As funções de hash típicas realizam seu cálculo sobre a representação binária interna da chave de busca.
  - Por exemplo, para uma chave de busca de string, as representações binárias de todos os caracteres na string poderiam ser somadas e a soma módulo número de buckets poderia ser retornada.

Tratamento de estouros de bucket

- O estouro de bucket pode ocorrer devido a
  - Buckets insuficientes
  - Distorção na distribuição de registros. Isso pode ocorrer por dois motivos:
    - vários registros possuem o mesmo valor de chave de busca
    - a função de hash escolhida produz a distribuição não uniforme de valores de chave
- Embora a probabilidade de estouro de bucket possa ser reduzida, ela não pode ser eliminada; ele é tratado pelo uso de *buckets de estouro.*

Tratamento de estouros de bucket (cont.)

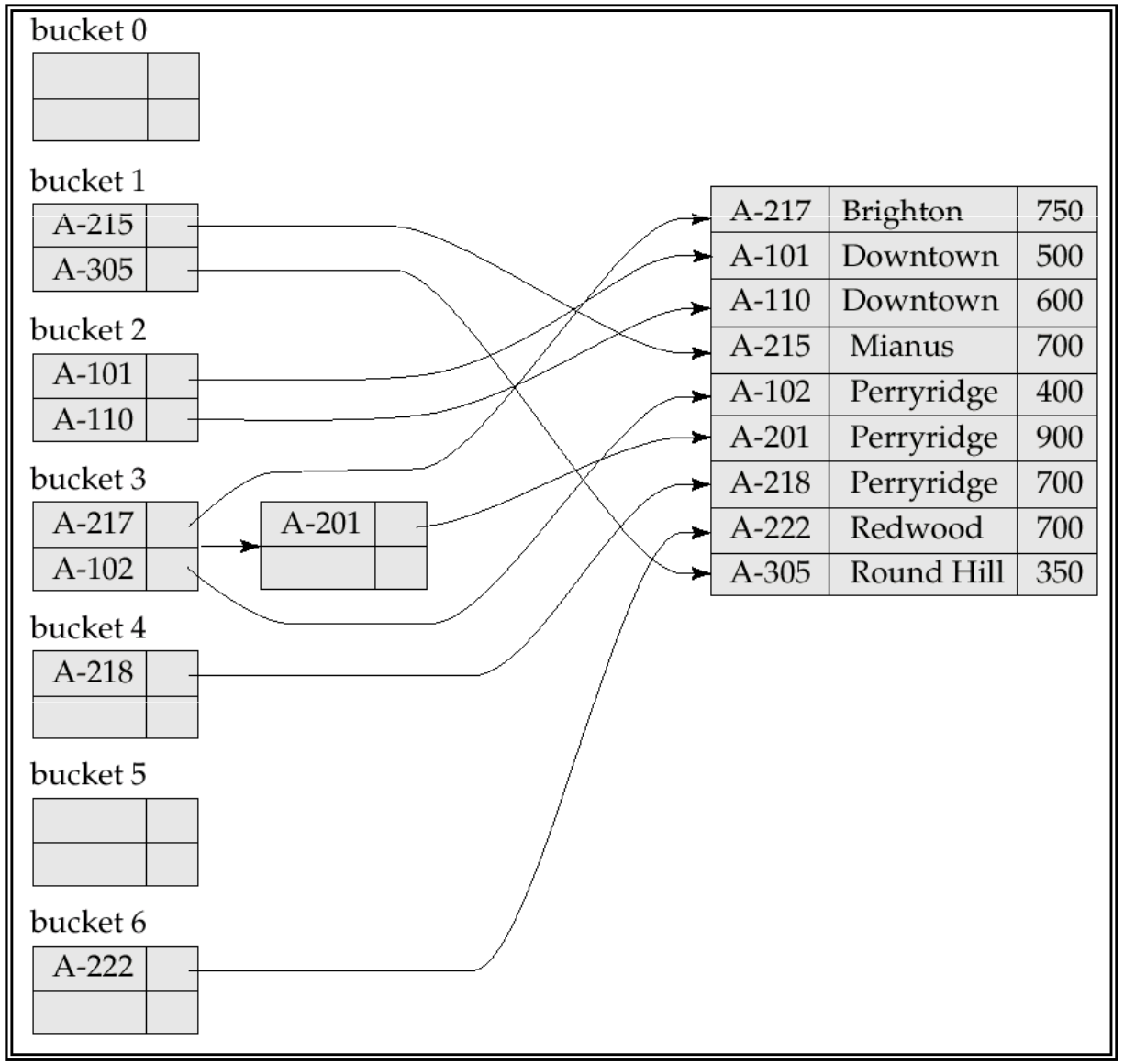
- Encadeamento de estouro – os buckets de estouro de determinado bucket são encadeados em uma lista interligada.
- O esquema acima é chamado de hashing fechado.
  - Uma alternativa, chamada hashing aberto, que não usa buckets de estouro, não é apropriada para aplicações de banco de dados.



Índices de hash

- O hashing pode ser usado não apenas para organização de arquivos, mas também para a criação de estrutura de índice.
- Um índice de hash organiza as chaves de busca, com seus ponteiros de registro associados, em uma estrutura de arquivo de hash.
- Estritamente falando, os índices de hash sempre são índices secundários
  - Se o próprio arquivo for organizado usando o hashing, um índice de hash primário separado sobre ele, usando a mesma chave de busca, é desnecessário.
  - Porém, usamos o termo índice de hash para se referir às estruturas de índice secundárias e arquivos organizados em hash.

Exemplo de índice de hash



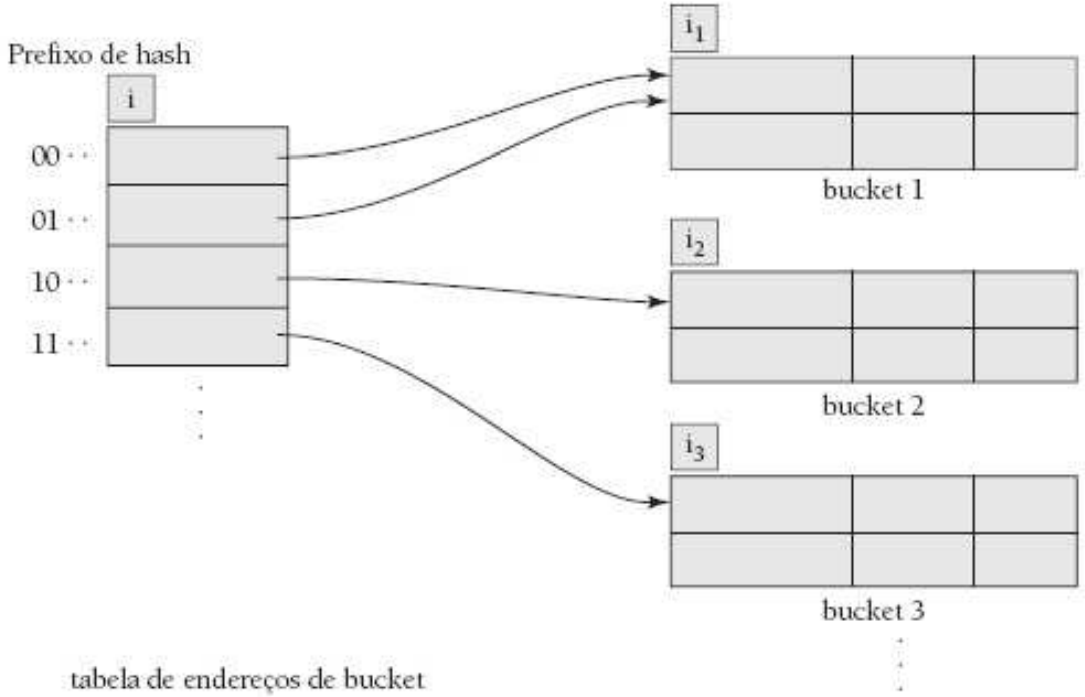
Deficiências do hashing estático

- No hashing estático, a função *h* mapeia os valores de chave de busca a um conjunto fixo *B* de endereços de bucket.
  - Os bancos de dados crescem com o tempo. Se o número inicial de buckets for muito pequeno, o desempenho degradará devido a muitos estouros.
  - Se o tamanho do arquivo em algum ponto no futuro for antecipado e o número de buckets for alocado de acordo, uma quantidade de espaço significativa será desperdiçada inicialmente.
  - Se o banco de dados encurtar, novamente o espaço será desperdiçado.
  - Uma opção é a reorganização periódica do arquivo com uma nova função de hash, mas isso é muito dispendioso.
- Esses problemas podem ser evitados por meio de técnicas que permitem que o número de buckets seja modificado dinamicamente.

Hashing dinâmico

- Bom para banco de dados que aumenta e diminui de tamanho
- Permite que a função de hash seja modificada dinamicamente
- Hashing extensível - uma forma de hashing dinâmico
  - Função de hash gera valores por um intervalo grande - normalmente, inteiros de *b* bits, com *b* = 32
    - não queremos criar um bucket para cada valor de hash (2^32 = 4 bilhões de buckets)
  - A qualquer momento, usa apenas um prefixo da função de hash para indexar em uma tabela de endereços de bucket.
  - Considere que o tamanho do prefixo seja *i* bits, $0 \leq i \leq 32$.
  - Tamanho da tabela de endereços de bucket = 2^i. Inicialmente, $i = 0$
  - O valor de *i* aumenta e diminui à medida que o tamanho do banco de dados aumenta e diminui.
  - Várias entradas na tabela de endereços de bucket podem apontar para um bucket.
  - Assim, o número real de buckets é $< 2^i$
    - O número de buckets também muda dinamicamente, devido à união e divisão de buckets.

Estrutura geral do hash extensível



Nessa estrutura, $i_2 = i_2 = i$, enquanto $i_1 = i - 1$ (veja os detalhes no próximo slide)

Hashing extensível *versus* outros esquemas

- Benefícios do hashing extensível:
  - Desempenho do hash não diminui com o crescimento do arquivo
  - Sobrecarga de espaço mínima
- Desvantagens do hashing extensível:
  - Nível extra de indireção para encontrar registro desejado
  - A própria tabela de endereços de bucket pode se tornar muito grande (maior que a memória)
  - Mudar o tamanho da tabela de endereços de bucket é uma operação dispendiosa

Hashing extensível *versus* outros esquemas

- Comparação de hashing fechado com hashing aberto
  - essa forma da estrutura descrita é também chamada de hashing fechado
  - hashing aberto: conjunto de buckets é fixo, e não existem cadeias de estouro
  - se bucket cheio, insere registro em outro bucket que possua espaço (em ordem cíclica)
- Hashing aberto tem sido usado para construir tabelas de símbolos em compiladores
  - em banco de dados, hashing fechado é preferível !
  - motivo: exclusão sob hashing aberto é trabalhosa (compiladores realizam apenas inserções e pesquisa nas tabelas), porém, em um sistema de banco de dados, é importante poder lidar com exclusões

Comparação entre indexação ordenada e hashing

- Custo da reorganização periódica
- Frequência relativa de inserções e exclusões
- É desejável otimizar o tempo de acesso médio em detrimento do tempo de acesso no pior caso?
- Tipo esperado das consultas:
  - O hashing geralmente é melhor na recuperação de registros com um valor especificado da chave.
  - Se as consultas por intervalo forem comuns, os índices ordenados são preferíveis

Acknowledgements

- Contém slides disponibilizados pela Editora Campus/Elsevier e autores do livro:
  - Abraham Silberschatz, Henry Korth, S. Sundarshan, Sistema de Banco de Dados, tradução da 5ª edição, Editora Campus/Elsevier