

 deti departamento de  
electrónica, telecomunicações  
e informática

## Indexação e Optimização

↳ Bom para o mercado de trabalho

**Base de Dados - 2019/20**  
**Carlos Costa**

1

1

 deti

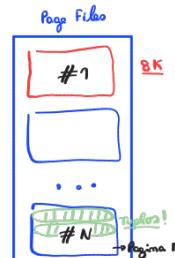
## Introdução - Motivação

- Imaginemos a seguinte consulta executada numa base de dados de uma sistema de informação hospitalar contendo milhões de pacientes:

```

    Pesquisar paciente...
    SELECT Fname, Lname, Patient_ID
    FROM Patients
    WHERE Fname='Mario' AND Lname='Pinto';
  
```

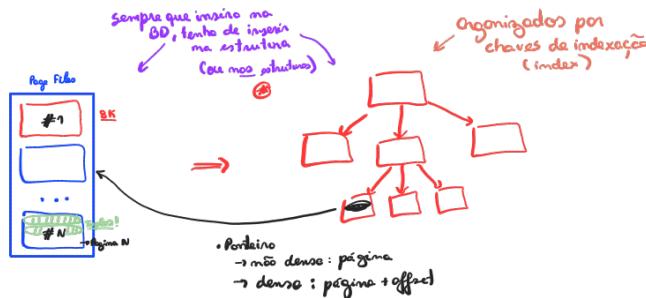
- Questões: Com é que o SGBD procura este paciente em tempo útil?
  - Percorre a relação tuplo a tuplo? ->  $O(n)$  !!!
  - Ordenação dos atributos?
  - Quais estruturas de dados a utilizar?
- Imagine-se que envolve a junção de relações e critérios de seleção com atributos de ambas as relações!



Se eu souber o ID do utilizador:  
 → uso offset & var directamente  
 Se não souber:  
 → percorro tudo -  $O(n)$

2

- ④ Para escolher os Index Keys temos de pensar...  
 → Sexo não deve ser!  
 • tem um conjunto limitado...  
 → Nome é bom  
 → ID...  
 { Dependendo do que queremos pesquisar!}



5/2/13

### B-Tree

Pesquisa Binária:  $\log_2 n$

se tiver  $n$  níveis  
 isto muda para

## Índices



- Guardo
- E permite saber onde está
- vou diretamente

Não GPUs também  
 é crítico!!!

- O problema agora é o tempo de I/O!  
 → Por isso é que parallelizam?

- Índices (indexes) são estruturas de dados que oferecem uma segunda forma (rápida) de acesso aos dados.
  - Melhora o tempo de consulta - crítico para desempenho de BD
  - Pode aumentar o volume de dados armazenado (overhead) e o tempo das inserções *Não exagerar!*
- É possível:
  - indexar qualquer atributo da relação
  - criar múltiplos índices (sobre atributos distintos)
  - criar índices com vários atributos
- Os atributos indexados denominam-se por
  - *Index Key*
- Existem índices implementados com diversas estruturas de dados tendo em vista objectivos diferenciados.

3

3



## Índices - Organização Física dos Dados

- Num SGBD os tuplos de uma relação estão distribuídos (armazenados) por várias páginas (ou blocos) em disco.
  - Cada página tem tipicamente milhares de bytes e suporta muitos tuplos.
  - Os tuplos de uma relação estão tipicamente distribuídos por várias páginas.
  - Os ficheiros (em disco) estão organizados em páginas.
- Os índices <sup>posso ter índices compostos</sup> são estruturas que:
  - Têm um valor ordenado (atributo indexado)
  - Um ponteiro para a sua localização
    - Não Denso: Início da Página (Bloco)
    - Denso: Offset do próprio tuplo na página
- Os índices também são guardados em páginas

4

4

2

## Índices

### Tipos Genéricos

- Single-Level Ordered
- Multi-Level

5

5

### Single-Level Ordered

- São estruturas de um único nível que indexam um atributo da relação.
  - Armazena cada valor do atributo indexado e a respectiva localização do tuplo na relação (ponteiro para a estrutura física de dados da tabela).
  - Índices são ordenados o que permite pesquisa binária sobre o atributo.

Analogia: Índice de um livro (palavra - página)

*Queremos melhor!,,*

- Permite uma pesquisa binária com complexidade:  $\log_2 b_i$  ( $b_i$  - index blocks)
  - A cada etapa do algoritmo, a parte da estrutura do índice a pesquisar é reduzida num factor de 2.

6

6

Literatura...

## Single-Level Index - Tipos

Fazemos como é possível organizar a informação!

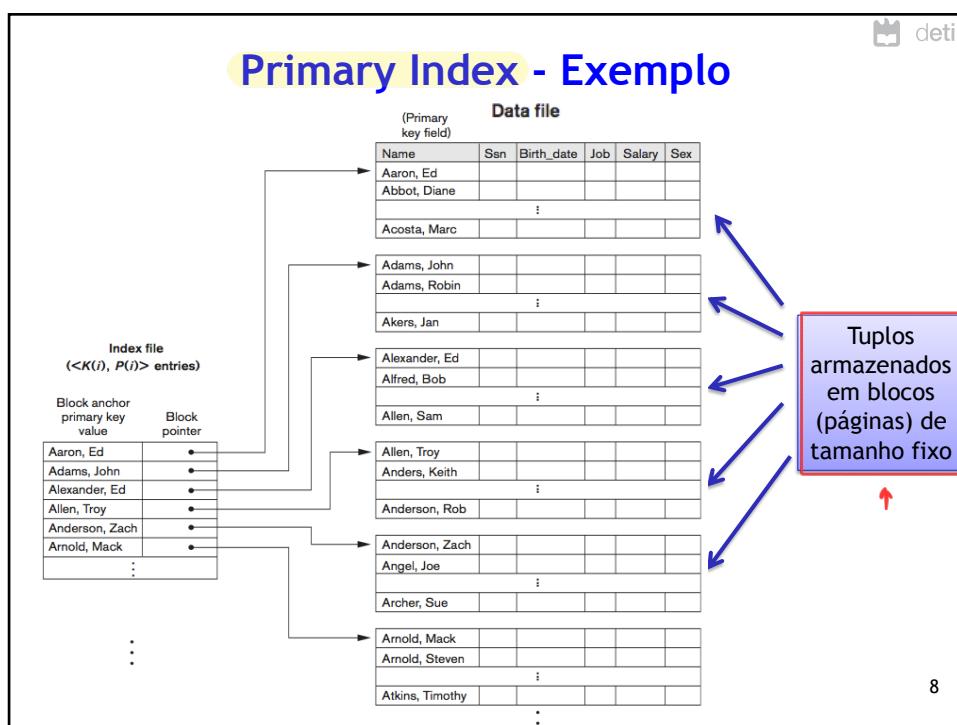
- Primary Index
  - Indexa um atributo **chave** da relação (não se repete)
- Clustered Index
  - Indexa um **atributo** que pode ter valores duplicados
  - Os atributos estão **agrupados**
- Secondary Index
  - Indexa **outros atributos** (chave candidata ou não chave)
  - Podemos ter **vários índices** deste tipo

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

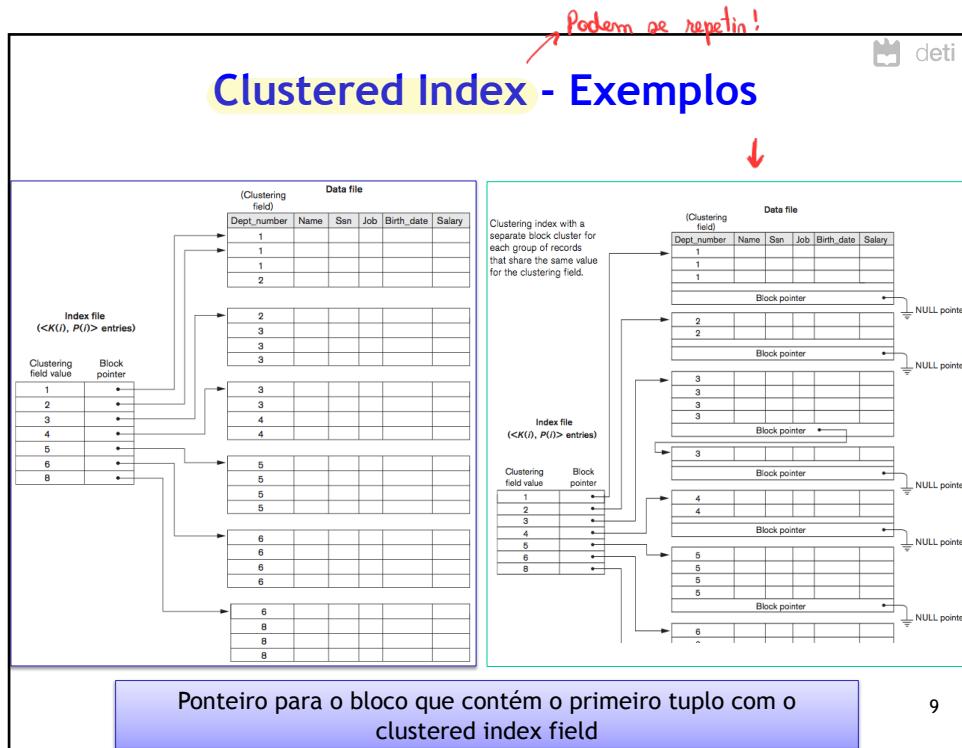
7

Nota: Só podemos ter um primary ou clustered

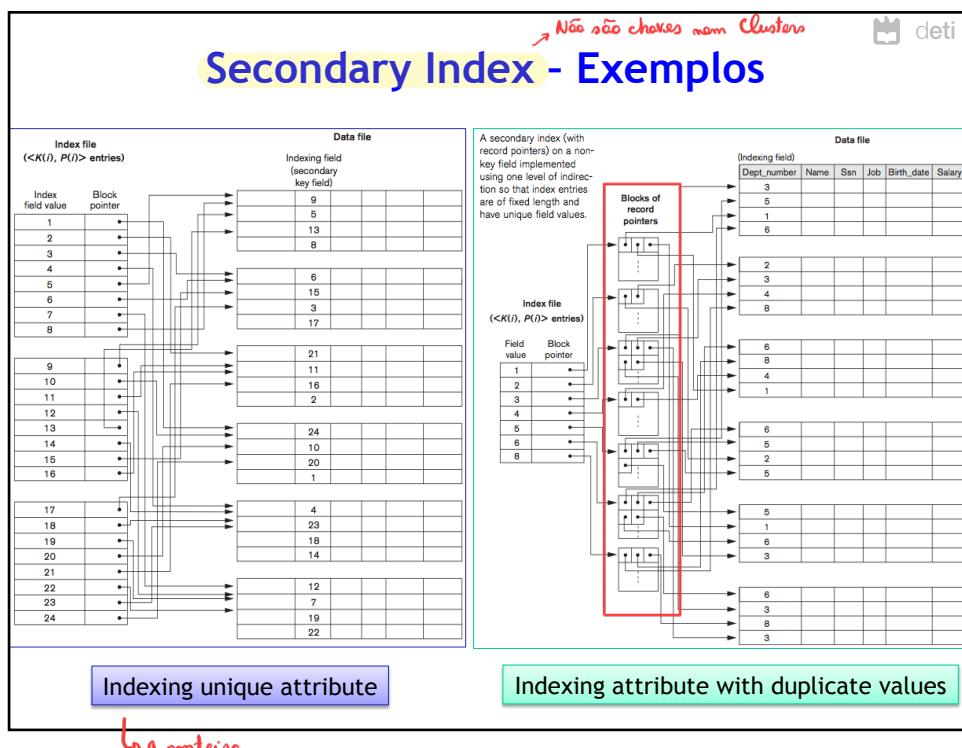
7



8



9



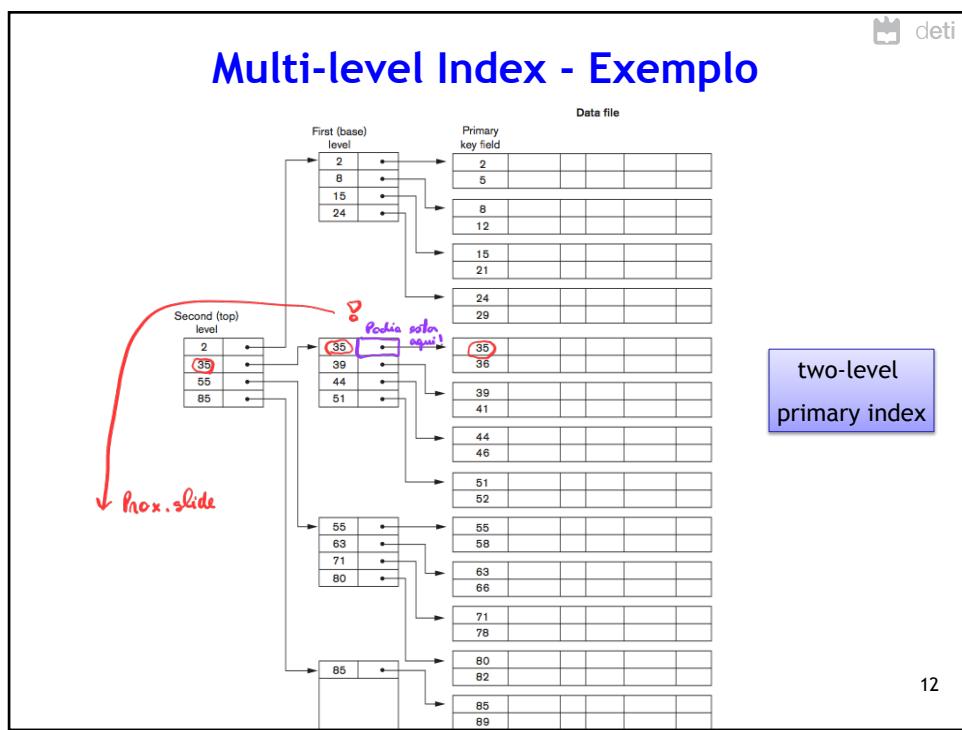
10

**Multilevel Index** → Organização em árvore (B-Tree)

- Single-Level: complexidade  $\log_2 b_i$  ( $b_i$  - index blocks)
- A ideia do multi-level é ter **vários níveis de indexação** passando a complexidade para:  $\log_{f_0} b_i$ 
  - Fan out:  $f_0$
  - A cada etapa do algoritmo estamos a reduzir o espaço de procura num factor  $f_0$
  - Usualmente  $f_0 > 2$   
Não são binários
- Estes índices são tipicamente implementados com estruturas em árvore balanceadas (equilibradas)
  - B-Tree

11

11



12

**Árvore de Pesquisa**

*Search tree of order p*

Cada nó contém, no máximo:  
 p - 1 valores e p ponteiros  
 !

*Tenho q elementos*

Subtree for node B

Root node (level 0)

Nodes at level 1

Nodes at level 2

Nodes at level 3

Diagram showing nodes  $P_1, K_1, \dots, K_{i-1}, P_i, K_i, \dots, K_{q-1}, P_q$  connected to a search tree. The tree has three levels of nodes labeled X. A red arrow points from the text "Não está equilibrado!" to the tree.

- *key value* tem associado um ponteiro para o registo de dados file/page/row
- $P_i$  - ponteiro para um nó filho ou NULL
- $K_i$  - valor a pesquisar

$K_1 < K_2 < \dots < K_{q-1}$   
 $K_{i-1} < X < K_i$  para  $1 < i < q$ ;  $X < K_i$  para  $i = 1$ ;  $K_{i-1} < X$  para  $i = q$

13

13

**Árvore de Pesquisa - Exemplo**

$p = 3$

Diagram illustrating a search tree of order 3. The root node contains key 5 and pointer 0. A red circle highlights the pointer 0 with the note "Podia ter mais um filho". The tree has two children: one pointing to a node with keys 3 and 1, and another pointing to a node with keys 6, 9, 7, 8, and 12. A blue box indicates "2 valores por nó" and "3 filhos por nó".

Legend:

- Tree node pointer
- Null tree pointer

14

14

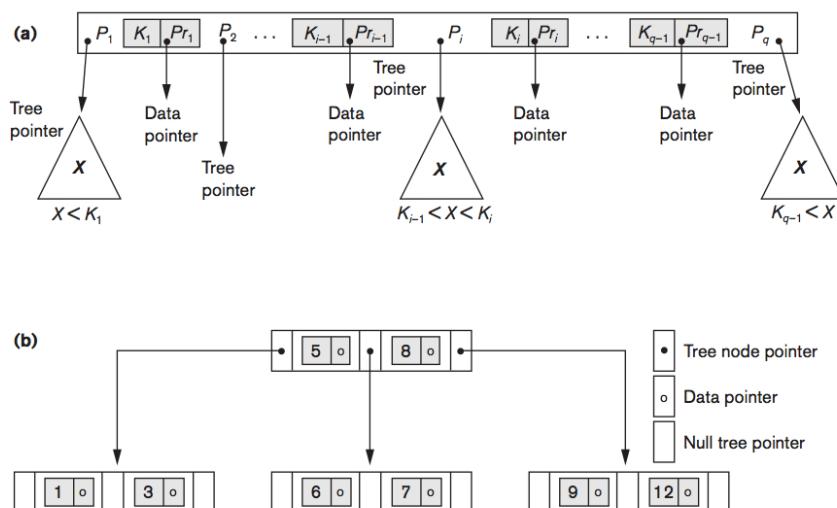
## Árvore de Pesquisa - Balanceada

- Uma árvore diz-se balanceadas (equilibrada) se a distância de qualquer folha ao nó raiz for sempre a mesma
  - i.e. os nós folha estão todos ao mesmo nível
- As operações de inserção e remoção são efectuadas segundo um algoritmo que mantém a árvore sempre equilibrada.
- B-Tree são árvores balanceadas muito utilizadas pelos SGBD para implementar índices multi-level
  - permitem uniformizar os tempos de pesquisa de valores na estrutura.

15

15

## B-Tree



B-tree structures. (a) A node in a B-tree with  $q - 1$  search values. (b) A B-tree of order  $p = 3$ . The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

16

**SQL - Index**

- SQL standard - não normaliza índices
- Existe uma sintaxe comum a muitos SGBD:

```

CREATE INDEX index_name ON Relation(attribute_list)
    -- Criar um índice para o atributo Pname

CREATE INDEX idxPName ON Project(Pname);
    -- Criar um índice multi-atributo
    CREATE INDEX idxEmpName ON Employee(Fname, Minit, Lname) Composite
    
```

```

DROP INDEX index_name
    -- Eliminar índice idxPName
    DROP INDEX idxPName; Falta "ON Employees"
    
```

Índice multi-atributo justifica-se se efetuarmos pesquisas contendo todos os atributos do Key Index (Fname, Minit, Lname). 17

17

**Seleção de Índices - Overview**

- A criação de índices deve ser criteriosa pois existem **mais e menos** valias:
  - Um índice pode acelerar o processo de pesquisa de um valor num atributo (ou gama de valores) e junções envolvendo esse atributo.
  - Um índice introduz overhead ao nível do volume de dados e do tempo de inserção, atualização e eliminação de tuplos (i.e. as operações são mais complexas).
- A escolha deve ser um compromisso entre:
  1. perceber se vamos ter necessidade de efetuar muitas pesquisas envolvendo determinado atributo (candidato a index).
  2. perceber se determinada relação vai ter modificações frequentes de dados.
- Recomendação:
  - estudar o tipo de consultas das aplicações e/ou utilizar registos de log (histórico de <sup>18</sup>queries) para ajudar na decisão.

18

## Seleção de Índices - Factor “Organização Física dos Dados”



- Os tuplos estão distribuídos por várias páginas (blocos).
- A pesquisa de um único tuplo obriga a carregar em memória toda a página onde ele se encontra.
  - Operações de I/O são bastante onerosas em termos temporais
- Os próprios índices também são guardados, total ou parcialmente, em páginas:
  - Operações de acesso e modificação dos índices também tem custos temporais

Recomendação:

- Tentar perceber se determinado índice obriga a carregar muitas (ou poucas) páginas em memória num processo de pesquisa
- Índices que necessitam de carregar poucas páginas da relação, para encontrar o tuplo, reduzem significativamente o tempo de pesquisa<sup>19</sup>

19

## Seleção de Índices - Critérios Genéricos

- Indexação das chaves da relação
  - Pesquisamos frequentemente por atributos chave da relação
  - Sendo a chave única, ou existe tuplo ou não.
    - Só uma página é carregada para memória para ter o tuplo.
- Se o índice não é chave da relação podemos ter (ou não) ganhos no tempo de pesquisa. Há duas situações recomendadas:
  - O atributo indexado tem poucos valores repetidos.
  - A relação têm o atributo indexado do tipo *clustered*.
    - Clustering é agrupar os valores desse atributo de forma a que ocupem o menor número de páginas.

20

20

## Seleção de Índices

### Caso de Estudo

21

21

## Cenário

- Tento 10 páginas
- Tento em média 1 ator → 3 Filmes
- 1 filme → 3 atores

- Relação: StarsIn(movieTitle, movieYear, starName)

### 3 operações usuais sobre a base de dados:

#### Q1: Procurar os títulos e ano de filmes de determinado ator

```
SELECT movieTitle, movieYear
FROM StarsIn
WHERE starName = s;
```

$\approx 3 \text{ elem.}$

-- s constante

#### Q2: Procurar os atores de determinado filme

```
SELECT starName
FROM StarsIn
WHERE movieTitle = t AND movieYear = y;
```

$\approx 3 \text{ elem.}$

-- t e y constantes

#### I: Inserir novo tuplo

```
INSERT INTO StarsIn VALUES(t, y, s);
```

-- t, y e s constantes

22

22

deti

## Pressupostos

- A relação StarsIn ocupa 10 páginas
- Caso de não indexação
  - custo de 10 para examinar todos os tuplos da relação
    - partindo do princípio que não estamos a utilizar clustering
  - depois de encontrado o primeiro tuplo, não é necessário fazer scan à relação toda para encontrar os tuplos adicionais.
- Em média, cada ator aparece em 3 filmes e um filme tem 3 atores
- Query - se tivermos indexado starName ou (movieTitle, movieYear)
  - custo médio de 3 acessos a disco para um filme ou ator
  - 1 acesso a disco é necessário para consultar um índice.
- Inserção - obriga a acessos a disco (leitura e escrita)...
  - à página onde vai ser inserido o novo tuplo.
  - para modificar o próprio índice.

23

23

A simétrica é: r/w Páginas

deti

## Tabela de Custos

M = 10 páginas

Action	No Index	Star Index	Movie Index	Both Index
Q1	$m = 10$ [leitura todo]	4 [ler índice den as 3 páginas]	10 $p_1$	4 [Somar na procura]
Q2	$m = 10$	10	4 $p_1$	4
I	2 [carga a página. Executar com o ator] 4 [No índice. Não existindo]	4	4 $p_1$	6 → Logo na inserção!
Cost Formula	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

$P_1$  e  $P_2$  - fração de tempo em que ocorre  $Q_1$  e  $Q_2$   
 Fracção de tempo em que fazemos I é:  $1 - P_1 - P_2$

- No index:
  - Q1 e Q2: Acesso a toda a relação (full scan) - 10 acessos de leitura
  - I: 1 acesso para consulta + 1 acesso para escrita
- Star Index:
  - Q1: 1 acesso ao index + 3 acessos páginas da relação
  - Q2: No index - custo 10
  - I: 2 acessos página do índice + 2 acesso páginas dos dados da relação
- Movie Index: simétrico de Star index
- Both Index:
  - Q1 e Q2: 1 acesso ao index + 3 acessos páginas da relação
  - I : (2 leitura + 2 escritas) para cada índice (total de 4) + 2 acessos à página os dados

24

24

• Qual o custo?



Já incluem TODOS  
a inserção física  
é imediata!

**Escolha de Índice(s)**

- Temos de olhar para a fórmula da custo
  - Depende dos valores de  $P_1$  e  $P_2$

**Exemplos:**

Action	No Index	Star Index	Movie Index	Both Index
Cost Formula	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

*Se usarmos  
páginas  
privadas*

*inserção demora  
8x mais!*

*procuro muito  
é preferível  
...*

1.  $P_1 = P_2 = 0.1$ 
  - Menor custo:  $2 + 8p_1 + 8p_2$
  - Opção: *No Index*
2.  $P_1 = P_2 = 0.4$ 
  - Menor custo:  $6 - 2p_1 - 2p_2$
  - Opção: Indexar starName e (movieTitle, movieYear)
3.  $P_1 = 0.5$  e  $P_2 = 0.1$ 
  - Menor custo:  $4 + 6p_2$
  - Opção: Indexar só starName

25

25

**Índices**

**SQL Server**

26

26

**SQL Server - Indexing**

**SQL Server tem dois tipos de índices\*:**

**ESTRUTURA BASE** → • **clustered**

- Os nós folha contêm os próprios dados da relação
- A tabela está ordenada pelo próprio índice
  - Só existe um por relação (default é a primary key)
- Analogia: Agenda de contactos telefónicos

cluster index!  
Estrutura base

**non-clustered** → indices secundários que trabalham sobre os principais

- Os índices apontam para a tabela base
  - que pode ser clustered ou heap (tabela non-clustered)
- Podemos ter vários numa relação
- Analogia: Índice no fim de um livro

temos uma estrutura com os ponteiros

*Saber bem os diferentes*

\*ambos implementados com B-trees

27

27

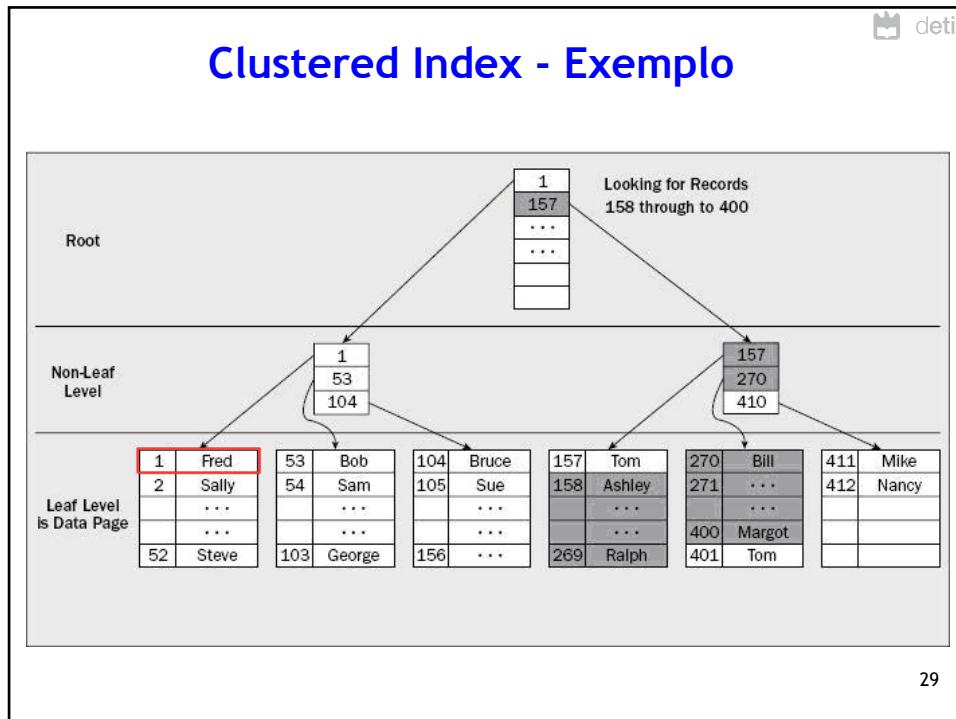
**SQL Server: Clustered index**

- Estrutura “dois em um”: índice + dados da relação
  - B-Tree ordenada pelo *cluster key index*
  - Dados nas folhas das árvores

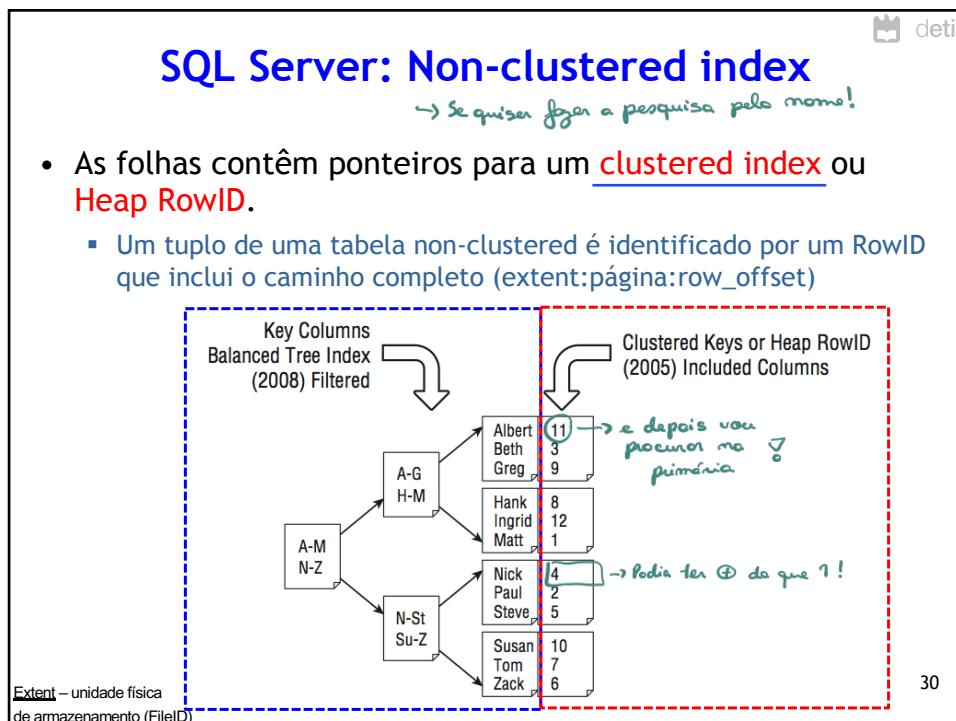
The diagram illustrates a clustered index structure. It features a 'Balanced Tree Index' on the left with three levels of nodes. The top level node contains ranges '1-6' and '7-12'. The middle level node contains ranges '1-3', '4-6', and '7-9', '10-12'. The bottom level nodes contain ranges '1-3', '4-6', '7-9', and '10-12'. Arrows point from these nodes to a 'Data Columns' section on the right. This section consists of four tables, each with three rows. The first table (rows 1-3) contains 'Matt', 'Paul', and 'Beth'. The second table (rows 4-6) contains 'Nick', 'Steve', and 'Zack'. The third table (rows 7-9) contains 'Tom', 'Hank', and 'Greg'. The fourth table (rows 10-12) contains 'Susan', 'Albert', and 'Ingrid'. A dashed blue box encloses the 'Key Columns' and the 'Balanced Tree Index' section, while a dashed red box encloses the 'Data Columns' section.

28

28

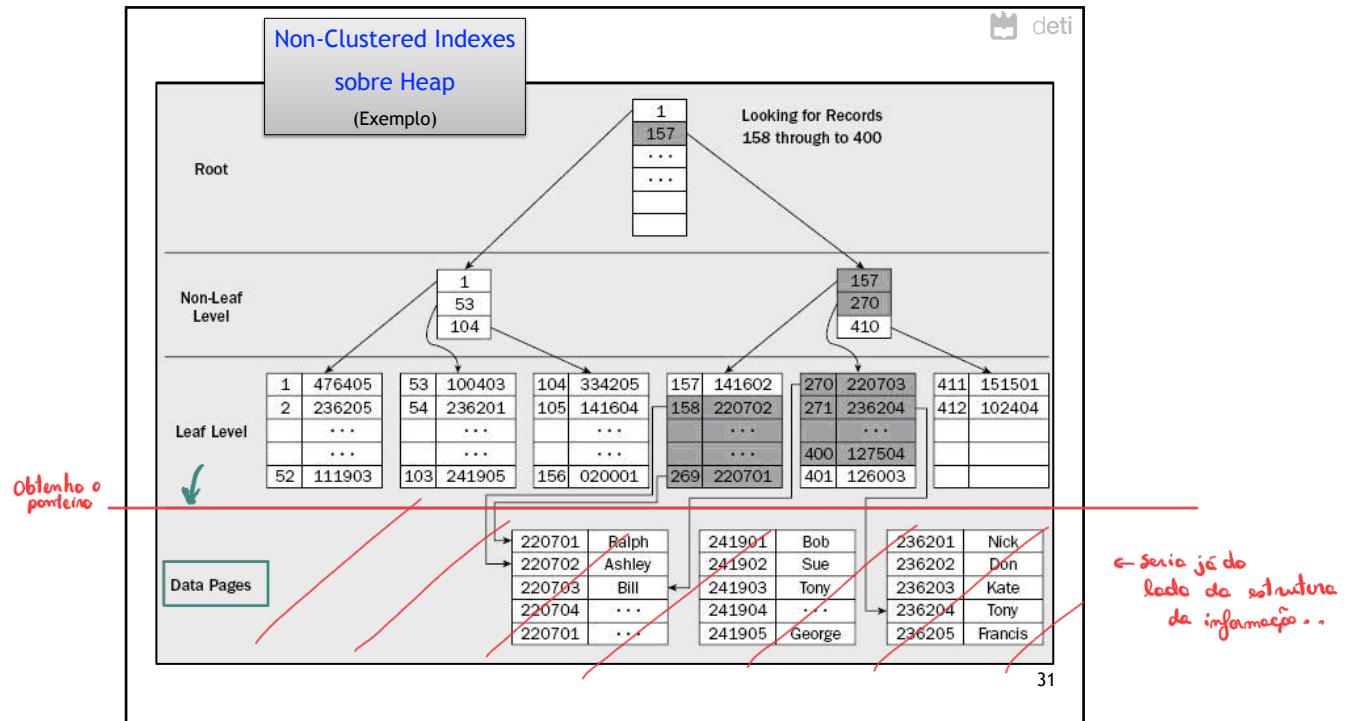


29

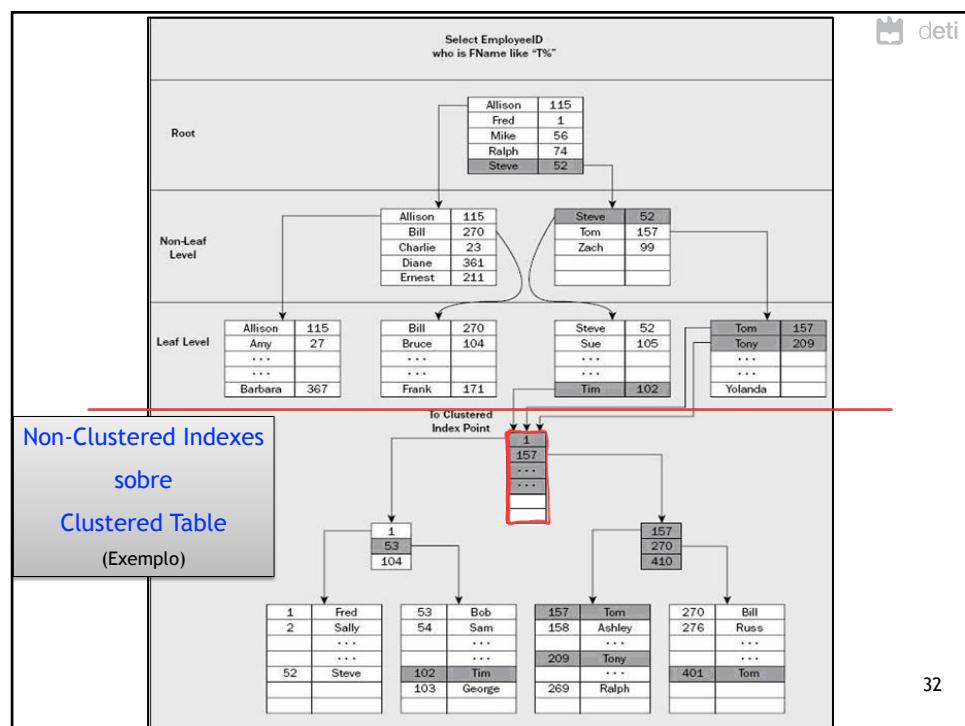


30

↓ Tenho depois  
a primária!



31



32

• Para charas primária é boa!  
• O custo de inserção é maior!

## B-Tree Page Split

→ Página ficou cheia  
→ Tento que partiu...

- Os índices da B-Tree devem manter-se ordenados pelo key index.
- Inserts, updates e deletes afectam os dados.
- O que acontece quando pretendemos fazer um insert e a página está cheia?
  - O SGBD divide a página cheia em duas (**page split**)
    - cria uma nova página
    - copia parte dos índices para a nova página
    - reflete esta nova realidade nos nós hierarquicamente superiores
    - insere o novo índice
  - O processo de **page split** é particularmente **penalizador** em termos de **desempenho temporal**.

Dividir página em duas  
→ Imprimimos aqui!

33

33

## B-Tree Page Split - Exemplo

1. B-tree structure of index before index page splitting

High-order page

6	12

Key 5 is to be added

5
---

Index page a

2	4	6

Index page c

8	10	12

2. Index page splitting

High-order page

4	6	12

Index page a'

2	4	

Index page b

5	6	

Index page c

8	10	12

• Penalizada em tempo  
• Muitas páginas com 50% da ocupação  
→ Fragmentos imóveis ...

Legend:  
Grey box: Locations where the index structure was changed by index page splitting.  
White box with black border: Key that was moved to another page by index page splitting.  
(Data is divided evenly between index pages a and b.)

34

34

**Índices - Opções de Especialização**

- **Unique** *para criar clustered index!*
  - A index key é única (não tem valores duplicados)
  - Por defeito, a criação de uma **chave primária** cria automaticamente um **unique clustered index** ! ← CLUSTERED index
  - Opcionalmente, podemos criar um unique non-clustered index
- **Composite**
  - Índice com vários atributos
  - A ordem dos atributos importa
    - Um índice só é considerado como podendo ser usado se a primeira coluna faz parte da query
    - Assim, podemos ter necessidades de índices com o mesmo atributo em ordem diferente.

35

35

**Índices - Opções de Especialização**

- **Filtered** *só parte da estrutura!*
  - Permite utilização da cláusula WHERE no CREATE INDEX
  - Só disponível para non-clustered index
  - Só indexamos parte dos tuplos da relação //
- Inclusão de Atributos num Índice
  - Podemos incluir non-key atributos nas folhas de um índice non-clustered.
  - Chamadas query “cobertas”
    - query em que todos os dados de que a query necessita estão no índice

36

36

**SQL Server Indexes - Exemplos**

```
-- Clustered Index
CREATE CLUSTERED INDEX IxOrderID ON OrderDetail(OrderID);

-- Clustered Composite Index
CREATE CLUSTERED INDEX IxGuideName ON Guide (LastName, FirstName);

-- Clustered UNIQUE Index
CREATE UNIQUE CLUSTERED INDEX IxGuideName ON Guide (LastName, FirstName);

-- FILTERED Index
CREATE INDEX IxActiveProduction ON Production.WorkOrders
(WorkOrderID, ProductID) WHERE Status = 'Active';
Apenas os que  
estão ativos!

-- Non-clusteres with column include
CREATE INDEX ixGuideCovering ON dbo.Guide (LastName, FirstName)
INCLUDE (Title);
```

37

**SQL Server: Heap vs Clustered Table**

Estudar cada caso...

- Ter sempre presente que:
  - Uma **heap table** insere os novos registos no final da tabela (unsorted).
    - um non-clustered índice (B-Tree) contém, nos nós folha, um RowID da heap.
  - Uma **clustered table** introduz o novo registo na B-Tree segundo a ordem da cluster index key.
- **Insert operation**- desempenho de uma solução clustered table está muito associado à ocorrência de page splits no processo de inserção:
  - depende das características da chave primária
  - dependente do facto dos novos tuplos terem (ou não) uma ordenação natural

38

38

 deti

## Escolha de um Clustered Index

- Chave Primária: "unique clustered index" (defeito)
  - verificar se esta opção é boa/desejada.
- “Evitar” chaves susceptíveis de criar “page split”
  - inserções sequenciais são boas
    - Exemplo: auto number - IDENTITY
- Chaves pequenas são preferíveis
  - Lembrar que são utilizadas nos índices não clustered...

✿

39

39

 deti

## Escolha de um Non-Clustered Index

- São tipicamente utilizados para optimizar os tempos das consultas
  - Atributos que aparecem frequentemente na cláusula WHERE.
- Ter em atenção os critérios genéricos de seleção referidos anteriormente.
  - SQL Server tem uma ferramenta (DBCC Show\_Statistic) que permite fazer o tracking da seletividade de um índice utilizando para o efeito estatísticas de utilização.
- Atributos chave estrangeira
  - JOINS
- Atributos sobre os quais são efetuadas consultas ordenadas

Dec-meos matriciais! (não normalmente jogamos da fábrica!)

e.g.: Dno

40

40

## B-Tree Tuning

- Objectivo: minimizar os Page Splits
- Index fill factor e pad index
  - Um índice necessita de ter um pouco mais de espaço livre em cada página para evitar que novas entradas obriguem a page split.
  - Fill factor permite definir a % de espaço livre. → Digno a % que este ocupado!
  - Pad index indica se só aplicamos o fill factor aos nós folhas (ou não). ON/OFF
- Exemplo:
 

```
-- index com 15% de espaço livre nas nós folha e intermediário
CREATE NONCLUSTERED INDEX IX_OrderNumber ON dbo.[Order]
(OrderNumber) WITH (FILLFACTOR = 85, PAD_INDEX = ON);
```
- Best Practice:
  - Compromisso entre a compactação dos dados (menor desperdício de espaço) e a ocorrência de page splits:
    - Utilizar fill factor próximo de 100% se temos inserções ordenadas ← usando o identity
    - 65-85% se tivermos mais inserções no meio da B-Tree

*→ Deixar em espaço vazio, para não perder. Em toda a estrutura ou só nos intermediários*

41

## Desfragmentação de Índices

- Processo de eliminação de “espaços vazios” resultantes:
  - Page Splits (1 page FULL 100% -> 2 pages ~50%)
  - Remoções de tuplos
- Regularmente devemos:
  - Verificar estado de fragmentação do índice  
“SQL Server sys.dm\_db\_index\_physical\_stats reports the fragmentation details and the density for a given table or index”
  - Reconstruir o índice caso este esteja muito fragmentado
 

```
ALTER INDEX IndexName ON TableName REORGANIZE
```

    - desfragmenta (ao nível das folhas) de acordo com o fill factor do índice
    - efectuado num conjunto de pequenas transações sem impacto nas operações de insert, update e delete.

*→ Desfragmentação do índice*

ou

```
ALTER INDEX ALL ON Frag REBUILD WITH (FILLCFACTOR = 98)
```

  - reconstrói o índice completamente (equivalente a um DROP + CREATE)
  - podemos alterar as características do índice. Por exemplo, o fillfactor.

42

## Desfragmentação de Índices - Exemplo

**Fragmentação dos índices da tabela Frag:**

```
USE tempdb;
SELECT * FROM sys.dm_db_index_physical_stats ( db_id('tempdb'),
object_id('Frag'), NULL, NULL, 'DETAILED');
```

index_id: 1 index_type_desc: CLUSTERED INDEX avg_fragmentation_in_percent: 99.1775717920756 page count: 22008 avg_page_space_used_in_percent: 68.744230294045	index_id: 2 index_type_desc: NONCLUSTERED INDEX avg_fragmentation_in_percent: 98.1501632208923 page count: 2732 avg_page_space_used_in_percent: 58.2316654311836
---	--

**Desfragmentar os dois índices (PK\_Frag e ix\_col):**

*→ Fato → FILLFACTOR*

```
USE tempdb;
ALTER INDEX PK_Frag ON Frag REORGANIZE;
ALTER INDEX ix_col ON Frag REORGANIZE;
```

index_id: 1 index_type_desc: CLUSTERED INDEX avg_fragmentation_in_percent: 0.559173738569831 page count: 15201 avg_page_space_used_in_percent: 99.538930071658	index_id: 2 index_type_desc: NONCLUSTERED INDEX avg_fragmentation_in_percent: 1.23915737298637 page count: 1614 avg_page_space_used_in_percent: 99.487558685446
--	---

*↓ Devia ter ficado 65-85% ↓*

*Nos isto danifica discos...*

43

## Problemas de Desempenho?

Query Execution Plan

- Com uma query?  
*... consultar o “execution plan”*

The execution plan diagram illustrates the flow of data from various tables through joins, scans, and scalar computations to produce the final results. It highlights the cost of each operation relative to the total query cost.

44

**Problemas de Desempenho na BD?**

1. Utilizar o SQL Server Profiler para capturar eventos
  - Ficheiro ou Tabela
2. Sujeitar a base de dados a um conjunto de consultas usuais
  - Idealmente - capturar alguns dias com a BD em produção
3. Utilizar os resultados da sessão do Profiler na ferramenta Database Engine Tuning Advisor

45

45

**Profiler + Engine Tuning Advisor**

The screenshot shows the Microsoft SQL Server Management Studio interface. On the left, the Object Explorer pane is open, showing a tree structure of database objects under 'BD-SQLSERVER\SQLEXPRESS2012 (SQL Server)'. In the center, a window titled 'BD-SQLSERVER\SQLEXPRESS2012 - CCosta\_AdventureWork\_IndexTuning' is displayed, which is the 'Database Engine Tuning Advisor' session. The 'General' tab is selected. The 'Session name:' field contains 'CCosta\_AdventureWork\_IndexTuning'. Under 'Workload', the 'File' radio button is selected, and the path 'C:\Users\Carlos Costa\Documents\Profiler\_Adventure\_CC1.tic' is shown. The 'Database for workload analysis:' dropdown is set to 'AdventureWorks2012'. Below this, the 'Select databases and tables to tune:' section shows a list of databases and tables. The 'AdventureWorks2012' database is selected, and its tables (company, master, model, msdb, tempdb, .p11...) are listed under 'Selected Tables'. A note at the bottom of this section says 'Provide a new session name. In the Workload section, select a database to which Database Engine Tuning Advisor will connect for analyzing the workload.' At the bottom right of the session window, it says 'Connections: 2'.

46

## Resumo

- Conceito de Índice
- Tipos de Indexação
  - Critérios de seletividade
- Estruturas B-Tree
- Indexação em SQL Server

47

47

## SQL Server

Tools

48

48



## Index Selectivity - DBCC Show\_Statistic

SQL Server uses its internal index statistics to track the selectivity of an index. DBCC Show\_Statistic reports the last date on which the statistics were updated, and basic information about the index statistics, including the usefulness of the index. A low density indicates that the index is very selective. A high density indicates that a given index node points to several table rows and that the index may be less useful, as shown in this code sample:

```
Use CHA2;
DBCC Show_Statistics (Customer, IxCustomerName);
```

Result (formatted and abridged; the full listing includes details for every value in the index):

Statistics for INDEX 'IxCustomerName'.					
	Rows	Sampled	Steps	Average Density	key length
Updated	Rows	Sampled	Steps	Density	key length
May 1,02	42	42	33	0.0	11.547619
All density	Average Length	Columns			
3.0303031E-2	6.6904764	LastName			
2.3809524E-2	11.547619	LastName, FirstName			

DBCC execution completed. If DBCC printed error messages,  
contact your system administrator.

Sometimes changing the order of the key columns can improve the selectivity of an index and its performance. Be careful, however, because other queries may depend on the order for their performance.

49