# GBC053–Gerenciamento de Banco de Dados
# Organização de Arquivos

## Parte 2 – Abordagem Comparativa

Ilmério Reis da Silva
ilmerio@facom.ufu.br
www.facom.ufu.br/~ilmerio/gbd
UFU/FACOM/BCC

# *Organização de Arquivos - OBJETIVO*

***Entender como os dados podem ser organizados no espaço em disco para alcançar bom desempenho e segurança nas operações com SGBDs***

# *Comparação de Desempenho de Arquivos*

- ***Tipos de arquivos e índices***
- ***Operações***
- ***Modelo de Custo***
- ***Custos de cada operação/alternativa***
- ***Tabela Comparativa***

# *Tipos de Organização de Arquivos*

- ***Arquivo não ordenado (heap file)***
- ***Arquivo ordenado (sorted file)***
- ***Arquivo indexado por Hash***
- ***Arquivo indexado por Árvore B+***
- ***Heap File + Índice não agrupado por Árvore B+ Alternativa 2 (k+rid)***
- ***Heap File + Índice baseado em Hash Alternativa 2***

# *Comparação de Desempenho - Operações*

***Operações em Arquivos e Índices***

- ***Scan (varredura) : ler todos os registros de um arquivo***
  - Carregar páginas do arquivo no *Buffer Pool*
  - Buscar registros em cada página
- ***Busca Igual (com predicado de igualdade k = valor)***
  - Carregar páginas com os registros selecionados
  - Buscar registros em cada página
  - Se o predicado não for pela chave, então realizar varredura
- ***Busca por faixa de valores (intervalo i.e. (x < k < y))***
  - Carregar páginas com os registros selecionados
  - Buscar registros em cada página
  - Se o predicado não for pela chave, então realizar varredura.

# *Desempenho de Arquivos – Operações cont...*

***Operações em Arquivos***

- ***Inserção :***
  - *Identificar página na qual o registro deve ser inserido*
  - Carregar a página no *Buffer Pool*
  - Incluir registro na página
  - Escrever página modificada no disco

- ***Remoção :***
  - Identificar a página contendo o registro
  - Carregar a página no *Buffer Pool*
  - Modificar a página
  - Escrever a página modificada no disco

# *Desempenho de Arquivos – Modelo de Custo*

***Modelo de Custo***

- ***B = número de Páginas***
- ***R = número de registros por página***
- ***D = tempo médio para ler ou escrever uma página no disco***
- ***C = tempo médio para processar um registro***
- ***H = tempo para aplicação da função Hash***
- ***C e H da ordem de nanosegundos e D microsegundos, logo o tempo gasto com IO é o custo dominante***
- ***Essas considerações são simplistas mas suficientes para mostrar a intuição das estruturas que serão estudadas em detalhe.***

# *Desempenho de Arquivos - Considerações*

- ***Um registro resultante da busca com igualdade***
- ***Arquivo Ordenado com compactação após remoção***
- ***Arquivo Hash com 80% de ocupação buckets, logo, tamanho total do arquivo = 1,25B páginas***
- ***Árvore Agrupada (Alternativa 1) com 67% ocupação, logo, folhas na árvore ocupam 1,5B páginas***
- ***Índices Árvore B+ não agrupada e Hash, usando Alternativa 2 com entradas de tamanho igual a 10% do registro e 67% de ocupação, logo:***
  - *média de 6,7R entradas por página/bucket*
  - *Hash com 0,15B páginas*
  - *Idem para folhas da árvore.*

# *Desempenho de Arquivos - Heap*

***Considerações sobre o***
***Arquivo não Ordenado (Heap)***

- ***Inserção por ordem de chegada, sempre no final do arquivo***
- ***Sem compactação em remoção***

# *Desempenho de Arquivos – Heap/Varredura*

***Varredura(Scan)***

Ler todas as páginas (BD)

- Processar R registros por página (BRC)
- Tempo = B (D + RC)
- Custo IOs = B

# *Desempenho de Arquivos – Heap/Busca =*

- ***BuscaIgual (predicado k = valor)***
  - Ler metade do arquivo(em média) para encontrar a página do registro (0,5BD)
  - Processar todos registros de cada página lida à procura do registro especificado (0,5BRC)
  - Tempo = 0,5B(D + RC)
  - Custo IO = 0,5B

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap*

- ***BuscaFaixa(intervalo )***
  - Ler todas as páginas do arquivo em busca de registros do intervalo (BD)
  - Processar todos os registros de cada página (BRC)
  - Tempo = B(D + RC)
  - Custo IO = B

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap*

- ***Inserção***
  - Ler a última página do arquivo (D)
  - Inserir o registro na página(C)
  - Rescrever a página modificada no disco(D)
  - Tempo = D + C + D = 2D + C
  - Custo IO = 2

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap*

- ***Remoção***
  - Localizar o registro a ser removido(BuscaIgual)
  - Remover o registro da página (C)
  - Rescrever a página modificada no disco (D)
  - Tempo = BuscaIgual + C + D
  - Custo IO = 0,5B + 1
  - Obs: lembrando que o arquivo não será comprimido após remoção.

# *Comparação de Desempenho de Arquivos*

***Considerações sobre o***
***Arquivo Ordenado***

- Inserção em ordem de uma chave de busca
- Busca binária no arquivo para localizar um registro
- Reorganização das páginas após inserção
- Compactação do arquivo após remoção

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***Varredura(Scan)***
  - Ler todas as páginas (BD)
  - Processar R registros por página (BRC)
  - Tempo = B (D + RC)
  - Custo IOs = B

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***BuscaIgual (predicado k = valor)***
  - Localizar página por meio de busca binária no arquivo ($D\log_2 B$)
  - Processar 2 registros(extremidades) por página em cada passo da busca ($2C\log_2 B$)
  - Realizar busca binária na página onde se localiza o registro ($C\log_2 R$)
  - Tempo = $(D+2C)\log_2 B + C \log_2 R$
  - Custo IO = $\log_2 B$

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***BuscaFaixa(intervalo )***
  - Localizar a página e registro de início do intervalo (BuscaIgual)
  - Ler todos os registros subsequentes na página (0,5RC)
  - Ler as demais páginas e processar seus registros até encontrar o final do intervalo. Seja #pgm o número adicional de páginas com registros no intervalo (#pgm (D+RC))
  - Tempo = BuscaIgual + #pgm D + RC (#pgm + 0,5)
  - Custo IO = $\log_2 B$ + #pgm

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***Inserção***
  - Localizar página para inserção (BuscaIgual)
  - Seja 2C o custo de inserir e realizar o shift dos registros na página.
  - Ler todas as páginas a partir da posição corrente(metade do arquivo) *(0,5BD)*
  - Realizar *o shift* em todas as páginas a partir da posição corrente *(0,5B 2C)*
  - Reescrever todas as páginas a partir da posição corrente (0,5BD)
  - Tempo = BuscaIgual + B (D + C)
  - Custo IO = $B + \log_2 B$

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***Remoção***
  - Localizar página para remoção (BuscaIgual)
  - Seja 2C o custo de remover e realizar o *unshift* dos registros em uma página
  - Ler as páginas à partir da posição corrente(metade do arquivo) *(0,5 BD)*
  - Realizar a remoção e *unshift* dos registros (0,5B 2C)
  - Reescrever todas as páginas a partir da posição corrente (0,5BD)
  - Tempo = BuscaIgual + B (D + C)
  - Custo IO = $B + \log_2 B$

# *Comparação de Desempenho de Arquivos*

## *Arquivo Ordenado*

- ***Observações:***
  - A reorganização do arquivo na inserção e remoção torna essas operações muito custosas
  - Uma alternativa é adiar a reorganização para ser feita periodicamente, prejudicando as buscas
  - Mas a principal solução deste problema é o arquivo indexado que mantem o custo de busca na mesma ordem e diminui consideravelmente o custo de inserção e remoção

# *Comparação de Desempenho de Arquivos*

***Considerações sobre o***
***Arquivo Indexado baseado em Hash***

- Hash com entradas do tipo Alternativa 1
- Hash sem *overflow*
- Páginas são ocupadas em 80%, pois espaço livre é deixado nas páginas para evitar *overflow* no bucket em caso de novas inserções
- Número de páginas no hash será:

$$B/0,80 = 1,25B$$

# *Comparação de Desempenho de Arquivos*

***Arquivo Hash***

- ***Varredura(Scan)***

  - Ler todas as páginas (1,25 BD)
  - Processar 0,8R registros em cada página (1,25B 0,8RC)
  - Tempo = 1,25B(D + 0,8RC)
  - Custo IO = 1,25B

# *Comparação de Desempenho de Arquivos*

## *Arquivo Hash*

- ***BuscaIgual (predicado k = valor)***
  - Calcular a função hash (H)
  - Ler o bucket (D)
  - Assumindo varredura média de metade do bucket para localizar o registro (0,4RC)
  - Tempo = H + D + 0,4RC
  - Custo IO = 1

# *Comparação de Desempenho de Arquivos*

## *Arquivo Hash*

- ***BuscaFaixa(intervalo)***
  - Ler todas as páginas do *hash* em busca de registros do intervalo (1,25BD)
  - Processar os registros de cada página (1,25B 0,8RC)
  - Tempo = 1,25B(D+0,8RC)
  - Custo IO = 1,25B

# *Comparação de Desempenho de Arquivos*

## *Arquivo Hash*

- ***Inserção***
  - Calcular a função hash (H)
  - Ler o bucket (sem overflow) (D)
  - Inserir registro no final do bucket (C)
  - Reescrever o bucket (D)
  - Tempo = H + C + 2D
  - Custo IO = 2

# *Comparação de Desempenho de Arquivos*

## *Arquivo Hash*

- ***Remoção***
  - Localizar o registro (BuscaIgual)
  - Remover o registro, reorganizando o bucket (0,4RC)
  - Escrever o bucket modificado (D)
  - Tempo = BuscaIgual + 0,4RC + D
  - Custo IO = 2

# *Comparação de Desempenho de Arquivos*

***Considerações sobre o***
***Arquivo Indexado baseado em Arvore B+ Agrupada***

- Entrada nas folhas do tipo Alternativa 1
- Folhas são ocupadas em 67%, pois espaço livre é deixado para novas inserções
- Número de folhas = B / 0,67 = 1,5B
- Número de registros por folha = 0,67R
- Seja F o número médio de filhos de cada nó interno, então, a altura da ávore $h = log_F 1,5B$

# *Comparação de Desempenho de Arquivos*

***Arvore B+ Agrupada***

- ***Varredura(Scan)***
  - Ler todas as folhas da árovore (1,5BD)
  - Processar os registros de cada folha (1,5B 0,67RC)
  - Tempo = B(1,5D + RC)
  - Custo IO = 1,5B

# *Comparação de Desempenho de Arquivos*

***Arvore B+ Agrupada***

- ***BuscaIgual (predicado k = valor)***
  - Localizar folha por meio de busca em profundidade na árvore (D$log_F 1,5B$)
  - Em cada nível, processar uma média de $log_2(F-1)$ entradas para localizar ponteiro: $((log_F 1,5B)(C log_2(F-1)))$
  - Localizar registro na folha por meio de busca binária (C$log_2 0,67R$)
  - Tempo = $log_F 1,5B$ (D + C$log_2(F-1)$) + C $\log_2 0,67\ R$
  - Custo IO = $log_F 1,5$B

# *Comparação de Desempenho de Arquivos*

***Arvore B+ Agrupada***

- ***BuscaFaixa(intervalo)***
  - Localizar início do intervalo (BuscaIgual)
  - Ler folhas subsequentes até encontrar o final do intervalo (1,5 #pgm D)
  - Processa entradas de cada folha (1,5 #pgm 0,67RC)
  - Tempo = BuscaIgual + 1.5 #pgm D + R #pgm C

    **= *BuscaIgual + 1,5 #pgm (D + 0,67RC)***
  - Custo IO = BuscaIgual + 1,5 #pgm

# *Comparação de Desempenho de Arquivos*

## *Arvore B+ Agrupada*

- ***Inserção***
  - Localizar a folha onde será inserido o novo registro (BuscaIgual)
  - Inserir nova entrada na folha, fazendo um *shift de* suas entradas. Seja 2C o custo do *shift* (2C)
  - Reescrever a folha (D)
  - Tempo = BuscaIgual + 2C + D
  - Custo IO = 1 + BuscaIgual

# *Comparação de Desempenho de Arquivos*

***Arvore B+ Agrupada***

- ***Remoção***
  - Localizar a folha onde será removido o registro (BuscaIgual)
  - Remover a entrada na página, fazendo um *unshift de* suas entradas. Seja 2C o custo do un*shift (2C)*
  - Reescreve folha (D)
  - Tempo = BuscaIgual + 2C + D
  - Custo IO = 1 + BuscaIgual

# *Comparação de Desempenho de Arquivos*

***Considerações sobre o***
***Arquivo Heap com Índice baseado Arvore B+ não agrupada***

- ***A entrada nas folhas da árvore ocupa 10% do registro***
- ***67% de ocupação das folhas***
- ***Portanto serão 6,7R entradas em cada folha***
- ***O número de folhas da ávore será***

$$N = 0{,}1B/0{,}67 = 0{,}15B$$

# *Comparação de Desempenho de Arquivos*

***Arquivo Heap + Arvore B+ não agrupada***

- ***Varredura(Scan)***
  - *Varrer o índice não é viável, pois seria necessário varrer as folhas e mais um IO no arquivo de dados para cada registro i.e.,*
    - ✓ *Custo IO > BR*
  - *Então, varrer o heap*
    - ✓ *Tempo = B (D + RC)*
    - ✓ Custo IOs = B

# *Comparação de Desempenho de Arquivos*

***Arquivo Heap + Arvore B+ não agrupada***

- ***BuscaIgual (predicado k = valor)***
  - Localizar a folha por meio de busca em profundidade na árvore ($Dlog_F 0.15B$)
  - Processar uma média de $log_2(F-1)$ entradas por nível ($log_F 0.15B$)($Clog_2(F-1)$)
  - Localizar a entrada na folha por meio de busca binária ($Clog_2 6,7R$)
  - Ler a página de dados (D) e processar o registro (C)
  - Tempo = $log_F 0,15B$(D + $Clog_2(F-1)$) + $Clog_2 6,7R+D+C$
  - Custo IO = 1 + $log_F 0,15$B

# *Comparação de Desempenho de Arquivos*

***Arquivo Heap + Arvore B+ não agrupada***

- ***BuscaFaixa(intervalo)***
  - Localizar início do intervalo (BuscaIgual)
  - Ler folhas subsequentes até encontrar o final do intervalo (*0,15D #pgm)*
  - Processar cada entrada (C R #pgm)
  - Para cada entrada na folha:
    - ✓ Ler página no arquivo de dados (D R #pgm)
    - ✓ Processar cada registro (C R #pgm)
  - Tempo = BuscaIgual + 0,15D#pgm + R#pgm (D + 2C)
  - Custo IO = BuscaIgual + #pgm (0,15 + R)

# *Comparação de Desempenho de Arquivos*

***Arquivo Heap + Arvore B+ não agrupada***

- ***Inserção***
  - Localizar a folha onde será inserida a nova entrada e página de dados onde será inserido o novo registro (BuscaIgual)
  - Inserir nova entrada na folha, fazendo um *shift de* suas entradas (2C)
  - Inserir novo registro na página do heap (C)
  - Reescrever folha e página de dados (2D)
  - Tempo = BuscaIgual + 2D + 3C
  - Custo IO = 2 + BuscaIgual

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap + Arvore B+ não agrupada*

- ***Remoção***
  - Localizar a folha de onde será removida a nova entrada e página de dados de onde será removido o novo registro (BuscaIgual)
  - Remover nova entrada na folha, fazendo um *unshift de* suas entradas (2C)
  - Remover novo registro na página de dados (C)
  - Reescrever folha e página de dados (2D)
  - *Tempo = BuscaIgual + 3C + 2D*
  - *Custo IO = 2 + BuscaIgual*

# *Comparação de Desempenho de Arquivos*

***Considerações sobre***
***Arquivo Heap com Índice baseado Hash***

- ***Hash com entradas do tipo Alternativa 2***
- ***Hash sem overflow***
- ***Páginas são ocupadas em 67%***
- ***Entrada de tamanho 10% do registro***
- ***Número de páginas no hash será:***

$$0{,}1B/0{,}67 = 0{,}15B$$

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap + Hash*

- ***Varredura(Scan)***
  - *Varrer o hash não é viável, pois seria necessário varrer os buckets e mais um IO no arquivo de dados para cada entrada i.e.,*
    - ✓ *Custo IO > BR*
  - *Então, varrer o heap*
    - ✓ *Tempo = B (D + RC)*
    - ✓ *Custo IOs = B*

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap + Hash*

- ***BuscaIgual (predicado k = valor)***
  - Calcular a função hash (H)
  - Ler o bucket (sem overflow) (D)
  - Varrer em média metade do bucket para localizar a entrada (C 3,35R)
  - Ler página no arquivo de dados (D)
  - Processar o registro (C)
  - Tempo = H + D + 3.35RC + D + C

    ***= H + 2D + C(3,35R + 1)***
  - Custo IO = 2

# *Comparação de Desempenho de Arquivos*

## *Arquivo Heap + Hash*

- ***BuscaFaixa(intervalo)***
  - Ler todos os buckets do hash e realizar um IO para cada entrada do intervalo ou fazer uso do heap
  - Vamos comparar o número de IO de duas alternativas
    - ✓Varrer o Hash e ler heap direto: (0,15B + R #pgm)
    - ✓Varrer o Heap: B
  - Logo se #pgm < (0,85B/R) será viável o uso do hash, caso contrário, ou quando não se conhece a estatística, usa-se o heap, i.e.:
    - ✓Tempo = B(D + RC)
    - ✓Custo IO = B

# *Comparação de Desempenho de Arquivos*

***Arquivo Heap + Hash***

- ***Inserção***
  - Calcular a função hash (H)
  - Ler o bucket (sem overflow)(D)
  - Ler página no arquivo de dados (D)
  - Inserir registro no final da página (C)
  - Inserir entrada no final do bucket (C)
  - Reescrever o bucket e a páginia de dados(2D)
  - Tempo = H +2D + 2C + 2D = H + 4D + 2C
  - Custo IO = 4

# *Comparação de Desempenho de Arquivos*

## ***Arquivo Heap + Hash***

- ***Remoção***
  - Calcular a função hash (tempo = H)
  - Ler o bucket (D)
  - Ler página no arquivo de dados(D)
  - Localizar e remover entrada no bucket (3,35RC)
  - Localizar e remover registro na página (2C)
  - Regravar o bucket e a páginia de dados (2D)
  - *Tempo = H +2D + 3,35RC + 2C + 2D*

    ***= H + 4D + C(3,35R + 2)***
  - *Custo IO = 4*

# *Comparação de Desempenho de Arquivos*

| TEMPO | VARREDURA | BUSCAIGUAL | BUSCAFAIXA | INSERÇÃO | REMOÇÃO |
|---|---|---|---|---|---|
| HEAP | B (D + RC) | O,5B (D + RC) | B (D + RC) | 2D + C | BuscaIgual + D + C |
| SORTED | B (D + RC) | (D+2C)$\log_2$B + C $\log_2$R | *BuscaIgual + D#pgm + RC(#pgm + 0,5)* | BuscaIgual + B(D+C) | BuscaIgual + B(D+C) |
| HASH FILE | 1,25B (D + 0,8RC) | H + D + 0,4RC | 1,25B (D + 0,8RC) | C + 2D + H | BuscaIgual + 0,4RC + D |
| BTREE FILE | B (1,5D + RC) | $\log_F$1.5B (D + C$\log_2$(F-1)) + C$\log_2$0,67R | BuscaIgual + 1,5#pgm (D + 0,67RC) | BuscaIgual + D + 2C | BuscaIgual + D + 2C |
| BTREE INDEX | B (D + RC) | $\log_F$0,15B (D + C$\log_2$(F-1)) + C$\log_2$6,7R + D + R | BuscaIgual + 0,15D#pgm + R#pgm(D + 2C) | BuscaIgual + 2D + 3C | BuscaIgual + 2D + 3C |
| HASH INDEX | B (D + RC) | H + 2D + C(3,35R + 1) | B (D + RC) | H + 4D + 2C | H + 4D + C(3,35R+1) |

# *Comparação de Desempenho de Arquivos*

| IO | VARREDURA | BUSCAIGUAL | BUSCAFAIXA | INSERÇÃO | REMOÇÃO |
|---|---|---|---|---|---|
| HEAP | B | 0.5B | B | 2 | 0.5B + 1 |
| SORTED | B | $\log_2 B$ | $log_2 B + \#pgm$ | B + $\log_2$B | $B + log_2 B$ |
| HASH FILE | 1.25B | 1 | 1.25B | 2 | 2 |
| BTREE FILE | 1.5B | $\log_F$1.5B | $\log_F$1.5B + 1.5 #pgm B | 1 + $\log_F$1.5B | 1 + $\log_F$1.5B |
| BTREE INDEX | B | $1 + log_F 0.15$B | $1 + log_F 0.15B$ + #pgm(0.15 + R) | $3 + log_F 0.15$B | $3 + log_F 0.15$B |
| HASH INDEX | B | 2 | B | 4 | 4 |

# *Organização de Arquivos*

***Considerações finais***

- ***Uso em ajuste(tunning) de SBD***
- ***Geram impactos positivos e negativos na carga de trabalho do SGBD***
- ***Analisar frequencia de operações***
- ***Sobrecarga de espaço em disco***
- ***Hash tem melhor desempenho em busca com predicado de igualdade***
- ***Árvore B+ tem melhor desempenho em busca por intervalo***
- ***Algumas consultas podem ser avaliadas usando somente o índice, por exemplo: employee(eid, dno, age, hobby, sal)***

  ***SELECT avg(sal) FROM employee***

***Considerações finais – Alguns exemplos de uso de índices agrupados, usando o esquema***

***employee(eid, dno, age, hobby, sal)***

1. **ganho depende de selectividade da condição**

```
SELECT dno
FROM   employee
WHERE  age > 40
```

2. **refinando a consulta anterior**

```
SELECT   dno, COUNT(*)
FROM     employee
WHERE    age > 10
GROUP BY dnO
```

**Se índice por** *age*, **exige** *sort* **por** *dno*

**Índice por** *dno* **melhora somente se agrupado**

## *Considerações finais – Alguns exemplos de uso de índices agrupados*

**funções de agregação e projeto de índices**

**SELECT** *dno*, $COUNT(*)$
**FROM** *employee*
**GROUP BY** *dno*

*sort* **por** *dno* **não usa índice**

**Se Árvore B+ por** *dno* **(agrupada ou não),**
**basta contar número de entradas no índice**

## Considerações finais

- ***Chaves compostas em caso de Árvores B+ podem usar prefixo como chave***

**EXEMPLOS:**

- **Árvore B** $< age, sal >$ **ou** $< sal, age >$ **para**

```
SELECT    eid
FROM      employee
WHERE     age BETWEEN 20 AND 30
          AND sal BETWEEN 3000 AND 50000
```

- **Árvore B** $< age, sal >$ **agrupada é melhor que** $< sal, age >$ **para**

```
SELECT    eid
FROM      employee
WHERE     age = 25
          AND sal BETWEEN 3000 AND 50000
```

## Considerações finais

- **Especificação de índices no SQL 99**

***CREATE INDEX IndAgeGrau***
***ON Estudantes***
***WITH STRUCTURE = BTREE,***
***KEY = (Idade, Média)***

***Exercícios***

# *Fim - Organização de Arquivos*