Complementos de Bases de Dados
– Índices –

Engenharia Informática

2º Ano / 1º Semestre

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt

DSI :: Escola Superior de Tecnologia de Setúbal :: Instituto Politécnico de Setúbal

Sumário

- Conceitos
- Índices ordenados
- Ficheiros de indexação em B+-Tree
- Ficheiros de indexação em B-Tree
- *Hashing* estático
- *Hashing* dinâmico
- Comparação entre modos de indexação

- Definição de índices em SQL
  - Exemplos
- Linhas orientadoras de estratégias de indexação

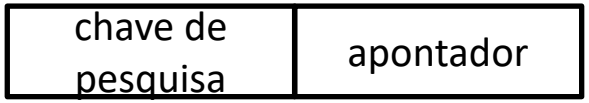
Conceitos

Definições

- ***Índices***:
  Estruturas para aumentar o desempenho no acesso à informação.
  - Exemplo: Catálogo de autores numa livraria.

➢ Fornecem um caminho de acesso aos registos

- **Um ficheiro de indexação** é constituído por registos, no seguinte formato:



- *Chave de Pesquisa*: atributo ou conjunto de atributos usado para "procurar" os registos num ficheiro.

❖ Os ficheiros de indexação são de menor dimensão que os ficheiros de dados originais.
❖ A existência de índices não modifica as relações nem a semântica das consultas

Conceitos

Definições

- **Tipos de índices**:
  - **Índices ordenados**:
    as chaves de pesquisa são armazenadas por uma certa ordem.
  - **Índices Hash**:
    as chaves de pesquisa são distribuídas uniformemente por "*buckets* "
    através de uma função de *Hash*

Índices Ordenados

Índice Primário, Clustered e Secundário

as entradas de indexação são ordenadas pela chave de pesquisa
(e.g. autores no catálogo de uma livraria)

- **Índice Primário:** a ordenação coincide com a ordenação do ficheiro de dados, pelo campo que é também chave primária

- **Índice Clustered**: é o índice sobre a chave de pesquisa que específica a ordem sequencial do ficheiro de dados contudo esse campo não é chave primária

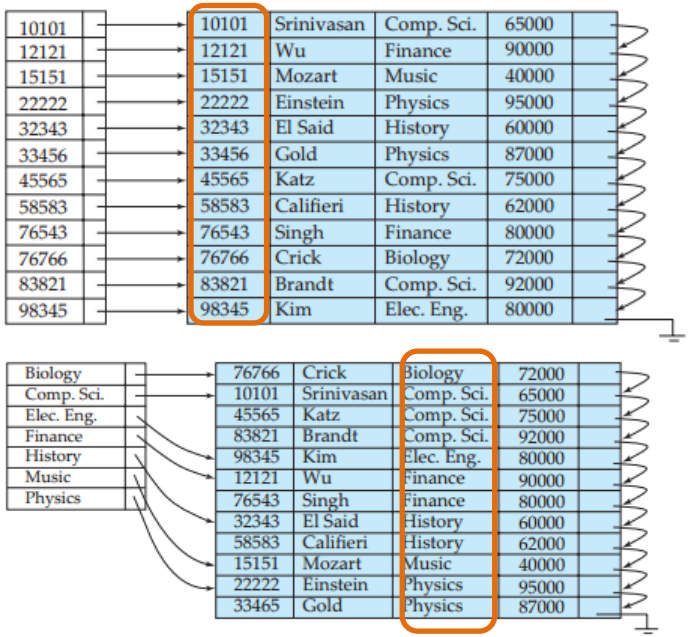
- **Índice secundário** (*nonclustered index*): índice em que a chave de pesquisa especifica uma ordem diferente da ordem sequencial do ficheiro.

Índices Ordenados

Dense Indices

Índices "***densos***":

- a cada valor da chave de pesquisa corresponde um registo de índice
  - ❖ se houver mais do que um registo com o mesmo valor apontado (caso possível se: chave de pesquisa <> chave primária) então o registo de índice apontará para o primeiro registo do valor apontado e os seguintes registos do mesmo valor serão subsequentes.
    - ✓ Está subentendida a ordenação do ficheiro de registos por ordem da chave de pesquisa



- se for um ***dense nonclustering índex***
  então cada entrada de registo de índice terá de guardar todos os ponteiros para todas as ocorrências do valor apontado

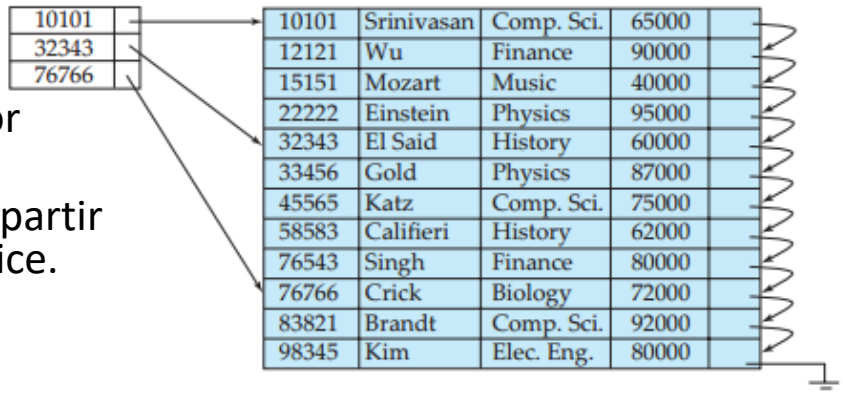
Índices Ordenados

Sparse Indices

Índices "***esparsos***":

Contém registos de índice, apenas para alguns dos valores da chave de pesquisa.

- Aplicável quando os registos estão ordenados sequencialmente pela chave de pesquisa.
- Para localizar um registo com o valor K da chave de pesquisa:
  1. Localizar o registo de índice com o maior valor < K, da chave de pesquisa.
  2. Pesquisar sequencialmente o ficheiro a partir do registo apontado pelo registo de índice.



- ❖ Ocupa menos espaço,
- ❖ Menor "*overhead* " nas operações de *insert* e *delete*.
- ❖ Normalmente mais lento na procura de registos.

Índices Ordenados

Avaliação das diferentes técnicas de indexação

- nenhuma é absolutamente melhor que outra em todos os aspetos
- Há que considerar:
  - Tipos de acessos à informação:
    - registos com um atributo, igual a um valor específico;
    - registos com um atributo, com valor num determinado intervalo de valores.
  - Tempo de acesso.
  - Tempo para *insert*.
  - Tempo para *delete*.
  - Espaço adicional.

✓ Dica: uma entrada num índice esparso por cada bloco

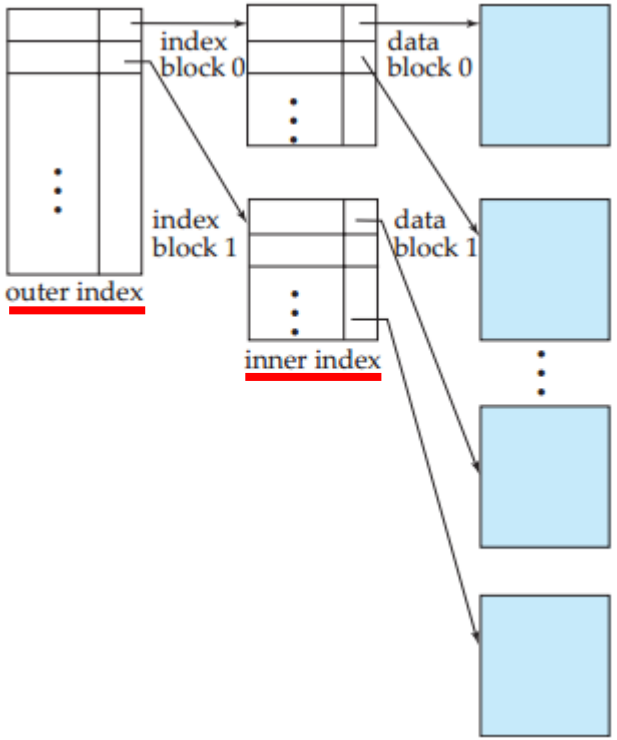
❖ O custo de processamento está normalmente associado a encontrar e trazer o bloco do ficheiro do disco, uma vez no bloco o tempo de procura dentro do mesmo é negligenciável

Índices Ordenados

Índices multinível

- Se o índice primário não "cabe" em memória o acesso torna-se dispendioso.

- De modo a diminuir o número de acessos a disco, tratar o índice primário como um ficheiro sequencial e criar um índice esparso sobre o índice primário:
  - **índice interno**:
    o ficheiro sequencial do índice primário
  - **índice exterior**:
    índice esparso do índice primário.



- Se o índice exterior não "couber" em memória, criar um novo nível de índice –> mais um nível.

- Todos os níveis dos índices devem ser atualizados nas operações de *insert, update* e *delete* !!
  (Update: pode ser abordado como um *delete* + *insert*)

Índices Secundários

Motivação

- Frequentemente, é necessário pesquisar registos por certos atributos, que não são o atributo da chave de pesquisa do índice primário ou *clustered*.

- Exemplo:
  na base de dados das contas bancárias, e os dados estão ordenados sequencialmente pelo número de conta.
  - Poderei pretender:
    a. obter todas as contas numa determinada agência
    b. obter as contas em que o saldo está num determinado intervalo de valores

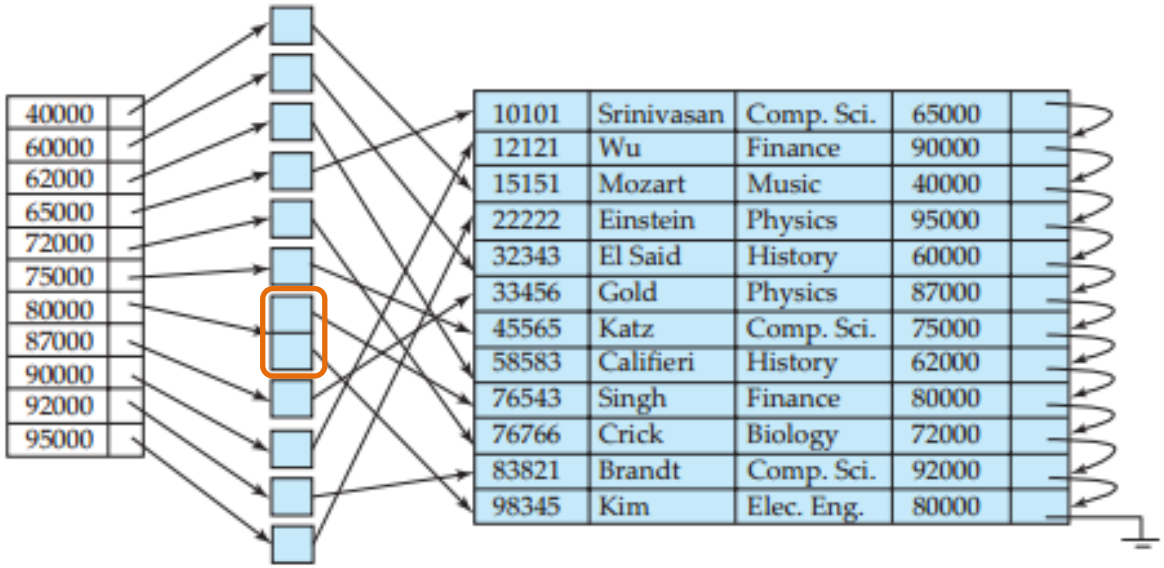
- Devo então criar índices secundários, para outras chaves de pesquisa.

Índices Secundários

Motivação

- Exemplo:
  Índice secundário pelo saldo de conta

✓ Os índices secundário terão de ser densos!



Índices Primários e Secundários

Considerações

- Os índices secundários têm de ser densos
- Os índices aumentam o desempenho na consulta de registos
- Quando a informação é modificada, cada ficheiro de índices também tem de ser atualizado
  - ❖ Introduz acréscimo de processamento quando existe modificação da informação.
- O varrimento sequencial sobre o índice primário é eficiente (os dados estão armazenados sequencialmente, na mesma ordem do índice)
- O varrimento sequencial sobre índices secundários é dispendioso
  - cada acesso a um registo pode implicar o acesso a um novo bloco do disco.

mini Sumário

1. Índices: Primário, Clustered, Secundário
2. Índices: Denso e Esparso

10:00

Exercícios

1. Distinga índice primário de índice *non-clustered*
2. É possível ter dois *clustered* índices na mesma relação/tabela?
3. Distinga índice denso de índice esparso
4. O índice secundário pode ser esparso? Porquê?
5. Num índice denso a remoção de um registo não originará necessariamente a remoção da chave do ficheiro do índice. Justifique.

Índices B$^+$-Tree

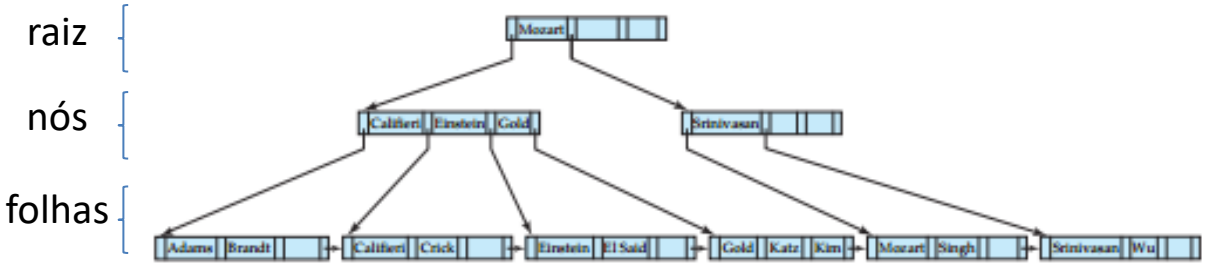
Características

- *Balanced Tree* como alternativa aos índices sequenciais:
- Desvantagem dos índices sequenciais:
  - O desempenho diminui à medida que a dimensão do ficheiro aumenta
  - É necessária reorganização periódica do ficheiro
- Vantagem dos índices B+-Tree:
  - Reorganização automática em alterações pequenas e locais, resultantes das operações de *insert* e *delete*
  - Não é necessária a reorganização total frequente do ficheiro de modo a manter o desempenho.
- Desvantagem dos índices B+-Tree:
  - Acréscimo de processamento em (alguns) insert e delete
  - Acréscimo de espaço ocupado

❖ As vantagens sobrepõem-se às desvantagens, por isso são comumente utilizados

Índices B$^+$-Tree

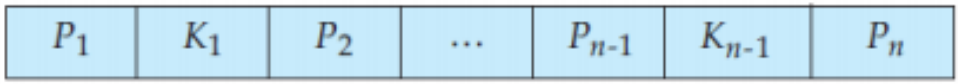
Características



- Normalmente índices densos
- Organização em árvore que têm as seguintes propriedades:
  - **Todos os caminhos da raíz às folhas têm o mesmo comprimento**
  - **Cada nó que não é raíz ou folha tem entre [n/2] e n filhos**
  - **Cada nó folha tem entre [(n-1)/2] e n-1 valores**
  - **Casos especiais:**
    - **Se a raíz não é folha tem pelo menos 2 filhos**
    - **Se a raíz é folha, pode ter entre 0 e (n-1) valores**

Índices B$^+$-Tree

Estrutura dos nós



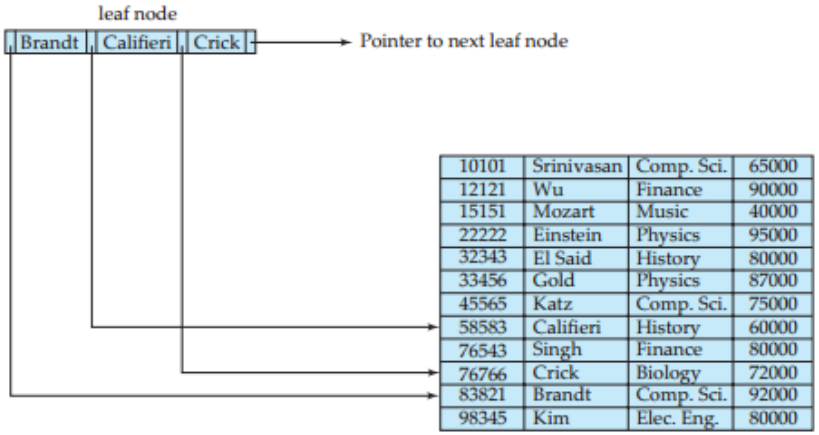
- Ki são os valores da chave de pesquisa
- Pi são os apontadores para os filhos (para nós que não são folha)
  ou,
  apontadores para registos ou *buckets* de registos (para os nós folha)

- Os valores da chave de pesquisa estão ordenados no nó
  K1 < K2 < K3 < ... < Kn-1

Índices B+-Tree

Estrutura dos nós folha

- O apontador Pi , aponta para um registo no ficheiro com o valor da chave de pesquisa Ki, (i = 1,2, …, n-1,)

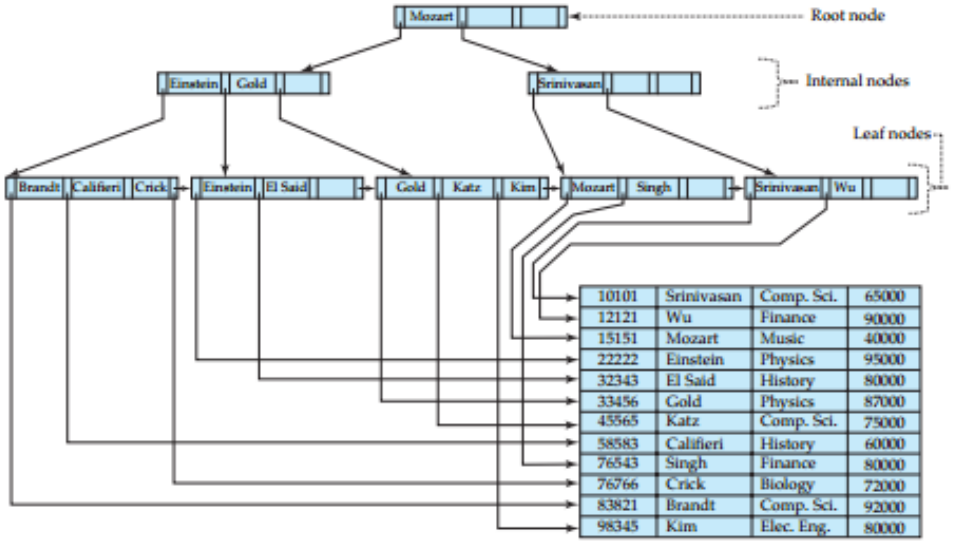


- P**n** aponta para a próxima folha ordenada pela chave de pesquisa

Índices B$^+$-Tree

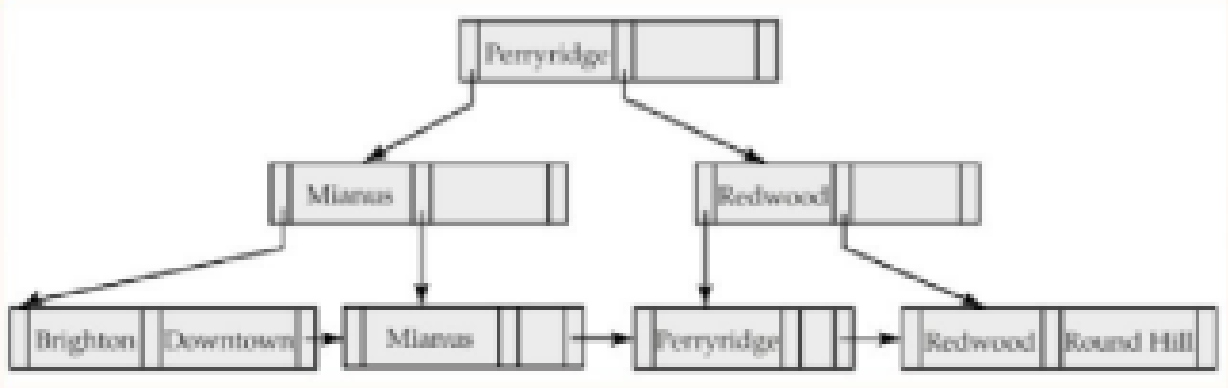
Estrutura dos nós não folha

- Constituem um índice esparso multinível dos nós folha
- Para um nó não folha com ***m*** (<n) apontadores:
  - A subárvore apontada por P1 conterá chaves de pesquisa de valor <= K1
- Para 2 <= i <= n-1,
  - todos os valores da chave de pesquisa da subárvore para a qual Pi aponta são maiores ou iguais a Ki-1 e menores que Km-1
- Contém até n apontadores, e pelo menos [n/2]
- O número de apontadores num nó é denominado de *fanout* do nó.



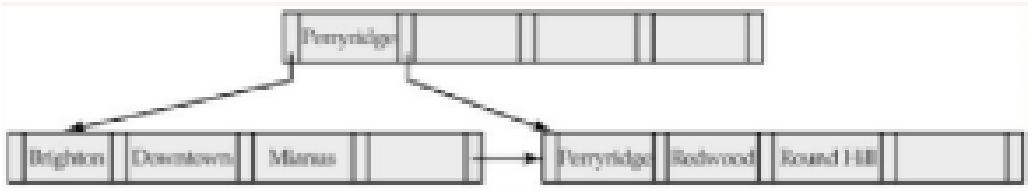
Índices B$^+$-Tree

Exemplo (n=3)



Índices B$^+$-Tree

Exemplo (n=5)



Numero de ponteiros no nó

fanout $\longrightarrow n$

min pointers por folha $\longrightarrow \lceil (n-1)/2 \rceil$

max pointers por folha $\longrightarrow (n-1)$

min pointers por nó $\longrightarrow \lceil (n)/2 \rceil$

max pointers por nó $\longrightarrow n$

- Os nós folha têm de ter entre 2 e 4 valores.
- Os nós não folha (exceto a raiz) têm de ter entre 3 e 5 ponteiros.
- A raiz tem de ter pelo menos 2 filhos.

Índices B$^+$-Tree

Altura da árvore e Espaço necessário

- Consideremos uma entrada no índice:
  12 (chave pesquisa) + 8 (apontador) = 20 Bytes
- Com páginas/blocos de 8KB
- Com cerca de 400 entradas por pagina
  - Assumindo paginas semipreenchidas contendo cerca de 250 entradas

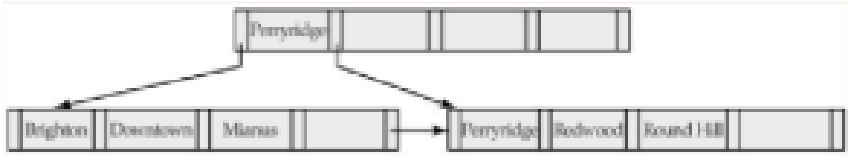
Nível	N.º de chaves	Tamanho (*bytes*)
1	250	8 K
2	$250^2 = 62\,500$	2 MB
3	$250^3 = 15\,625\,000$	500 MB

- Mesmo contendo 15 milhões de chaves, mantendo somente a raiz em memória (8K) é possível encontrar qualquer chave com um máximo de 2 acessos ao disco.

❖ Atualmente dada a memória disponível é possível com 1 ou mesmo nenhum acesso ao disco.

Índices B+-Tree

Consultas I

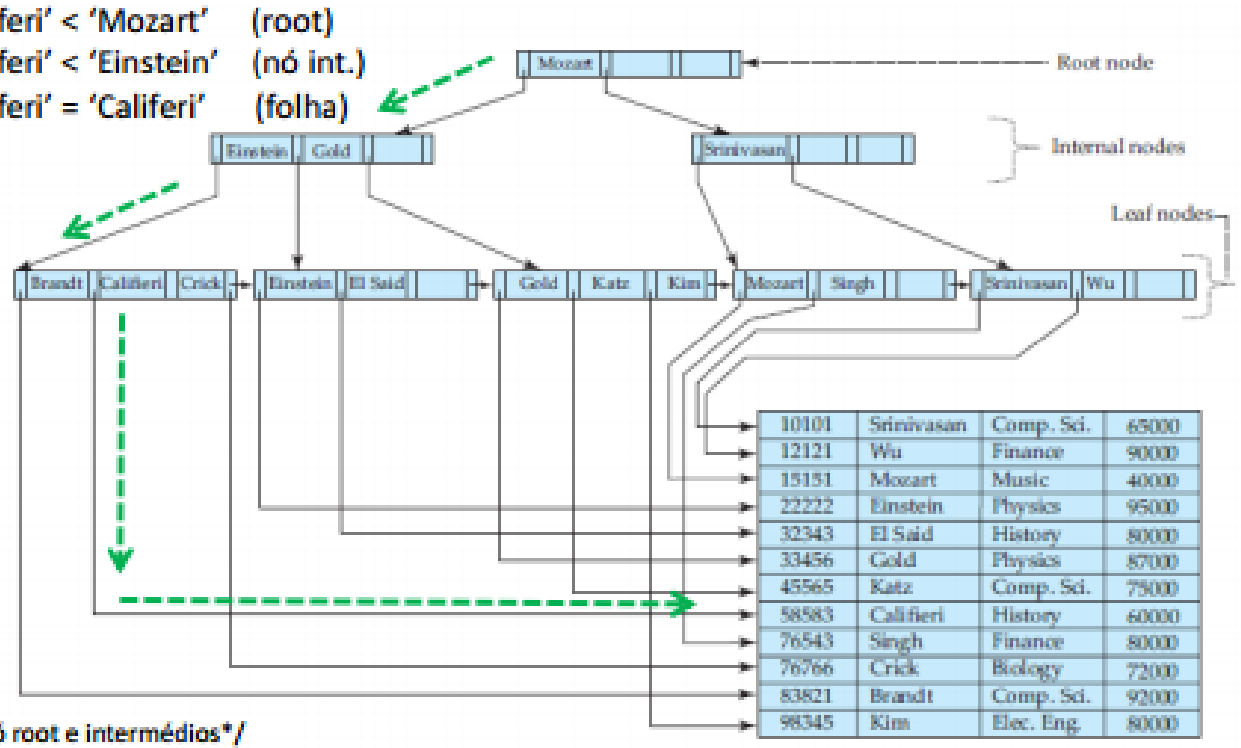


- Obter os registos com o valor K da chave de pesquisa
  1. Inicio no nó raiz
     i. Examinar o nó para o valor mais pequeno da chave de pesquisa > K
     ii. Se o valor existe, assumir que é Ki.
     Seguir para o nó filho apontado por Pi
     iii. Senão com K >= Km-1, seguir para o nó apontado por Pm
  2. Se o nó "seguido" pelo apontador do passo anterior não é folha, repetir o procedimento anterior.
  3. Quando no nó folha,
     para a chave i, Ki = K, seguir o apontador Pi para o registo ou *bucket*.
     :: Senão não existem registos com o valor K.

Índices B+-Tree

Consultas II

'Califeri'?
'Califeri' < 'Mozart' (root)
'Califeri' < 'Einstein' (nó int.)
'Califeri' = 'Califeri' (folha)



/*Nó root e intermédios*/
Procurar a menor chave superior ao valor
If (found) Seguir pointer da esquerda
Else Seguir pointer da direita

B+-tree for *instructor* file ($n = 4$).

Índices B^+-Tree

Consultas III

- No processamento de uma consulta, é obtido um caminho entre a raiz e uma folha
- Se existem K valores de pesquisa, o caminho não é maior que:
  $[log_{[n/2]}(k)]$

- Um nó tem normalmente a dimensão de um bloco de disco, tipicamente 4KB ou 8KB, e *n* por volta de 100
  (para 4KB temos 40 bytes por entrada de índice)

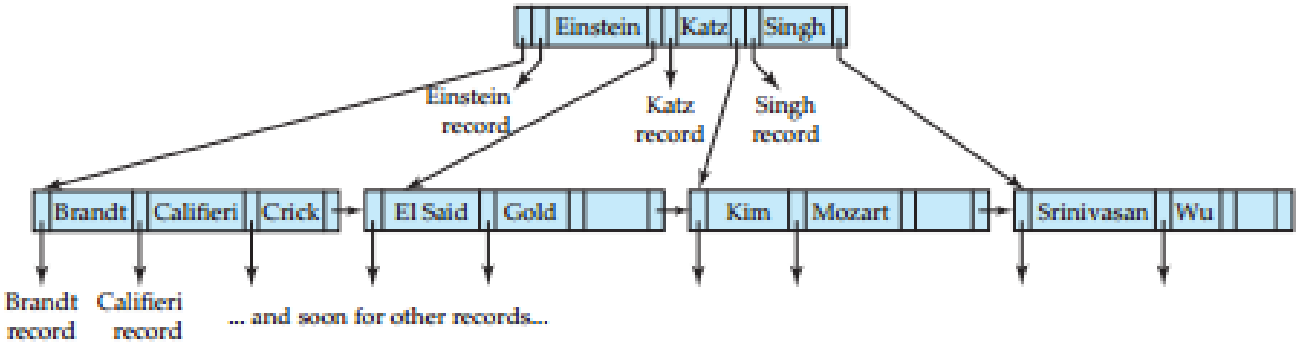
- Com 1 milhão de valores de pesquisa e n=100,
  no máximo são processados 4 (= $\log_{50}$ 1 000 000) nós

- Comparando com uma árvore binária (n=2) seria necessário percorrer 20 nós
  (= $\log_2$ 1 000 000)
  - A diferença é significativa pois um acesso a um nó pode ter associado um acesso ao disco

Índices B-Tree

Definição

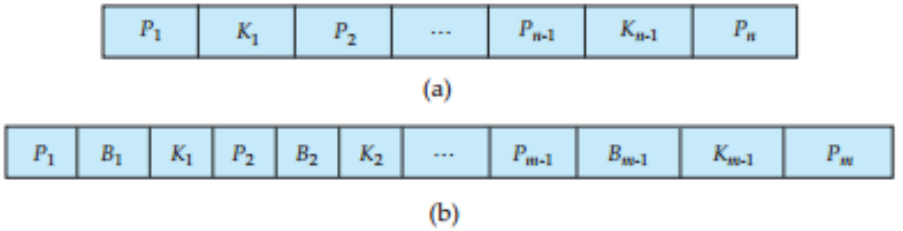
- Semelhantes às B^+-Tree, mas…
  elimina redundância de armazenamento dos valores das chaves de pesquisa, pois estes apenas aparecem uma vez na árvore



❖ Estrutura dos nós:

a) Folha

b) Não folha



Índices B-Tree

Características

- **Vantagens**:
  - Menor número de nós
  - Por vezes não é necessário percorrer a árvore até às folhas para obter o valor da chave de pesquisa.
- **Desvantagens**
  - Só uma pequena fração de todos os valores da chave de pesquisa são obtidos mais "cedo"
  - Os nós não folha têm maior dimensão
  - As operações de *insert* e *delete* têm processamento mais complicado

❖ Tipicamente as vantagens não se sobrepõem às desvantagens

mini Sumário

1. Índices não sequenciais
2. B+-Tree *vs* B-Tree

10:00

Exercícios

1. Que vantagem apresentam as B+-Tree face às árvores binárias?

2. Que vantagens apresentam os índices B-Tree sobre os B+-Tree? Sobrepõem-se às desvantagens?

3. Que vantagens apresentam os índices B+-Tree face aos índice sequenciais?

Hashing - estático

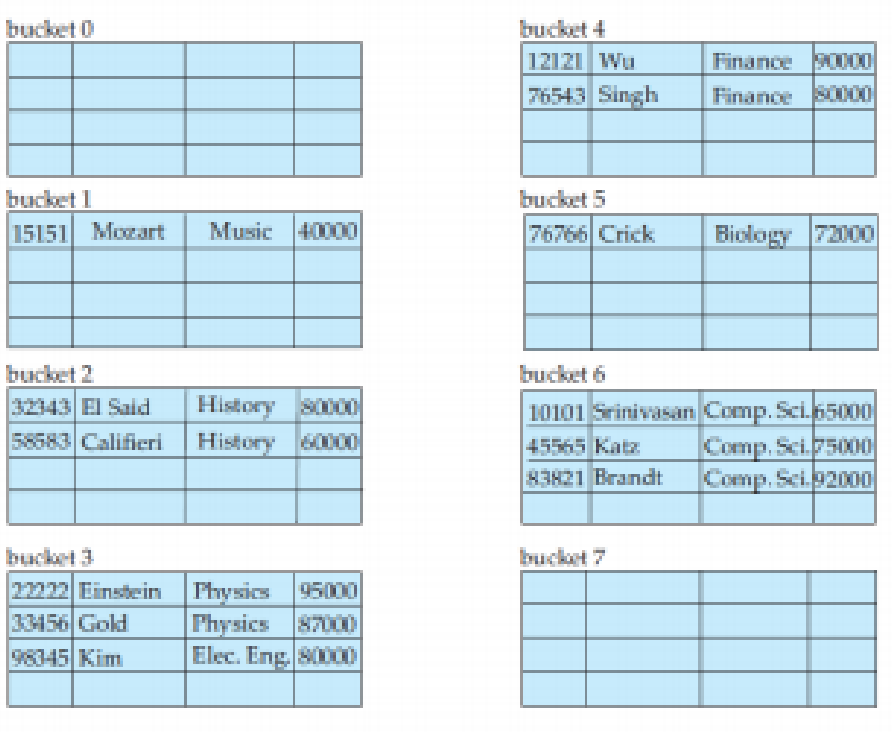
Definições

- Um *bucket* é uma unidade de armazenamento, que contém um ou mais registos (tipicamente um *bucket* corresponde a um bloco de disco).
- Num ficheiro com organização *hash*, o acesso direto a um *bucket,* através do valor da chave de pesquisa, é obtido utilizando uma função de *hash*
- Com K o conjunto de valores da chave de busca e
  B o conjunto de todos os *buckets*,
  H é uma função de hash de K para B
- A função de *hash*, é utilizada para aceder, inserir e remover registos
- Registos com diferentes valores de pesquisa, podem estar associados ao mesmo *bucket*;
  - assim o *bucket* tem de ser "varrido" sequencialmente para localizar o registo.

Hashing estático

Exemplo

Ficheiro com organização *hash*



bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hashing estático

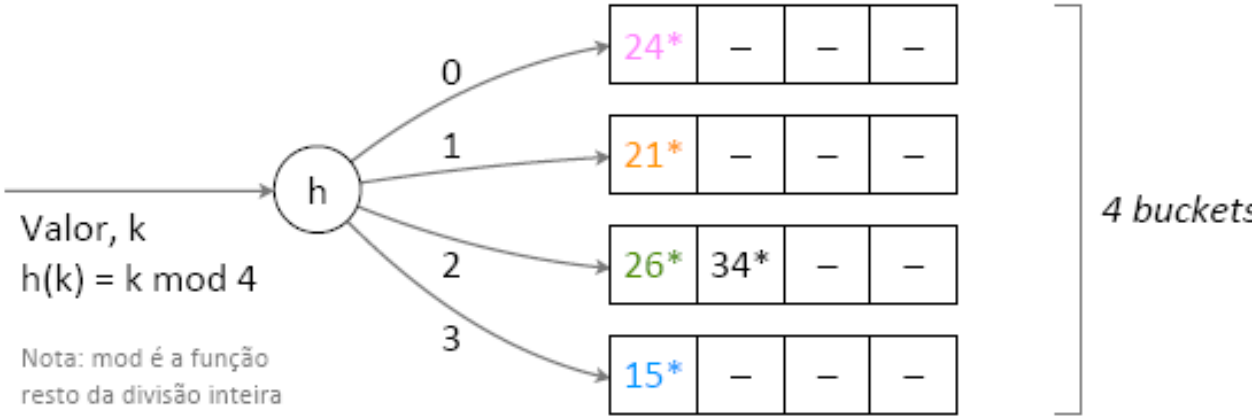
Funções de *Hash*

- No pior caso, a função hash mapearia todos os valores da chave de pesquisa, para o mesmo *bucket*;
  - assim o tempo de acesso é proporcional ao número de valores da chave de pesquisa existentes
- A função *hash* (dispersão) deve cumprir os seguintes requisitos:
  - uma função **uniforme**,
    ou seja cada *bucket* é atribuído o mesmo número de valores da chave de pesquisa (de todo o intervalo de valores possíveis).
  - uma função ***aleatória***,
    em que cada *bucket* contém, a cada momento, em média, aproximadamente o mesmo número de valores da chave de pesquisa.
- Tipicamente as funções calculam e utilizam a representação binária dos valores da chave de busca (por exemplo a representação binária dos caracteres)

Hashing estático

Funções Hash (o principio)

- Função h dispersa valores indexados por vários *buckets*
  - Cada valor indexado fica num só *bucket*



Hashing estático

Funções Hash (exemplos)

Organização *hash* do ficheiro de contas bancárias, utilizando o nome da agência como chave de pesquisa

- Assumir 26 *buckets*, em que a função de *hash*, associa o valor binário *i* do primeiro caracter do nome, com o *bucket i*;

❖ Não se obtém uma distribuição uniforme, pois é de esperar mais nomes a começar com a letra A do que com a letra X.

Hashing estático

Funções Hash (exemplos II)

Organização *hash* do ficheiro de contas bancárias, utilizando o saldo como chave de busca:

- Assumir valor mínimo 1 e máximo 100.000;
- Assumir 10 *buckets*, com os intervalos de 1-10.000, 10.001-20.000, ...
- A distribuição é uniforme pois cada *bucket* está associado com 10.000 valores da chave

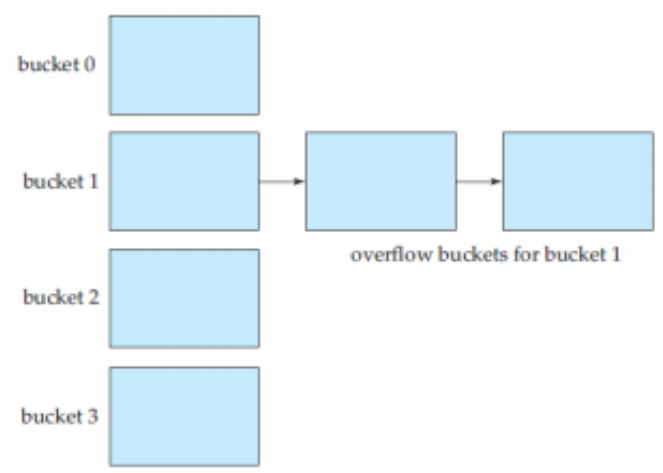
❖ Não é aleatória pois é de esperar que existam mais valores entre 1-10.000 do que em 90.001 e 100.000.

Hashing estático

Bucket overflow

Pode ocorrer por:

- Número de *buckets* insuficiente;
- Alguns *buckets* têm mais valores que outros, denominado como *bucket skew*:
  - múltiplos registos com o mesmo valor da chave de pesquisa;
  - a função de *hash* escolhida, resulta numa distribuição não uniforme dos valores da chave de pesquisa



- Solução:
  - Cadeias de *overflow*: os *buckets* de *overflow* são associados através de uma lista. É denominado *closed hashing*.

Índices *Hash*

Definições

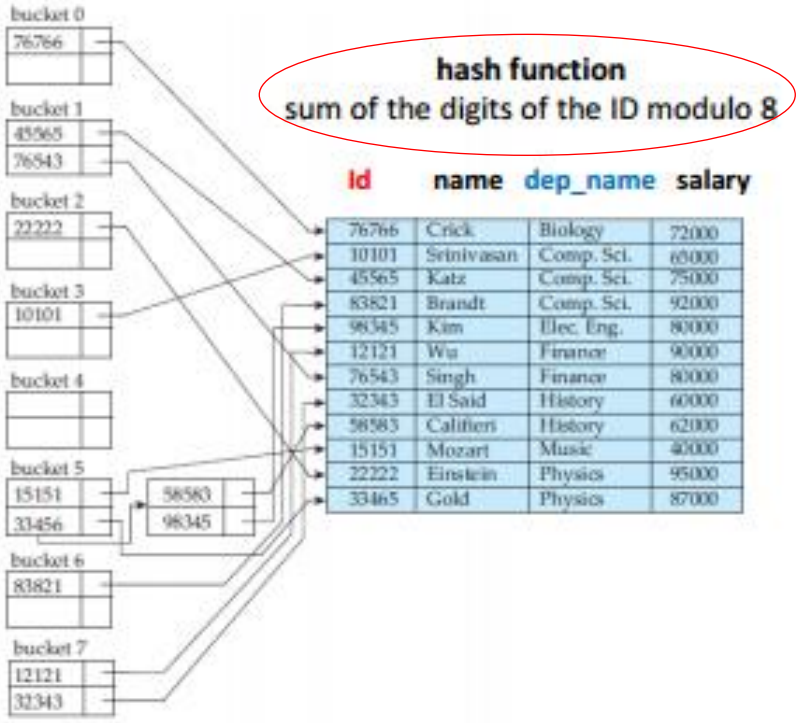
- Um índice *hash*, organiza os valores da chave de pesquisa, com os respectivos apontadores para os registos, num ficheiro com uma estrutura de *hash*.

❖ Índices *hash*, são sempre índices secundários

- ➢ ***Porquê ?***

❖ Eficientes em consultas de igualdade

- ➢ ***Porquê ?***



Índices *Hash*

Deficiências

- A função de *hash* associa os valores na chave a um conjunto fixo de *buckets*:
  - Se o número de *buckets* é reduzido, o aumento da BD, provoca *bucket overflow*, logo diminuição do desempenho
  - Se o número de buckets é elevado (prevendo o aumento da BD), temos desperdício de espaço inicialmente.
  - Se a BD diminui, temos desperdício de espaço (devido ao *bucket overflow*)
    - Pode-se optar por uma re-organização periódica, mas é muito dispendiosa

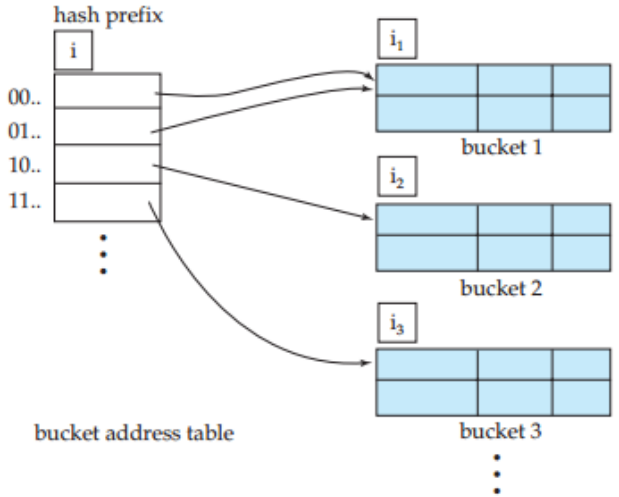
- Solução:
  ***Hash* dinâmico**

Hash Dinâmico

Características

- Eficiente para bases de dados que aumentam e diminuem de dimensão
- Permite a modificação dinâmica da função de *hash*
  - Em função do tamanho dos dados
    (embora processo condicione temporariamente acesso a alguns blocos)

- *Hash* extensível (uma forma de *hash* dinâmico)
  - a função de *hash* gera valores dentro de um intervalo alargado
    - tipicamente inteiros de 32 bits
  - Utiliza a cada momento um prefixo para endereçar um conjunto de *buckets*
  - Sendo *i* o comprimento do prefixo e 0<= i <=32;
  - Número máximo de buckets: 2^{32}
  - O valor de *i* varia com a dimensão da base de dados.



Hash Dinâmico

Exemplo

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hash function for *dept_name*.

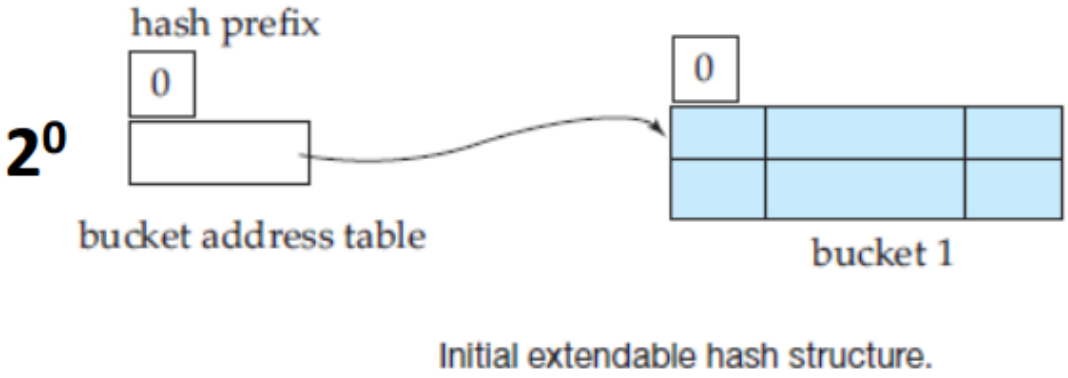
search key = dept_name -> 32-bit hash values

Hash Dinâmico

Exemplo

Situação inicial

Pressuposto (*didático*): *bucket* só poderá conter **2 registos**



Initial extendable hash structure.

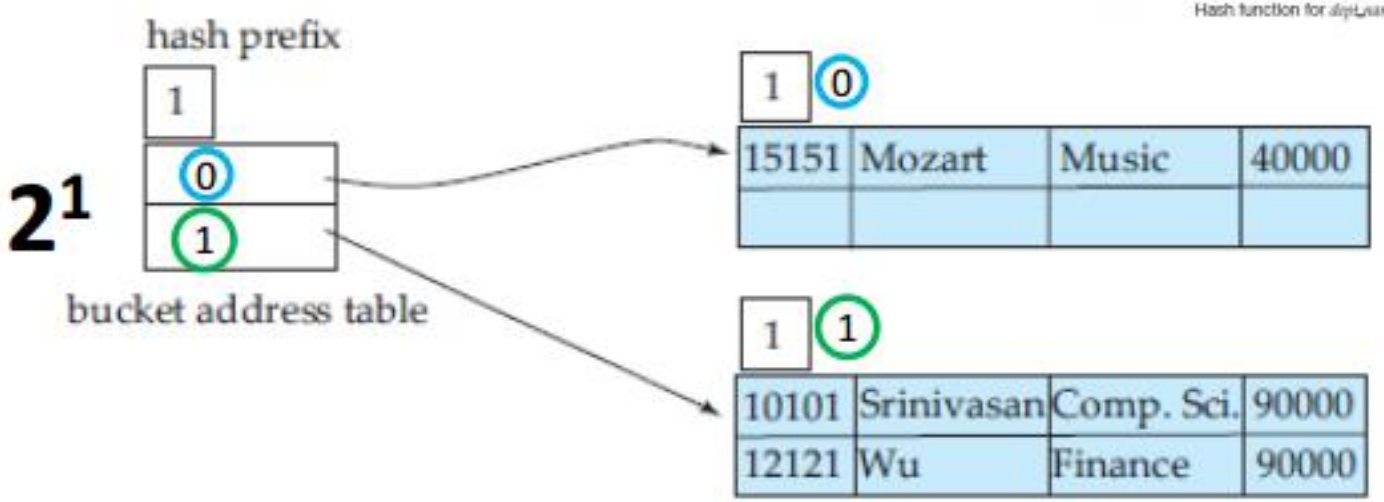
Hash Dinâmico

Exemplo

Inserção:
(10101, Srinivasan, **Comp. Sci.**, 65000)
(12121, Wu, **Finance**, 90000)
(15151, Mozart, **Music**, 40000)

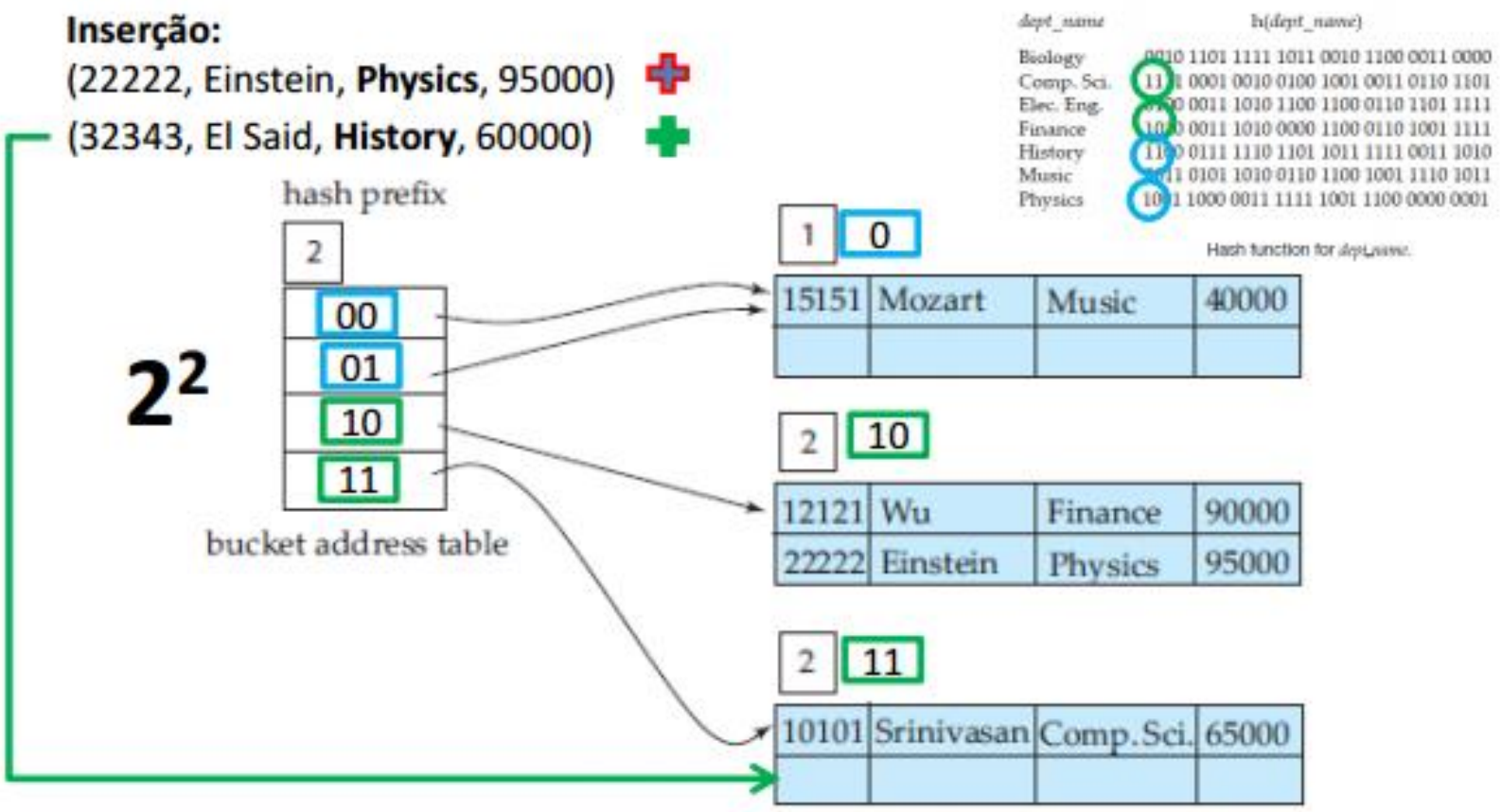
dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1011 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hash function for *dept_name*.



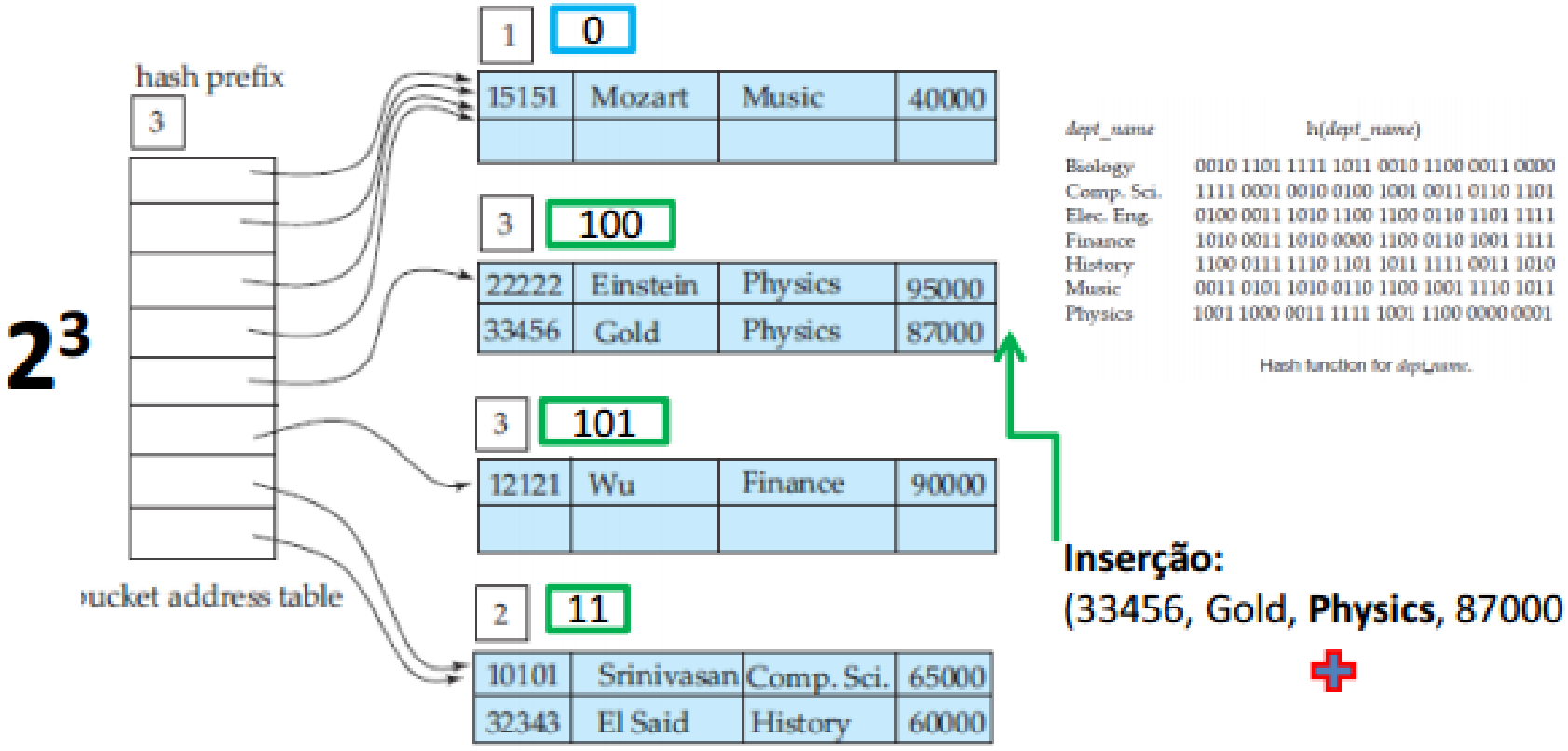
Hash Dinâmico

Exemplo



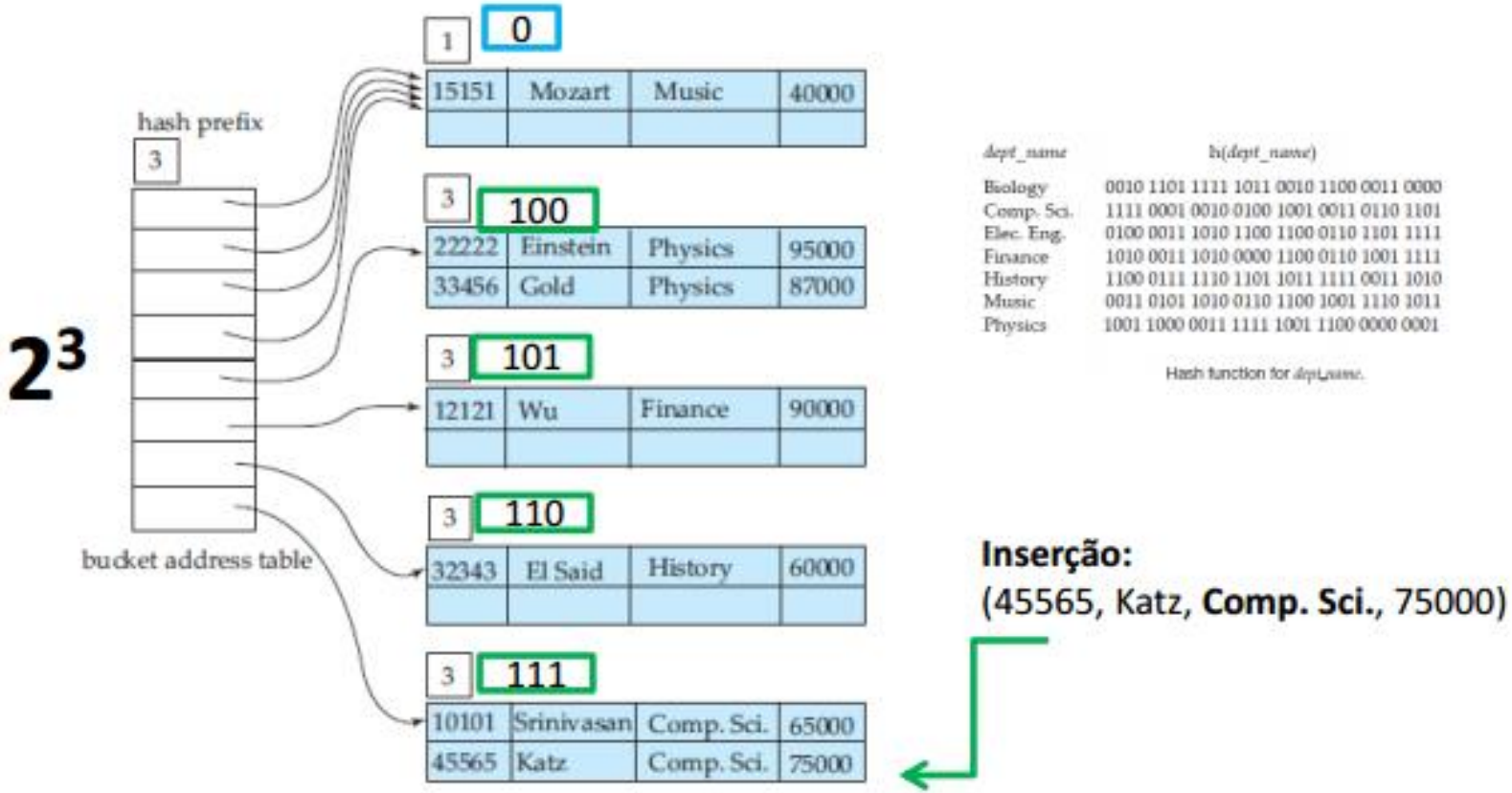
Hash Dinâmico

Exemplo



Hash Dinâmico

Exemplo



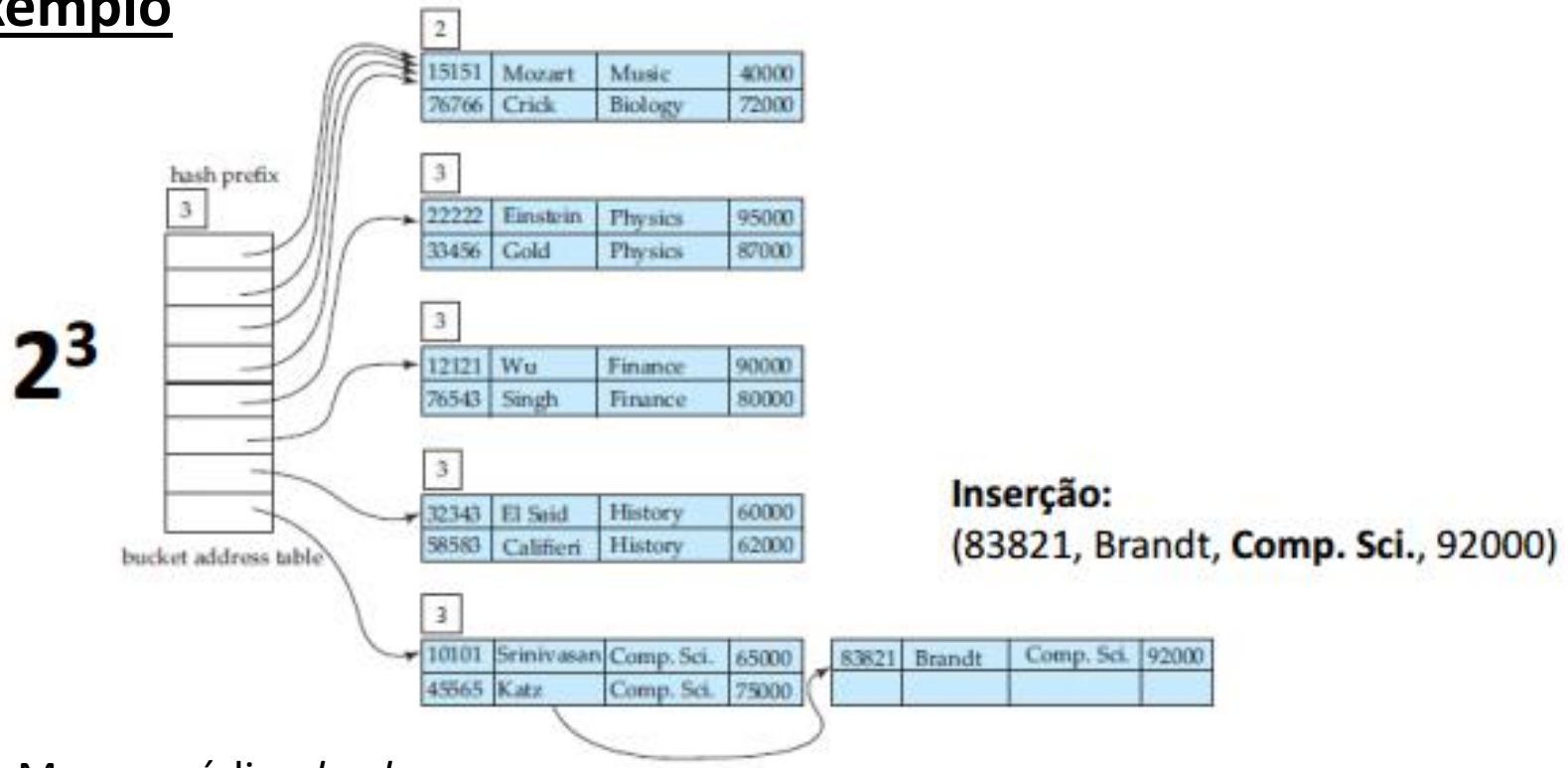
dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Hash function for *dept_name*.

Inserção:
(45565, Katz, **Comp. Sci.**, 75000)

Hash Dinâmico

Exemplo



- Mesmo código *hash*
- Não aumenta numero do prefixo, mas segue para *bucket overflow*

Comparação

***Hash* extensível *vs* outros métodos**

- Vantagens
  - O desempenho não se degrada com o aumento da dimensão do ficheiro;
  - *Overhead* de espaço mínimo.

- Desvantagens
  - Nível extra para obter o registo procurado;
  - A tabela de endereços de *buckets*, pode tornar-se muito grande (não caber em memória);
  - Alterar a dimensão da tabela de endereços de *buckets*, é uma operação dispendiosa.

✓ Mais adequado para critérios de pesquisa baseados em igualdades

mini Sumário

1. Índices Hash
2. Hash estático *vs* dinâmico

05:00

Exercícios

1. O que entende por *bucket overflow*?
2. Quais as vantagens do hash dinâmico?
3. Em que tipo de consultas os índices hash poderão não ser eficientes?

Complementos de Bases de Dados
– Índices –

Engenharia Informática

2º Ano / 1º Semestre

Cláudio Miguel Sapateiro
claudio.sapateiro@estsetubal.ips.pt

DSI :: Escola Superior de Tecnologia de Setúbal :: Instituto Politécnico de Setúbal