centro universitário FEI

AULA 3- ARQUIVOS E ÍNDICES (PARTE DOIS)

PROFA. DRA. LEILA BERGAMASCO

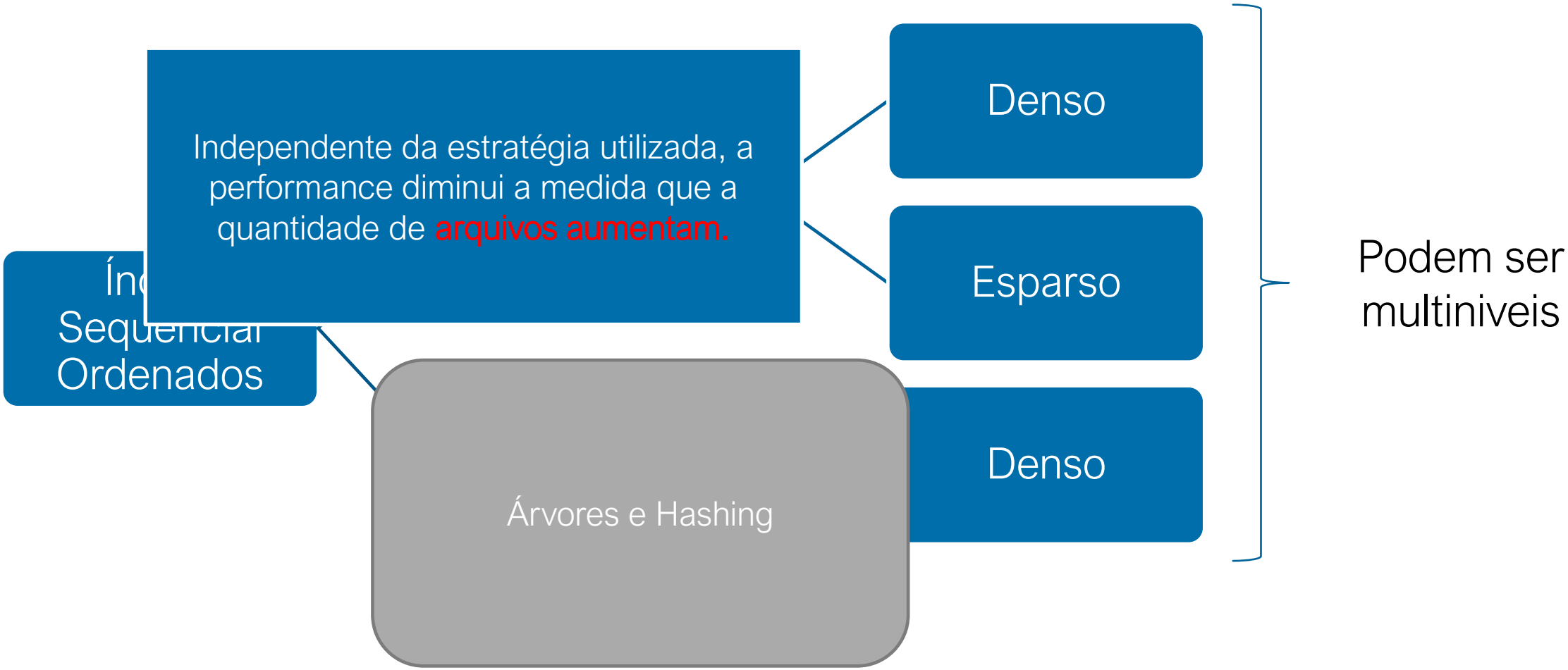
CC6240 – Tópicos Avançados em Banco de Dados

centro universitário FEI

AGENDA

- Índices Btree
- Índices Hash

RECAPITULANDO



centro universitário FEI

ÍNDICES BTREE

ÁRVORES B+

- Árvore balanceada: o caminho da raiz até o nó folha é do mesmo tamanho
  - Cada nó não folha tem entre n/2 e n nós filhos.

- Desvantagens:
  - sobrecarga na inserção/exclusão
    - Porém → "aceitável" já que se evita o custo de reorganização.
  - Espaço desperdiçado: os nós podem estar vazios até a metade
    - Porém →custo de armazenamento é "melhor" do que custo de performance

ÁRVORE B+

P1	K1	P2	K2	....	Pn-1	Kn-1	Pn

- n-1 chaves de busca K e seus n ponteiros P que apontam para os

numero_conta	nome_agência	saldo	
A-217	Bairro Assunção	750	
A-101	Bairro dos Casa	500	
A-110	Bairro dos Casa	600	
A-215	Centro	700	

ÁRVORE B+

Reparem que nome_agência está sempre ordenado no índice mas não precisa estar do arquivo.

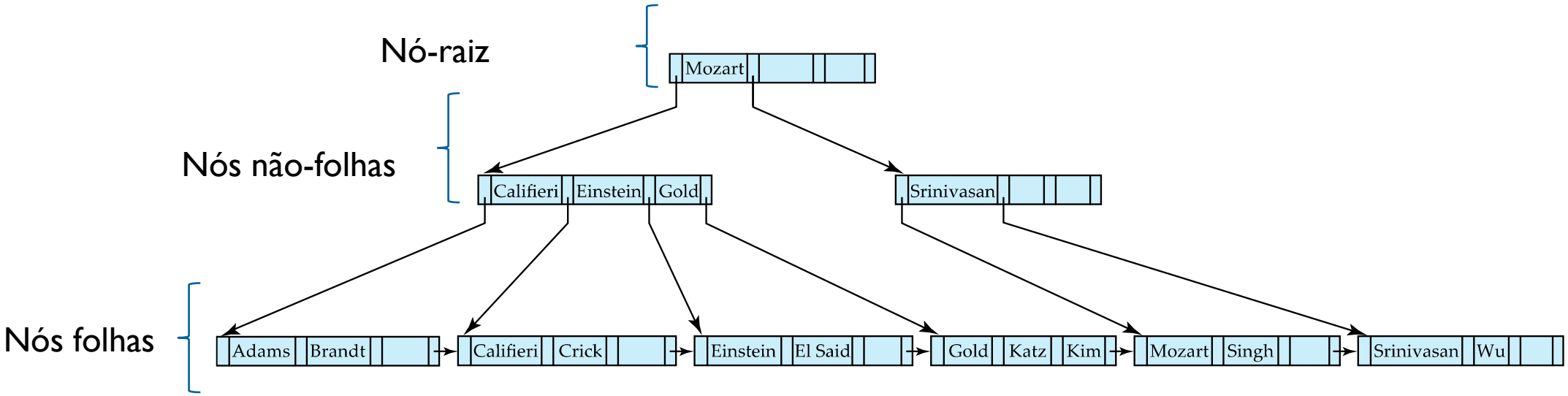
P1	K1	P2	K2	....	Pn-1	Kn-1	Pn

- n chaves de busca K e seus n ponteiros P que apontam para os registros

numero_conta	nome_agência	saldo	
A-101	Bairro dos Casa	500	
A-110	Bairro dos Casa	600	
A-215	Centro	700	
A-217	Bairro Assunção	750	

ESTRUTURA ÁRVORES B+

- Cada nó folha pode armazenar até n-1 valores E pelo menos (n-1)/2 valores
  - Arquivo conta com n=4
- Valores de nó folha não se sobrepõe
  - Pn encandeia nós folhas na ordem da chave de busca
- Nó-não folha possui até n ponteiros E pelo menos n/2 ponteiros

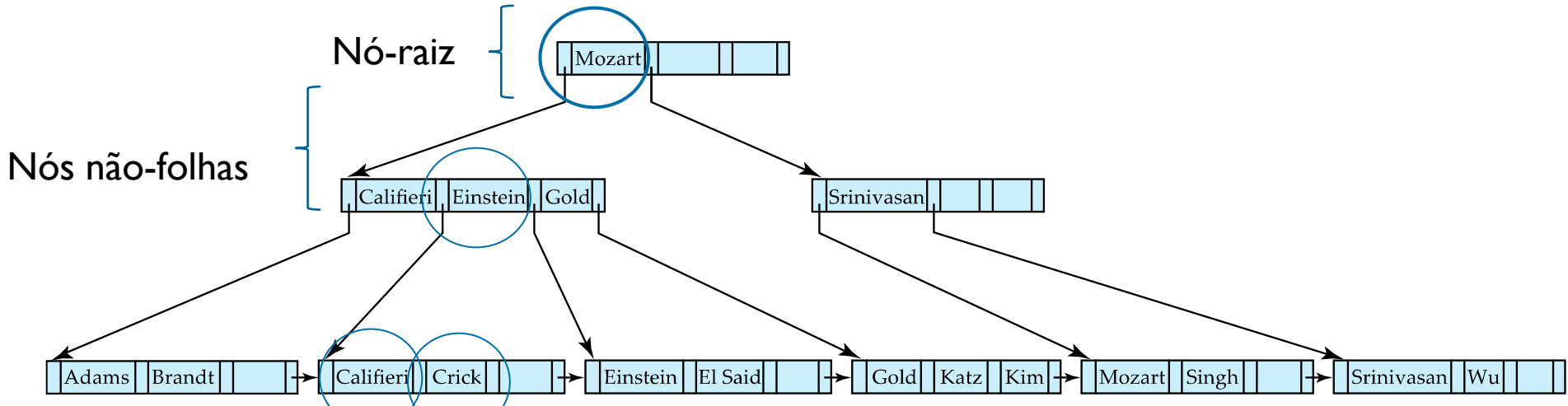


1Kb = 1024 bytes

CONSULTAS

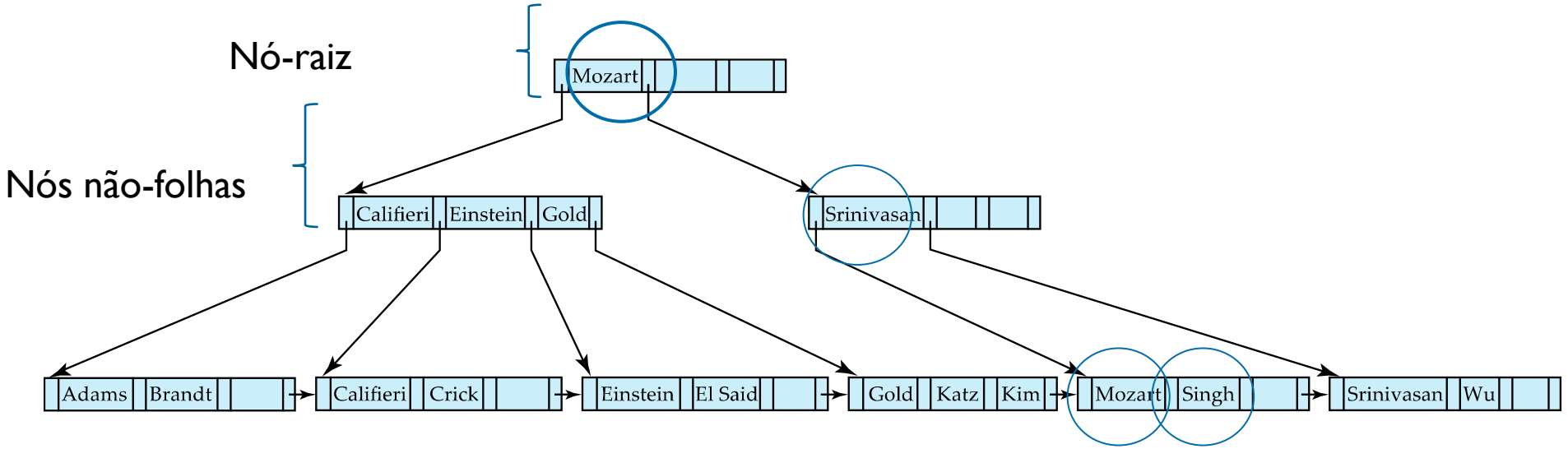
- Sempre custo não maior que log $_{n/2}$(K)
  - Exemplo: n=4 e K(chave de busca)=13
    - Não podemos acessar mais que 4 nós
- Para v=Crick
- 1 nó = 1 bloco de disco = 4kb
- Se chave de busca = 32 bytes e ponteiro = 4bytes
  - ~113 nós por bloco!

1000000 de chaves de registro, com n=100 → apenas 4 acessos a blocos.



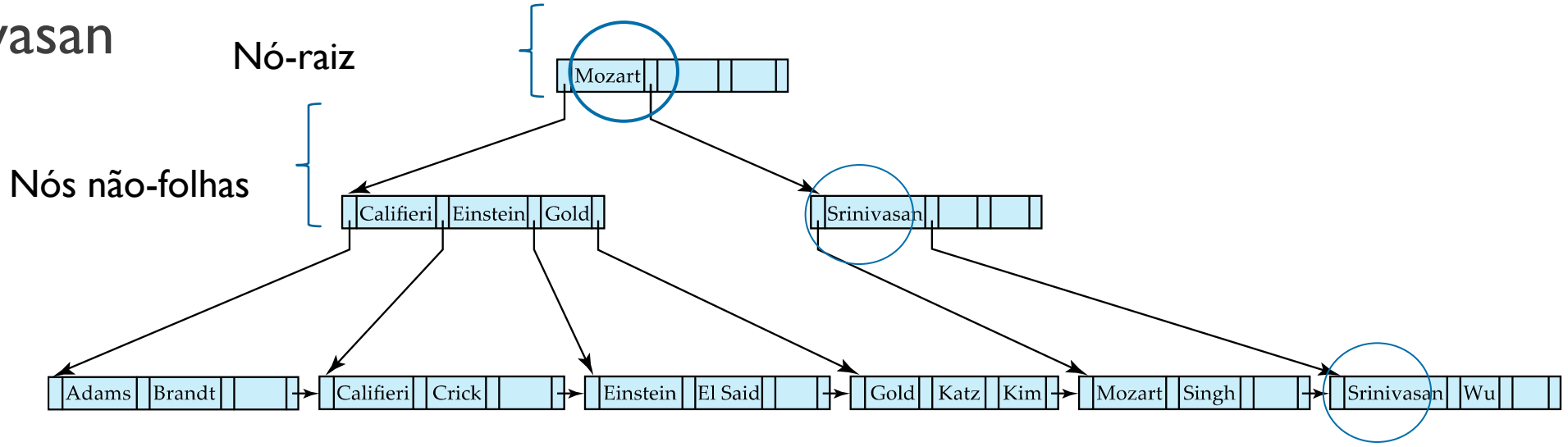
CONSULTAS

- Para v=Crick
- Para v=Singh



CONSULTAS

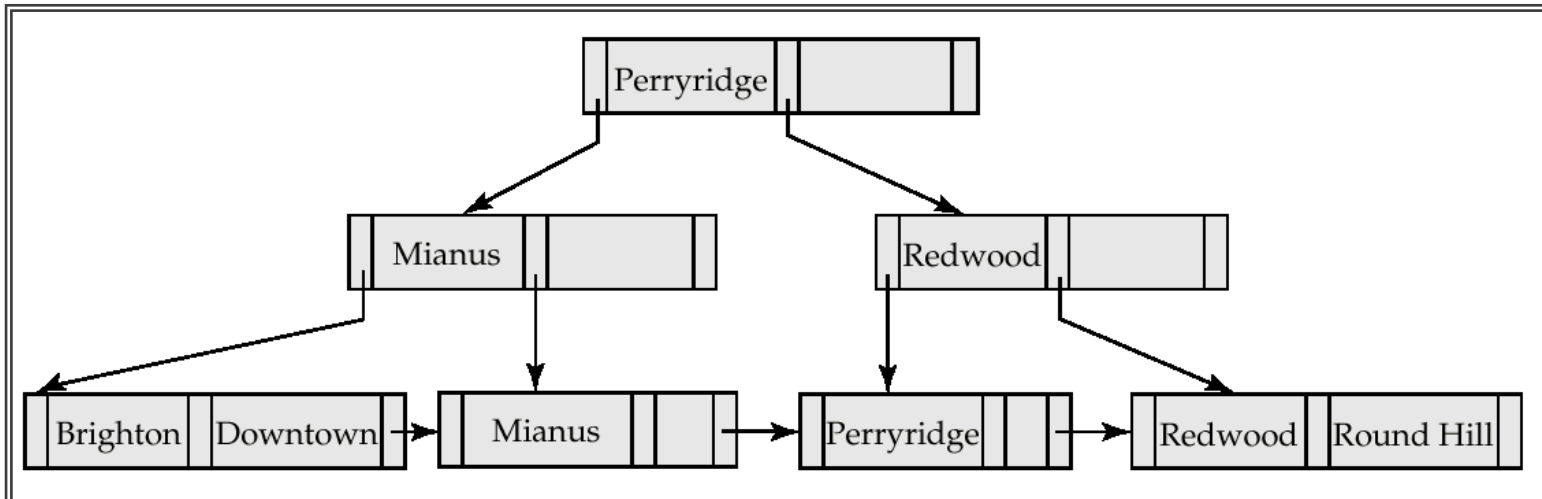
- Para v=Crick
- Para v=Singh
- Para v=Srinivasan



https://www.youtube.com/watch?v=YZECPU-3iHs

INSERÇÃO

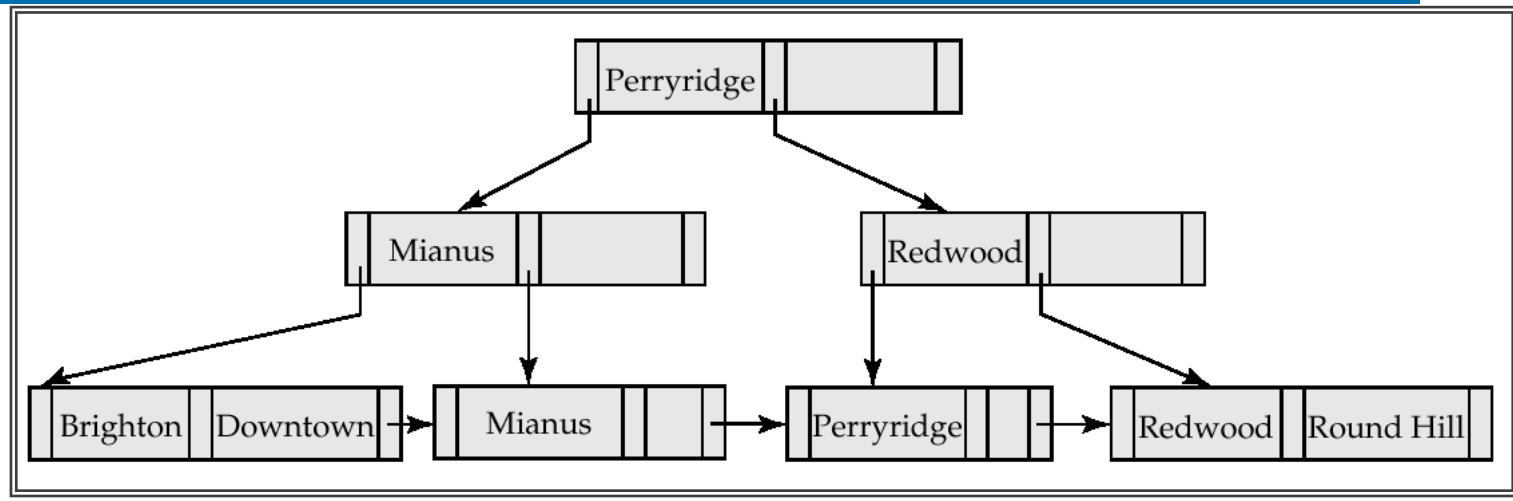
- Há uma redistribuição entre os nós para garantir o balanceamento.
  - Se nó tiver menos que n/2 ponteiros
- Pior caso: nó raiz precisa ser dividido e altura da árvore aumenta 1
- Se já tiver a chave de busca → adiciona o registro
- Se não tiver → adiciona a chave, faz o balanceamento, adiciona o registro
- Adicionar → Clearview (n=3)

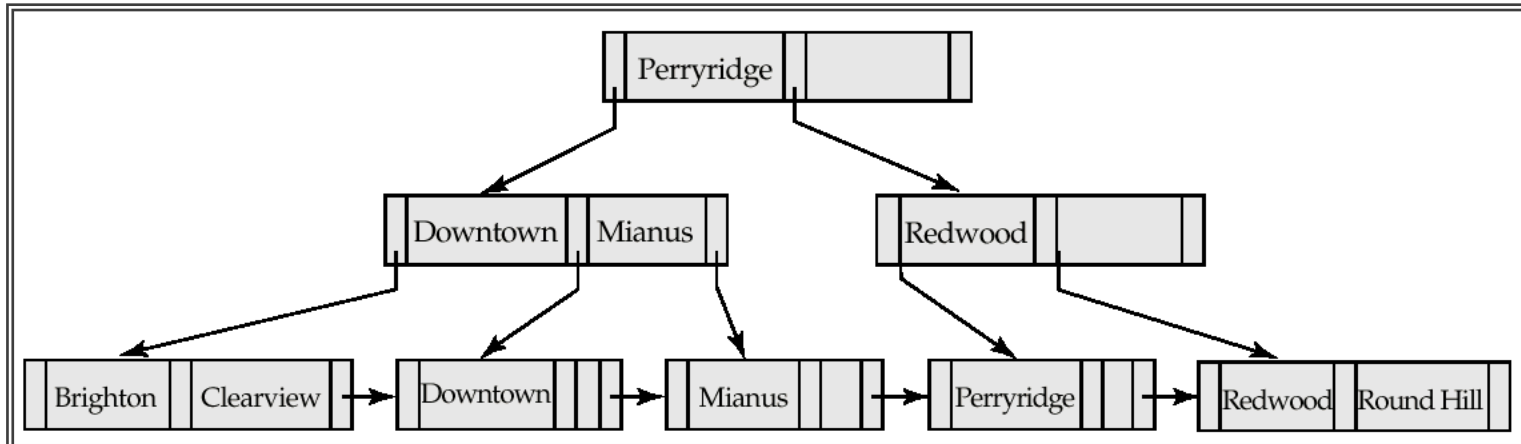


https://www.youtube.com/watch?v=YZECPU-3iHs

INSERÇÃO

- Colocamos os n/2 primeiros valores no nó existente e o restante no outro nó
  - Brighton – Clearview – Downton
  - 3/2 = 1,5 → 2
- Atualizamos nó pai
  - Com o menor valor da chave de busca do novo nó folha
  - Se não tiver espaço – divide o pai

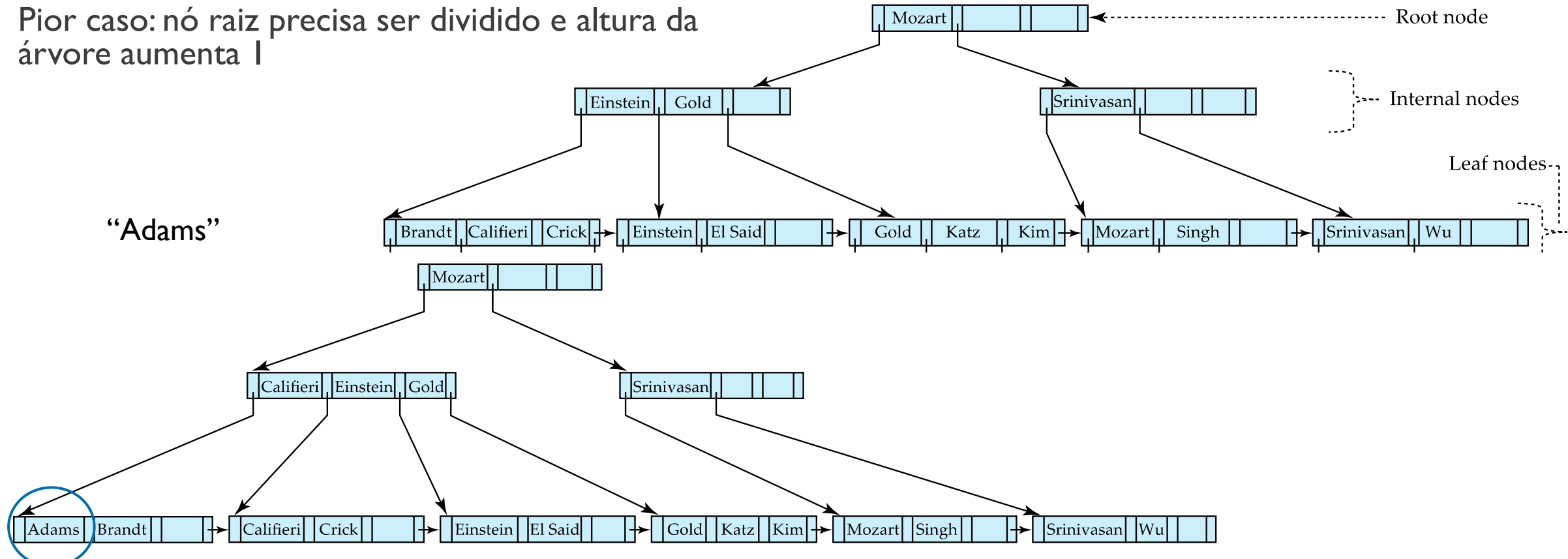




https://www.youtube.com/watch?v=47DUnfH1D0w

INSERÇÃO

- Há uma redistribuição entre os nós para garantir o balanceamento.
  - Se nó tiver menos que n/2 ponteiros
- Pior caso: nó raiz precisa ser dividido e altura da árvore aumenta 1



EXCLUSÃO

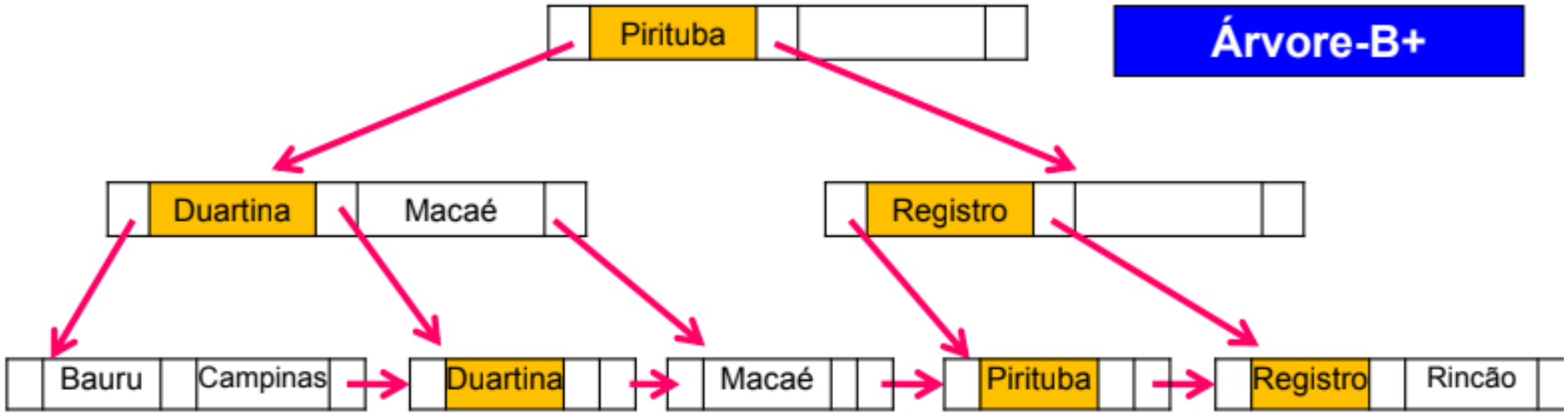
https://www.youtube.com/watch?v=YZECPU-3iHs

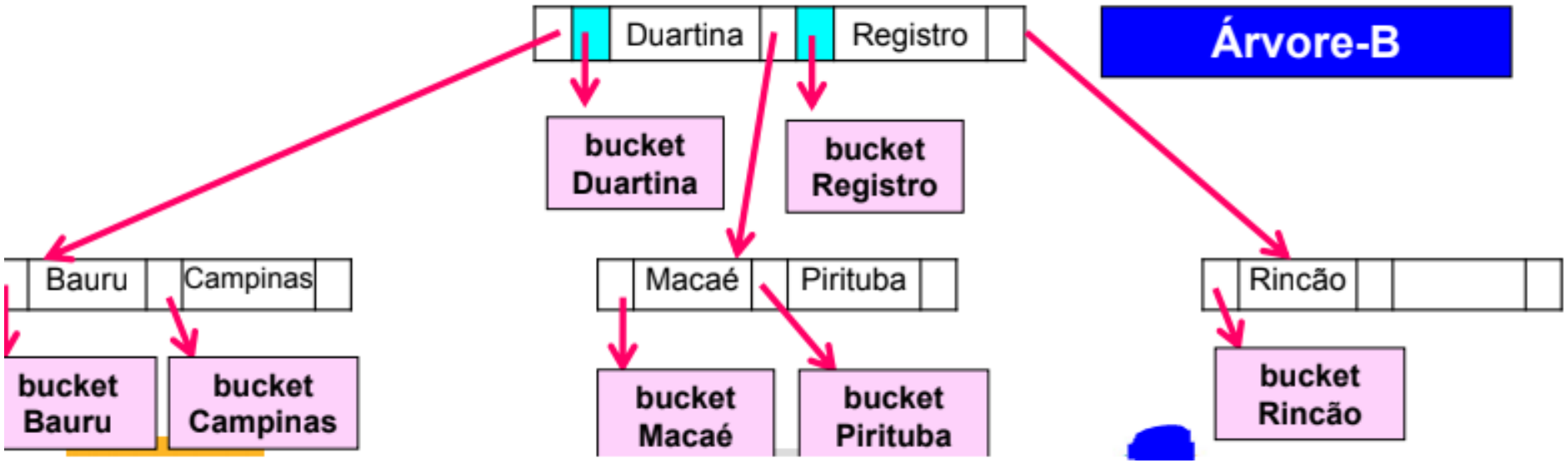
- Mesma lógica que inserção.
  - Procura chave, exclui e faz o balanceamento se necessário.
    - Nó-folha vira pai (vice-versa)
    - Nós se fundem

centro universitário FEI

ÍNDICES ÁRVORES B

- Semelhante índices Árvore-B+
- Diferença: Árvore-B elimina armazenamento redundante de valores de chaves de procura
  - podemos armazenar índice com número menor de nós;
- chaves de procura dos nós-folhas não aparecem em nenhuma outra parte da árvore-B:
  - temos que incluir um ponteiro adicional para cada chave de procura em um nó não-folha
  - ponteiros adicionais apontam para registros de arquivos ou para o bucket da chave de procura associada.


Árvore-B+
Pirituba
Duartina
Macaé
Registro
Bauru
Campinas
Duartina
Macaé
Pirituba
Registro
Rincão


Árvore-B
Duartina
Registro
bucket Duartina
bucket Registro
Bauru
Campinas
Macaé
Pirituba
Rincão
bucket Bauru
bucket Campinas
bucket Macaé
bucket Pirituba
bucket Rincão

INSERÇÃO E EXCLUSÃO – ÁRVORES B

- Mais complexas que B+
  - Necessário verificar nó-pai
  - Re-arranjos posteriores a inserção/Remoção talvez sejam necessários

- Logo, vantagens de espaço das Árvore-B são poucas
  - • para grandes índices e normalmente não superam desvantagens citadas
- Árvore-B+ é mais simples
  - • por isso, preferida por muitos projetistas de BD.

ORGANIZAÇÃO DE ARQUIVOS USANDO ÁRVORES

- Até agora falamos de estruturar os índices utilizando árvores, porém é possível organizar os próprios arquivos utilizando a mesma estratégia

- Àrvore B+ mais utilizada pelos mesmos motivos que índices

centro universitário FEI

ÍNDICES HASH

centro universitário FEI

HASHING

- Mesmo utilizando árvores, os nós-folha armazenam os dados de forma sequencial.
- Além disso é obrigatória a utilização de índices.
- A estratégia de hashing elimina, em alguns casos, a utilização de índices e também fornece meios de criar índices otimizados

centro universitário FEI

CONCEITOS

- Buckets: unidade de armazenamento. Pode armazenar 1 ou mais registros.
- Para simplificar: 1 bucket = 1 bloco de disco

- K = conjunto de valores de todas as chaves de busca
- B = conjunto de todos os valores de buckets
- h = função de hash. Mapeia K em B
- Logo, "inserir um registro com chave de busca = Ki, calculamos a função hash h que oferecerá o endereço correto do bucket que armazenará o registro." --- end_B = h(Ki)

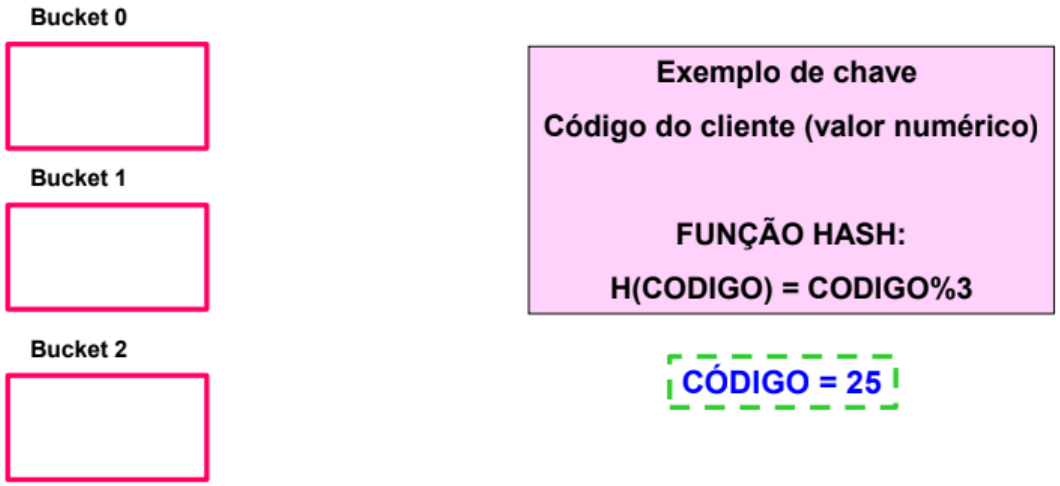
centro universitário FEI

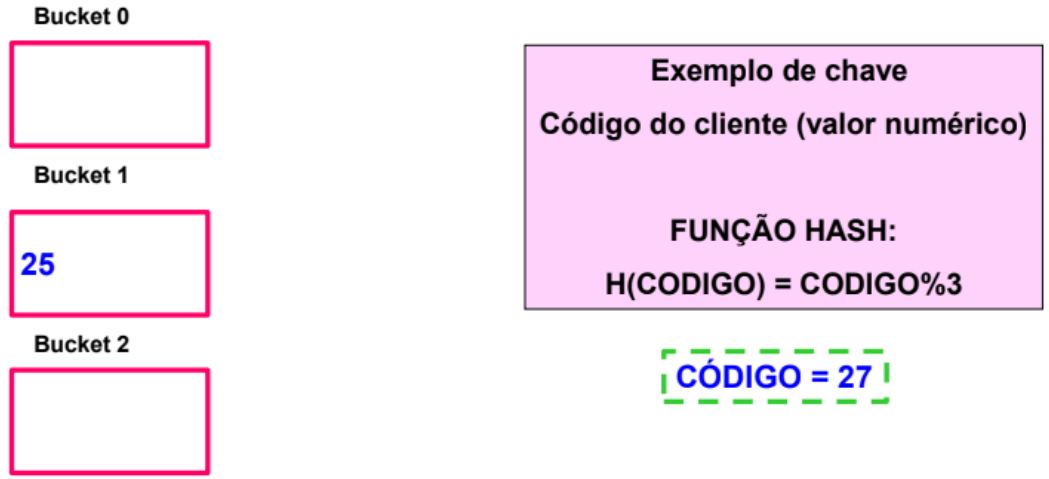
CONCEITOS

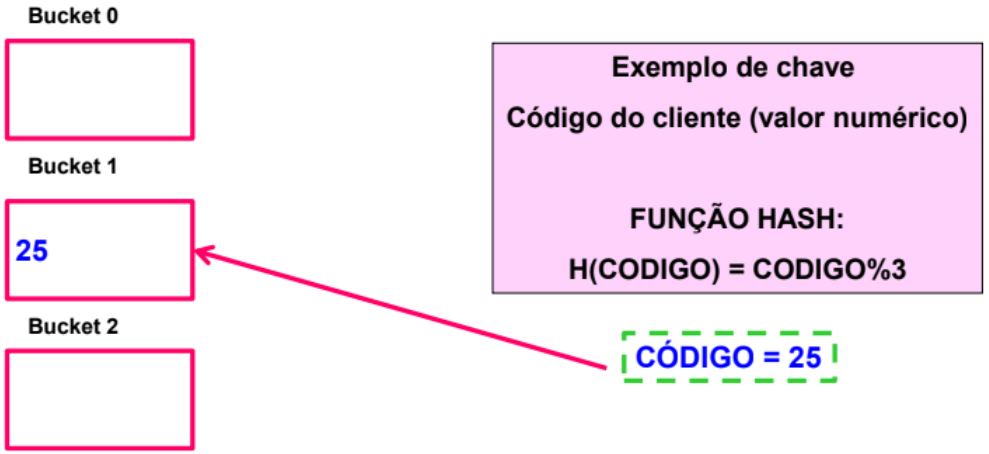
- Existem vários registros por bucket. Logo é possível encontrar h(Ki)=h(Kj).
  - Acessamos o bucket e acessamos o registro – Mesma lógica do custo de árvore.
- Consulta, Inserção, Exclusão
  - Muito simples!
    - Calculamos h(Ki), acessamos o bucket, lemos/inserimos/removemos o registro
- Hash pode ser usado de duas formas
  - Organização de arquivo de hash: registros ficam em buckets, função é calculada. Não há índices.
  - Organização de índice de hash: índices ficam em buckets, possuem ponteiros diretos para o registro.

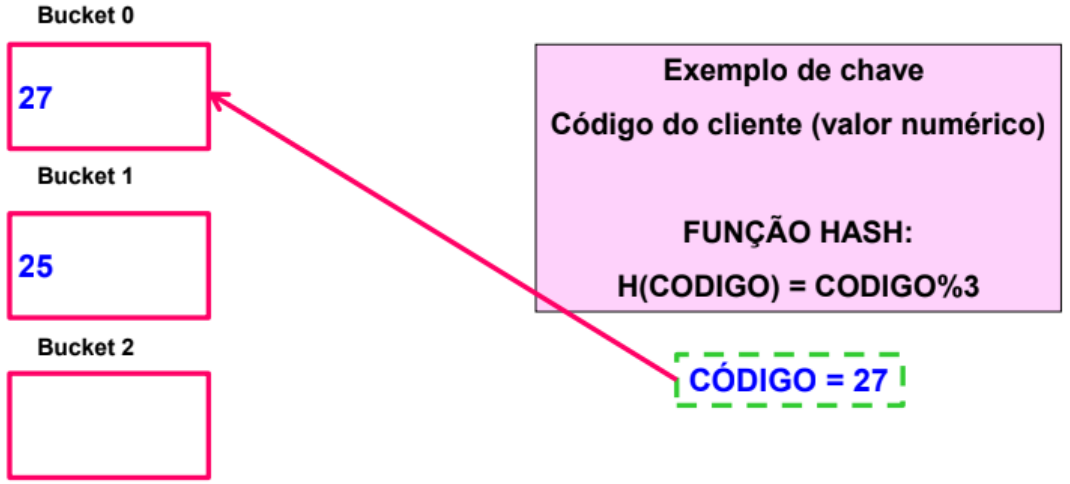
ORGANIZAÇÃO DE ARQUIVOS EM HASH ESTÁTICO

- Pior função hash: mapeia todas as chaves para o mesmo bucket.
  - Torna-se uma leitura sequencial.
- Como não sabemos quantas chaves terão, é desejável alcançar dois objetivos:
  - Distribuição uniforme: que os buckets possuam a mesma quantidade de chaves de busca
  - Distribuição aleatória: não terá complexidade de armazenamento sequencial, ordenados, etc..


Bucket 0
Bucket 1
Bucket 2
Exemplo de chave
Código do cliente (valor numérico)
FUNÇÃO HASH:
H(CODIGO) = CODIGO%3
CÓDIGO = 25


Bucket 0
Bucket 1
25
Bucket 2
Exemplo de chave
Código do cliente (valor numérico)
FUNÇÃO HASH:
H(CODIGO) = CODIGO%3
CÓDIGO = 27


Bucket 0
Bucket 1
25
Bucket 2
Exemplo de chave
Código do cliente (valor numérico)
FUNÇÃO HASH:
H(CODIGO) = CODIGO%3
CÓDIGO = 25


Bucket 0
27
Bucket 1
25
Bucket 2
Exemplo de chave
Código do cliente (valor numérico)
FUNÇÃO HASH:
H(CODIGO) = CODIGO%3
CÓDIGO = 27

FUNÇÃO HASH

- Vamos definir função hash para mapear o arquivo de contas dos clientes por cidade, sendo Cidade a chave de busca.
  - Não é possível aplicar a mesma estratégia de %
  - Quantos buckets? Alguma sugestão?

Cidade	Nome	Endereço
Avaí	Júlia	Rua dos Anjos, 123
Bauru	Ana	Av. Antonio, 865
Brasília	Bruno	Av. Antonio, 865
Brasília	Maria	Rua Estreita, 89
Petrópolis	Carlos	Alameda das Rosas, 634
Petrópolis	Maria	Rua São Paulo, 432
Rondonópolis	Luiza	Avenida Dom Pedro, 800
Santos	Bruno	Rua Aparecida, 7600
Santos	Maria	Rua Santa Rita, 632

FUNÇÃO HASH

- Vamos definir função hash para mapear o arquivo de contas dos clientes por cidade
- Quantos buckets? Alguma sugestão?
  - 26 buckets (um para cada letra do alfabeto): será uniforme?

NÃO!!!

- Funções hash típicas
  - executam cálculos sobre a representação binária interna à máquina de caracteres da chave de procura
  - Exemplo:
    - 1) calcula a soma das representações binárias dos caracteres de uma chave
    - 2) retorna o resto da soma dividido pelo número de buckets

1 + 22 + 1 + 9 =
33 % 10 = 3

Cidade	Nome	Endereço
Avaí	Júlia	Rua dos Anjos, 123
Bauru	Ana	Av. Antonio, 865
Brasília	Bruno	Av. Antonio, 865
Brasília	Maria	Rua Estreita, 89
Petrópolis	Carlos	Alameda das Rosas, 634
Petrópolis	Maria	Rua São Paulo, 432
Rondonópolis	Luiza	Avenida Dom Pedro, 800
Santos	Bruno	Rua Aparecida, 7600
Santos	Maria	Rua Santa Rita, 632

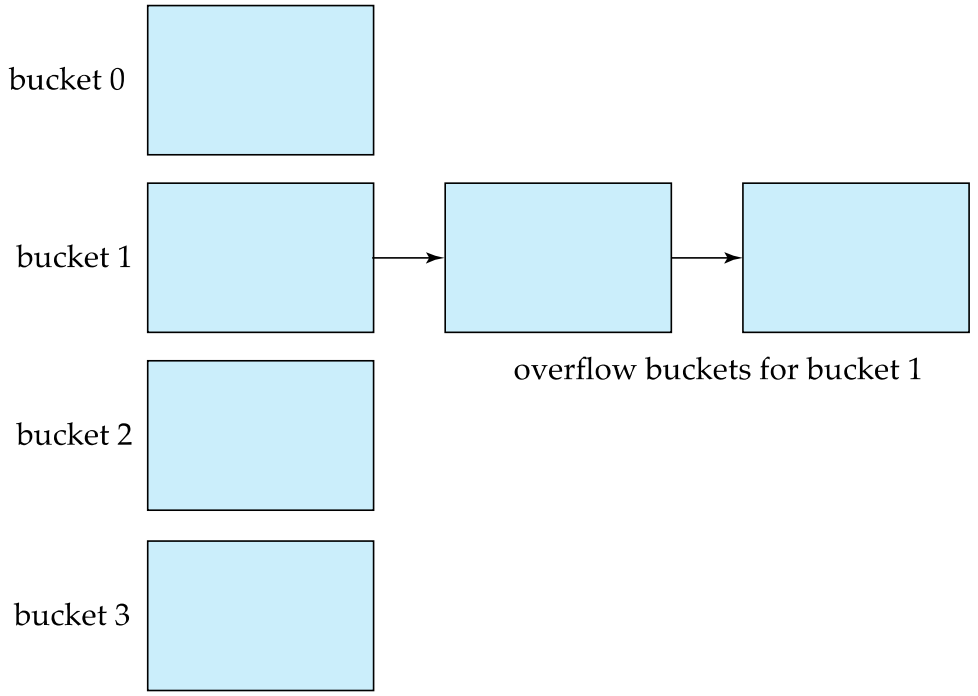
O projeto da função hash é crucial!

ESTOURO DE BUCKETS

- Como nem sempre a volumetria de um BD é conhecida, é possível que a quantidade estimada de buckets não seja suficiente para quantidade de registros
  - Estouro de bucket
- Buckets insuficientes : número de buckets (nB) deve ser > nr / fr
  - nr = número total de registros armazenados
  - fr = número de registros que cabem no bucket
- Distorção: mais registros para alguns buckets
  - registros múltiplos podem ter a mesma chave de procura
  - função hash pode resultar em distribuição não uniforme de chaves de procura
- Para reduzir probabilidade de overflow:
  - escolha do número de (nr / fr)*(1+d)
  - d = fator de fudge – tipicamente ao redor 0,2
  - aproximadamente 20% do espaço dos buckets será perdido

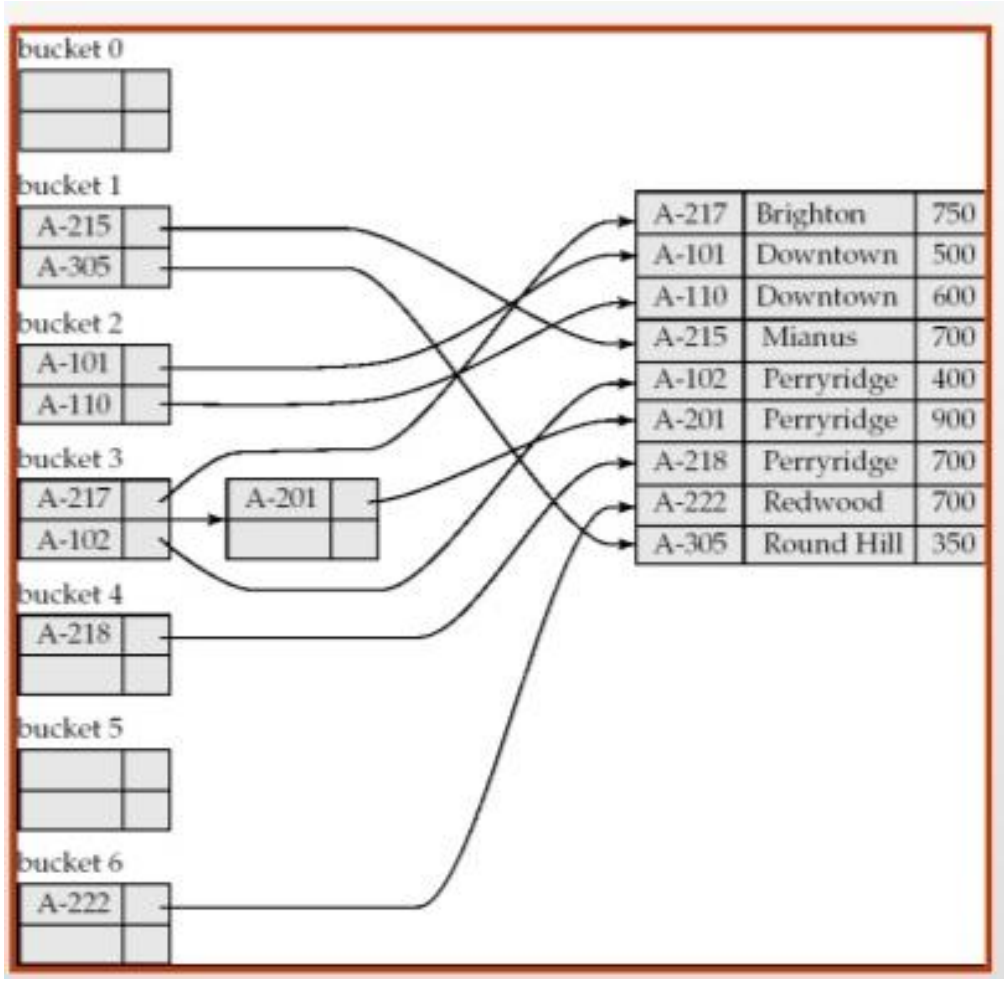
ESTOURO DE BUCKET

- Mesmo com cálculo anterior, overflow ainda pode acontecer.
  - Solução: bucket de overflow
  - se bucket b está cheio ao inserir um registro este é armazenado no bucket de overflow
  - se bucket de overflow cheio -> novo bucket de overflow
  - buckets de overflow de um bucket são encadeados em uma lista ligada
  - manipulação desse tipo de lista é denominado encadeamento de overflow



ÍNDICE DE HASH

Função hash = Soma dos dígitos da conta % 7



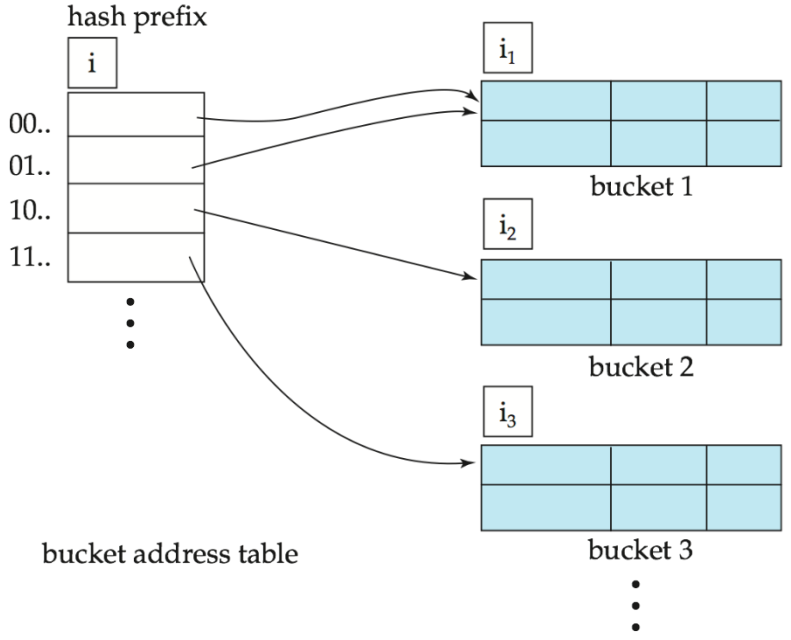
centro universitário FEI

PROBLEMAS DE HASH ESTÁTICO

- Problema do hashing estático : definição do número de buckets
- Opções:
  - escolher função hash com base no tamanho atual do arquivo
    - degradação de desempenho à medida que BD cresce
  - escolher função hash com base no tamanho previsto do arquivo → desperdício de espaço
  - reorganizar periodicamente estrutura de hash, escolher função hash nova e gerar novas atribuições de bucket: operação demorada (proibição de acesso durante a operação)
- Técnicas de hashing dinâmico : permite modificar função hash dinamicamente para acomodar crescimento ou diminuição do BD.
  - Uma das formas: hashing expansível (ou extensível)

HASH DINÂMICO - EXTENSÍVEL

- Trata mudança no tamanho do BD por meio de divisão e fusão de buckets
  - eficiência espacial mantida
  - reorganização realizada apenas em um bucket por vez → overhead de desempenho aceitável e baixo
- Como funciona:
  - escolhe função hash h com propriedades desejadas de uniformidade e aleatoriedade
  - função gera valores dentro de grande faixa – inteiros binários de b bits (valor típico para b = 32) – no máximo 2^32 – 4 milhões de buckets
  - O que se faz: criação de buckets por demanda. Não usa os b bits, mas i bits, sendo 0<= i <=32
  - valor de i cresce ou diminui de acordo com o tamanho do BD

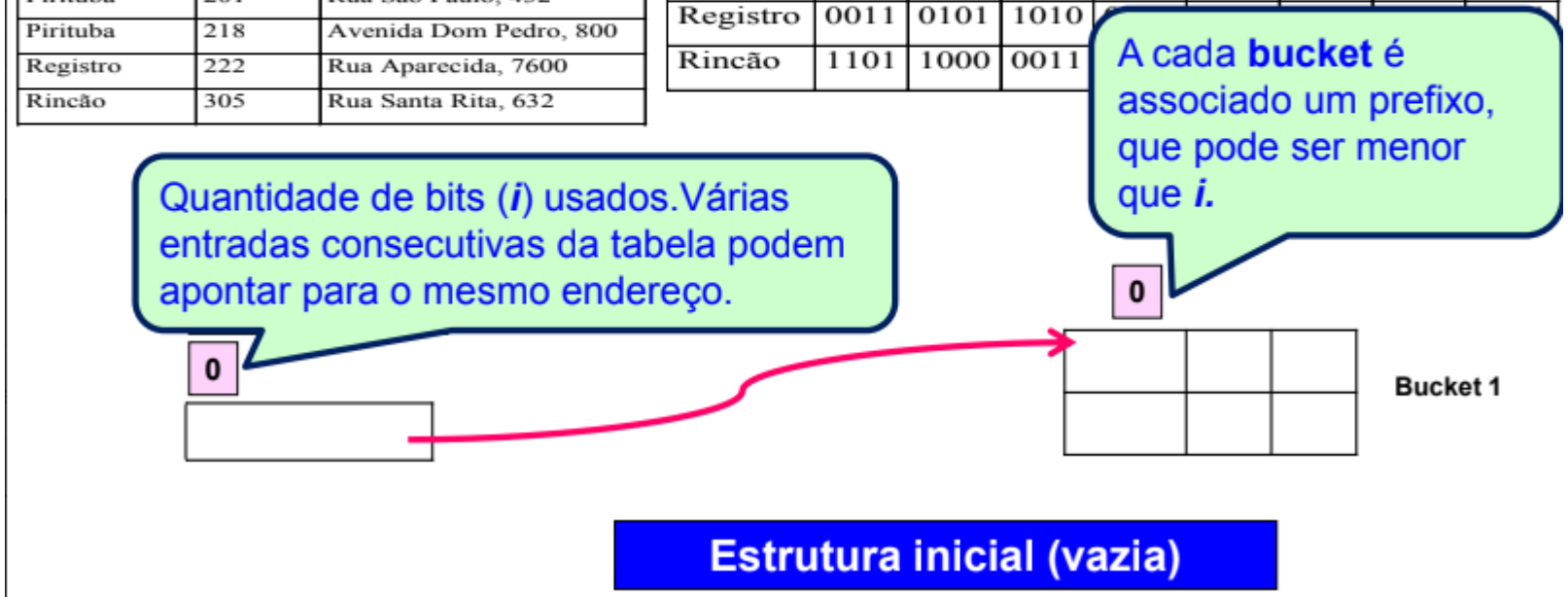


PRATICANDO

Assumiremos que um bucket pode ter no max. 2 registros

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010					
Rincão	1101	1000	0011					



PRATICANDO

Assumiremos que um bucket pode ter no max. 2 registros

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001

Não olhamos prefixo, apenas inserimos

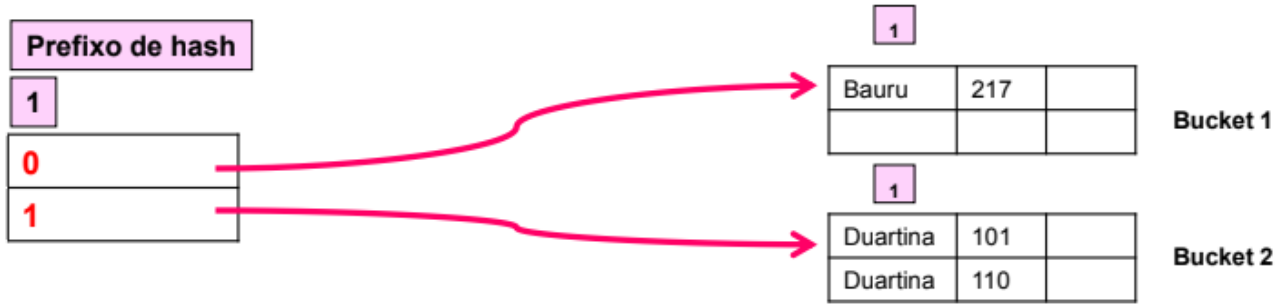
Encheu!

PRATICANDO

- Bucket cheio, vamos aumentar o número de bits 0->1 // 2^1 = 2 buckets
- Começamos a olhar para prefixo e separamos quem tem 0 fica no bucket 0 e quem tem 1 no endereço do prefixo 1

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001

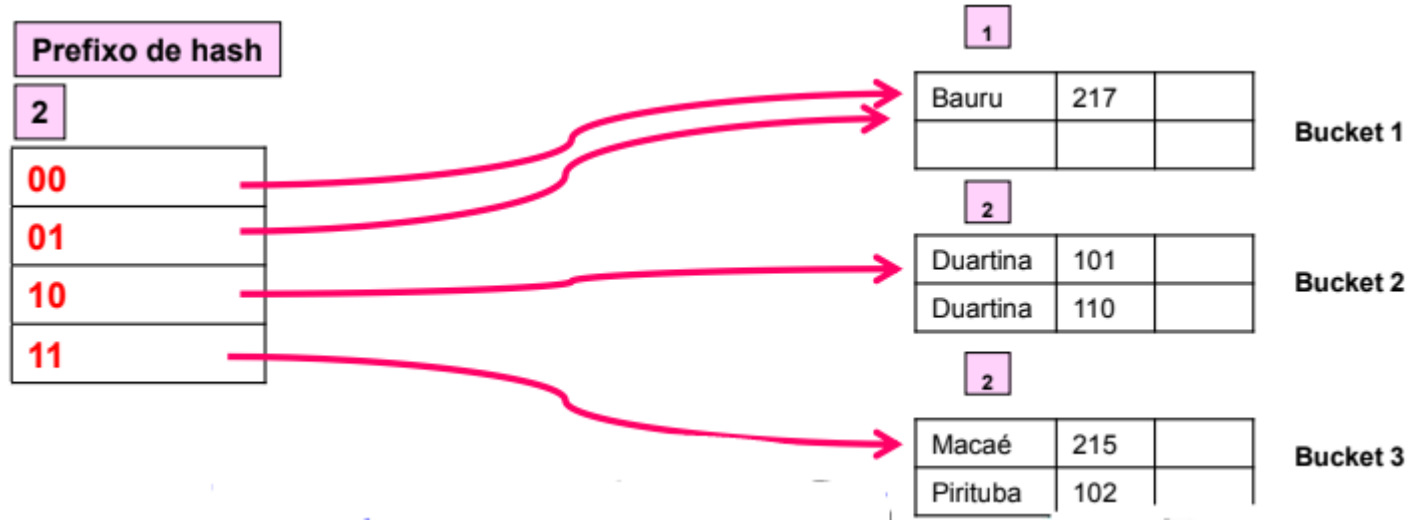


PRATICANDO

- Bucket cheio, vamos aumentar o número de bits 1->2 // 2^3 = 4 buckets
- Começamos a olhar para prefixo e separamos de acordo com o prefixo.

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001

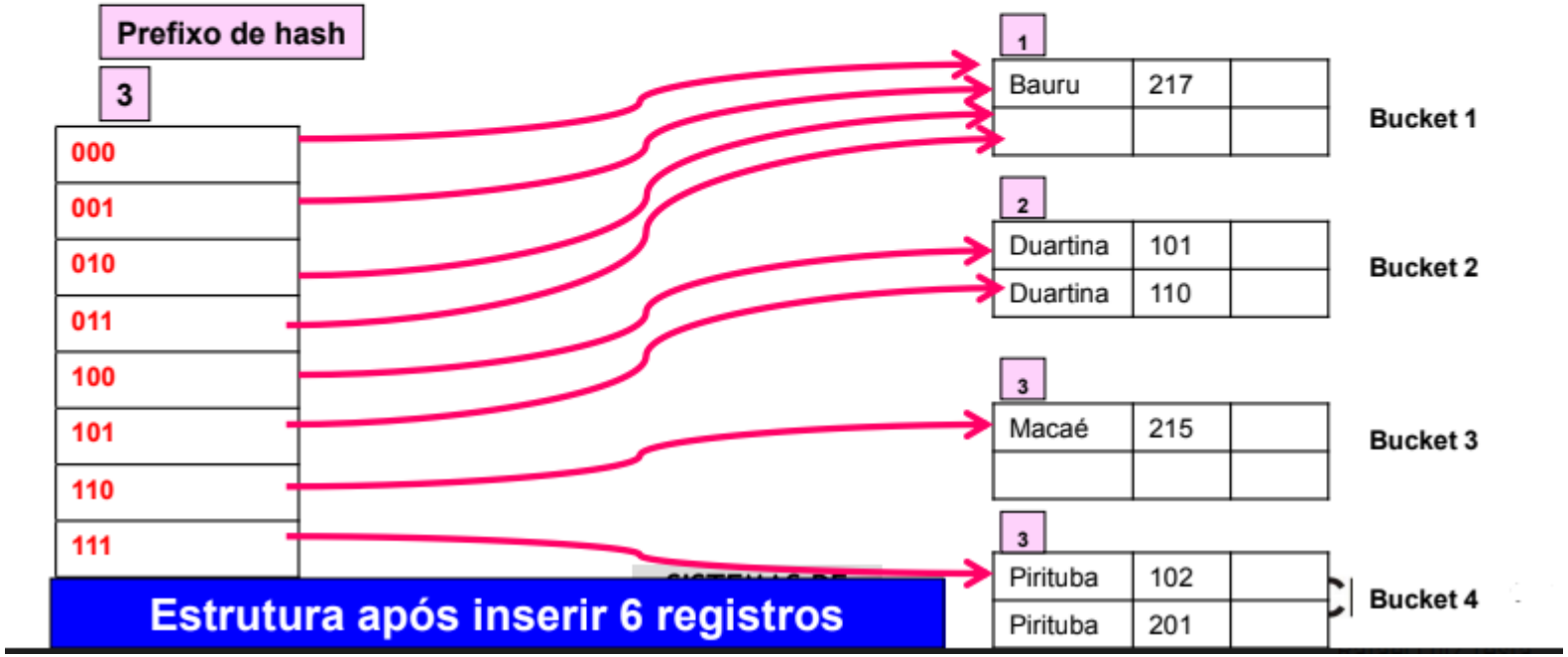


PRATICANDO

- Bucket cheio, vamos aumentar o número de bits 1->2 // 2^3 = 4 buckets
- Começamos a olhar para prefixo e separamos de acordo com o prefixo.

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001

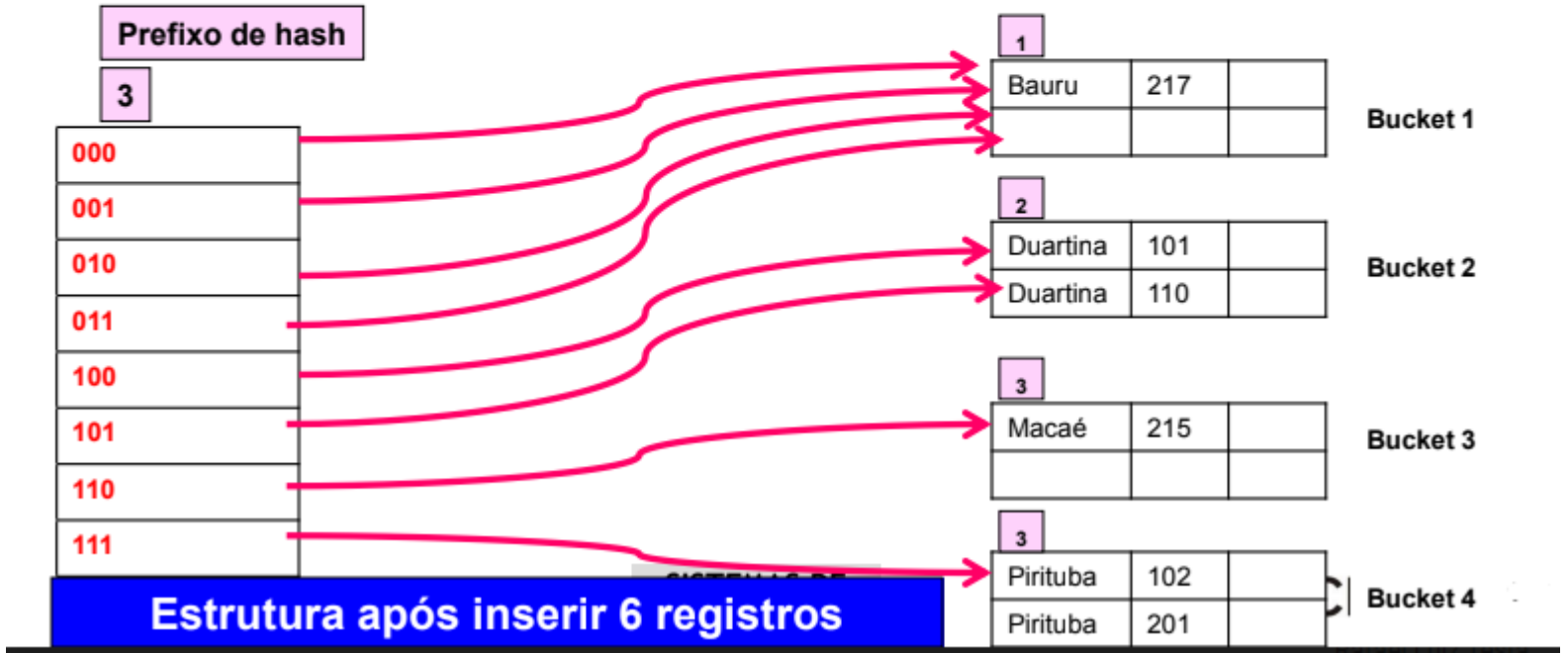


PRATICANDO

- Bucket cheio, vamos aumentar o número de bits 1->2 // 2^3 = 4 buckets
- Começamos a olhar para prefixo e separamos de acordo com o prefixo.

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Piritaba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0011	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001

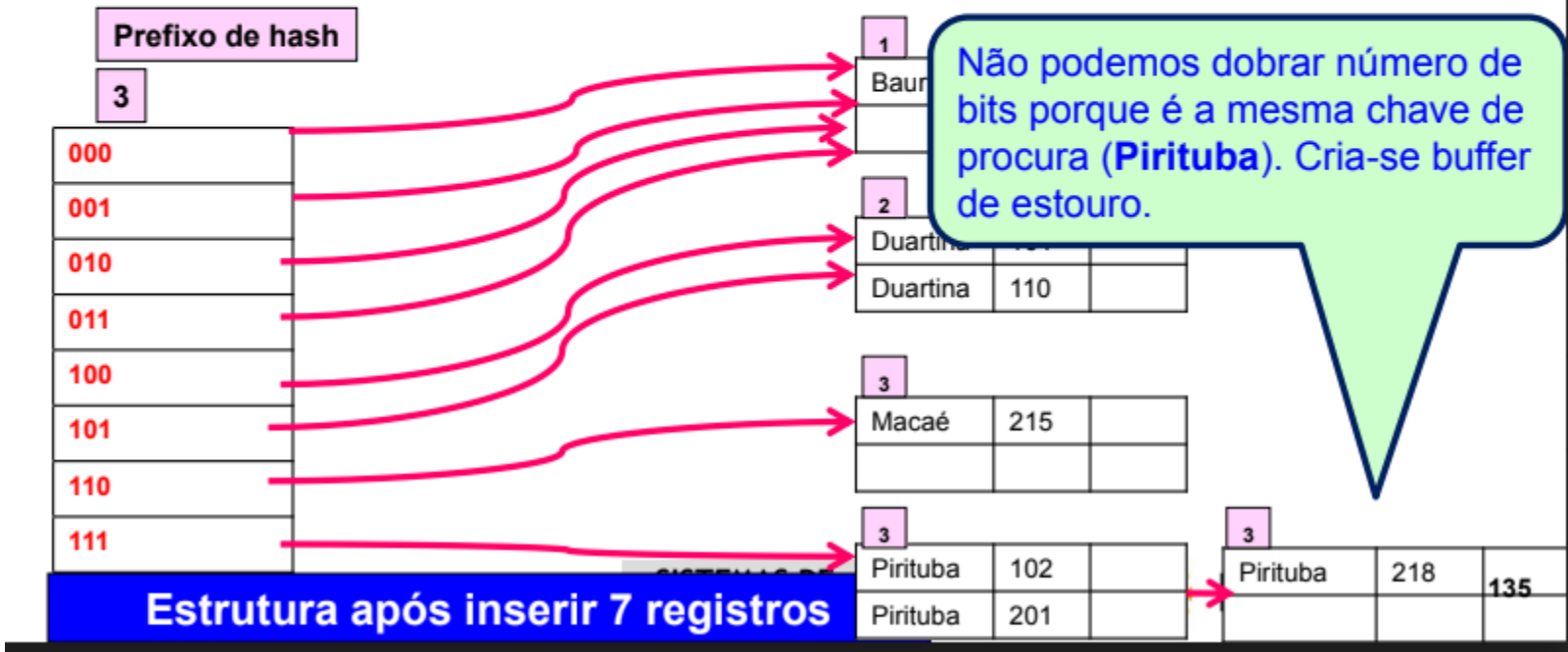


PRATICANDO

- Bucket cheio, vamos aumentar o número de bits 1->2 // 2^3 = 4 buckets
- Começamos a olhar para prefixo e separamos de acordo com o prefixo.

Cidade	Conta	Endereço
Bauru	217	Rua dos Anjos, 123
Duartina	101	Av. Antonio, 865
Duartina	110	Av. Antonio, 865
Macaé	215	Rua Estreita, 89
Pirituba	102	Alameda das Rosas, 634
Pirituba	201	Rua São Paulo, 432
Pirituba	218	Avenida Dom Pedro, 800
Registro	222	Rua Aparecida, 7600
Rincão	305	Rua Santa Rita, 632

Cidade	h(cidade)							
Bauru	0010	1101	1111	1011	0010	1100	0011	0000
Duartina	1010	0011	1010	0000	1100	0110	1001	1111
Macaé	1100	0111	1110	1101	1011	1111	0001	1010
Pirituba	1111	0001	0010	0100	1001	0011	0110	1101
Registro	0011	0101	1010	0110	1100	1001	1110	1011
Rincão	1101	1000	0011	1111	1001	1100	0000	0001



centro universitário FEI

REMOÇÃO

- Remoção de um registro com chave de procura K1
  - mesmo procedimento da procura, parando no bucket j
  - remove chave de procura de j e o registro do arquivo
  - remover j se ficar vazio
  - se buckets forem fundidos -> tamanho da tabela de endereço de bucket pode ser cortado pela metade

centro universitário FEI

ÍNDICE MAPA DE BITS (BITMAP)

centro universitário FEI

MAPA DE BITS

- Array de bits
- Índice de mapa de bits sobre o atributo A da tabela T consiste em um mapa de bits para cada valor que A pode assumir
  - Cada mapa possui tantos bits quanto a quantidade de registros.
  - Bom quando existem diferentes chaves sendo utilizada na consulta

centro universitário FEI

MAPA DE BITS

- Array de bits
- Índice de mapa de bits sobre o atributo A da tabela T consiste em um mapa de bits para cada valor que A pode assumir
  - Genero: (m/f)
  - Intervalo de reda: (0-500, 1000-3000, etc…)
  - Cada mapa possui tantos bits quanto a quantidade de registros.
  - Bom quando existem diferentes chaves sendo utilizada na consulta

Assume valor 1 se o atributo possuir o valor e 0 caso contrário

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

- Útil para consultas com múltiplos atributos (AND ou OR)
  - Cada operação retorna um mapa de bits, ela aplica a intersecção combinando os resultados.
- Todos as mulheres com renda nível L1:

```
SELECT * from table

WHERE gender=F AND income_level = L1
```

- 01101 AND 10100 = 00100

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

centro universitário FEI

VANTAGENS E DESVANTAGENS

- Hashing extensível x Hashing estático
- Vantagens:
  - desempenho não se degrada com crescimento do arquivo
  - overhead mínimo
  - tabela de endereços contém só um endereço para cada valor hash
  - nenhum bucket reservado para crescimento futuro: economia de espaço
- Desvantagens:
  - procura envolve nível de acesso indireto adicional
  - maior complexidade na implementação

centro universitário FEI

VANTAGENS E DESVANTAGENS

- Hashing x Indexação Ordenada
- Vantagens:
  - Não necessita de ordenação
  - Não precisa de espaço reservado para índice
  - Não degrada com aumento de arquivos
- Desvantagens:
  - maior complexidade na implementação

centro universitário FEI

OPÇÕES

Organização de arquivo	Índice
Sequencial	Sequencial
Árvores	Árvores
Hashing	Hashing

VANTAGENS E DESVANTAGENS

- Maioria dos sistemas de BD usa somente algumas ou uma única forma de indexação ordenada ou hashing
  - Aspectos para que o desenvolvedor de BD escolha:
    - custo da reorganização periódica do índice ou da organização hash
    - frequência de inserções e remoções
    - se compensa otimizar tempo médio de acesso às custas do aumento do tempo de acesso no pior caso
    - tipos de consultas com maior probabilidade de ocorrência

VANTAGENS E DESVANTAGENS

- select A1, A2, ..., An
- from r
- where Ai = c1

- select A1, A2, ..., An
- from r
- where Ai <= c2 and Ai >= c1

Qual índice usar em cada caso?

VANTAGENS E DESVANTAGENS

- select A1, A2, ..., An
- from r
- where Ai = c1

Hashing preferível (tempo constante)

Busca ordenada:
tempo=proporcional ao
número de valores de c no atributo Ai

centro universitário FEI

VANTAGENS E DESVANTAGENS

- select A1, A2, ..., An
- from r
- where Ai <= c2 and Ai >= c1

Como seria a procura para esta consulta usando índice hash?

Acha bucket com valor c1. E fica difícil seguir ponteiros até bucket para achar c2.
Então, qual o melhor?
Ordenado!

Como seria a procura para esta consulta usando índice ordenado?

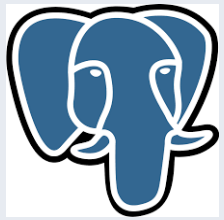
Primeiro busca c1. Quando encontrar bucket com valor = c1, segue cadeia de ponteiros até achar c2.

centro universitário FEI

ÍNDICES NA PRÁTICA

- Índices
  - Ordenados
    - Denso
    - Esparso
  - Arvores
    - B
    - B+
  - Hash
    - Estático
    - Dinâmico
  - Mapa de bits

BRIN*	Bitmap
GIN	Inverted Index = Esparso
Rtree	Spatial data
x**	Somente em partições, clusters
Secondary Indexing***	

							
Hash	x	x**	x	x	x		x
Bitmap	x*	x					
Btree	x	x	x	x			x***
Function-based		x					
BRIN	x						
GIS	x						
Rtree			x				
Geospatial				x			
Compound Index				x			

centro universitário FEI

ÍNDICES - SQL

- `CREATE INDEX ON idx_aluno Alunos (codAluno)`
- `CREATE INDEX ON idx_aluno Alunos using hash (codAluno)`

centro universitário FEI

PRÁTICA

```
create table cc6240.example_index as select s,
md5(random()::text)
from generate_Series(1,1000) s;


select * from cc6240.example_index
where s=10
```

BIBLIOGRAFIA

- ABRAHAM SILBERSCHATZ, HENRY F. KORTH, S. SUDARSHAN. Sistema de Banco de Dados. 6. Campus. 0. ISBN 9788535245356.

- ELMASRI, RAMEZ, SHAMKANT B. NAVATHE. Sistemas de banco de dados. Vol. 6. São Paulo: Pearson Addison Wesley, 2011.

- DATE, CHRISTHOPER J. Introdução a Sistemas de Bancos de Dados, 5ª. Edição. Campus, Rio de Janeiro (2004).

OBRIGADO E ATÉ A PRÓXIMA AULA!