

Indexação e Optimização

Base de Dados - 2024/25

Carlos Costa

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!

Índices

- 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
- 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.

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** 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

Índices

Tipos Genéricos

- Single-Level Ordered
- Multi-Level

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)

- 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.

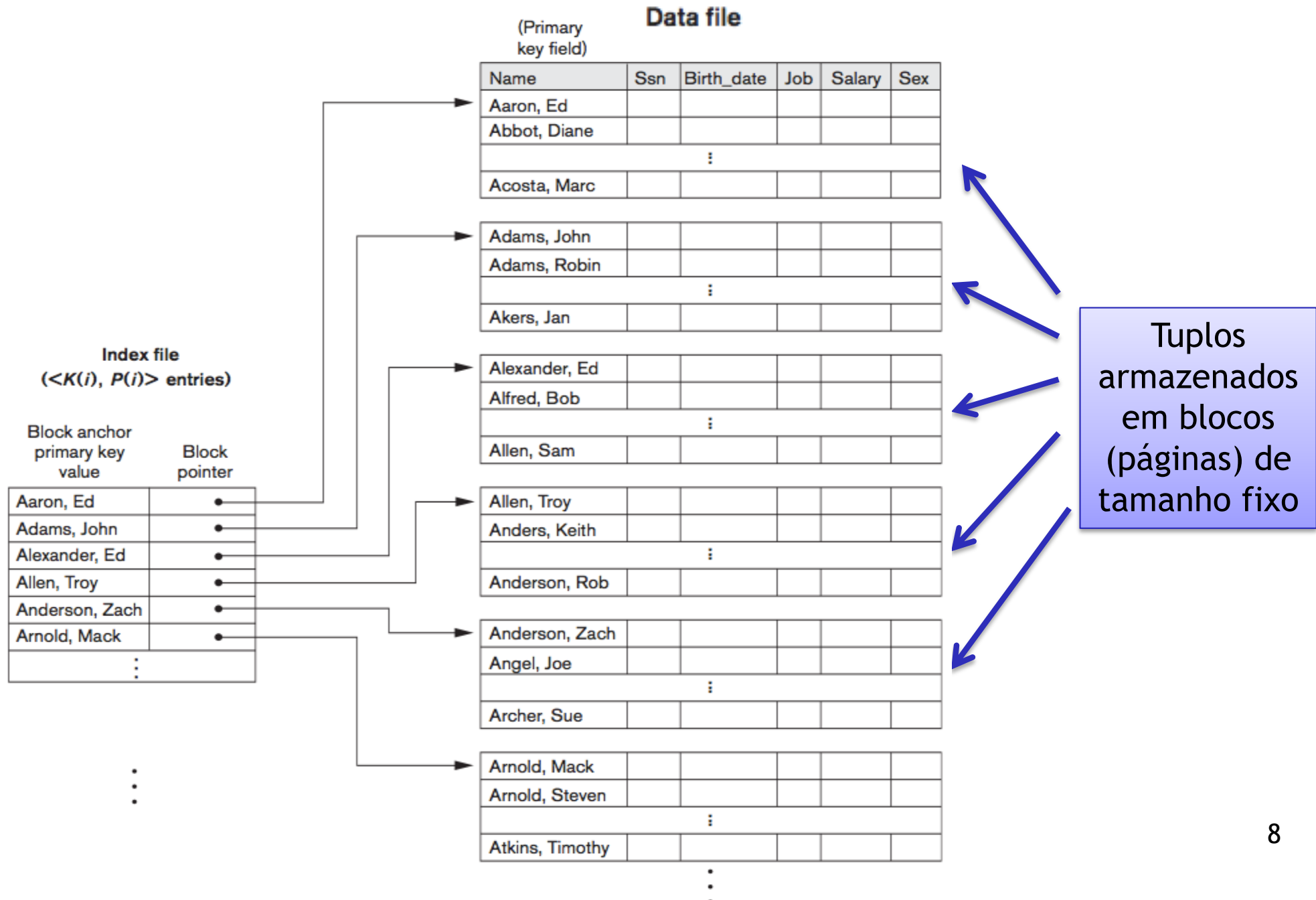
Single-Level Index - Tipos

- 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

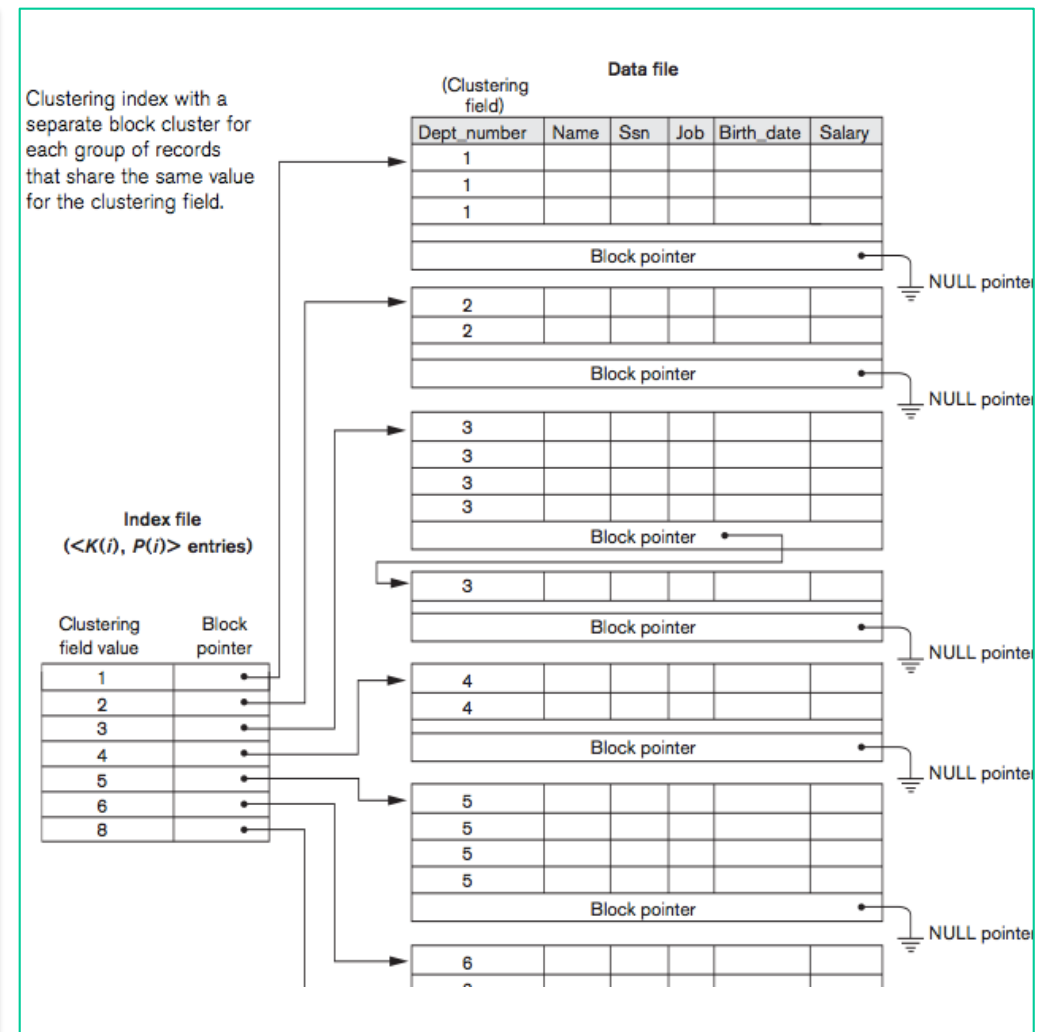
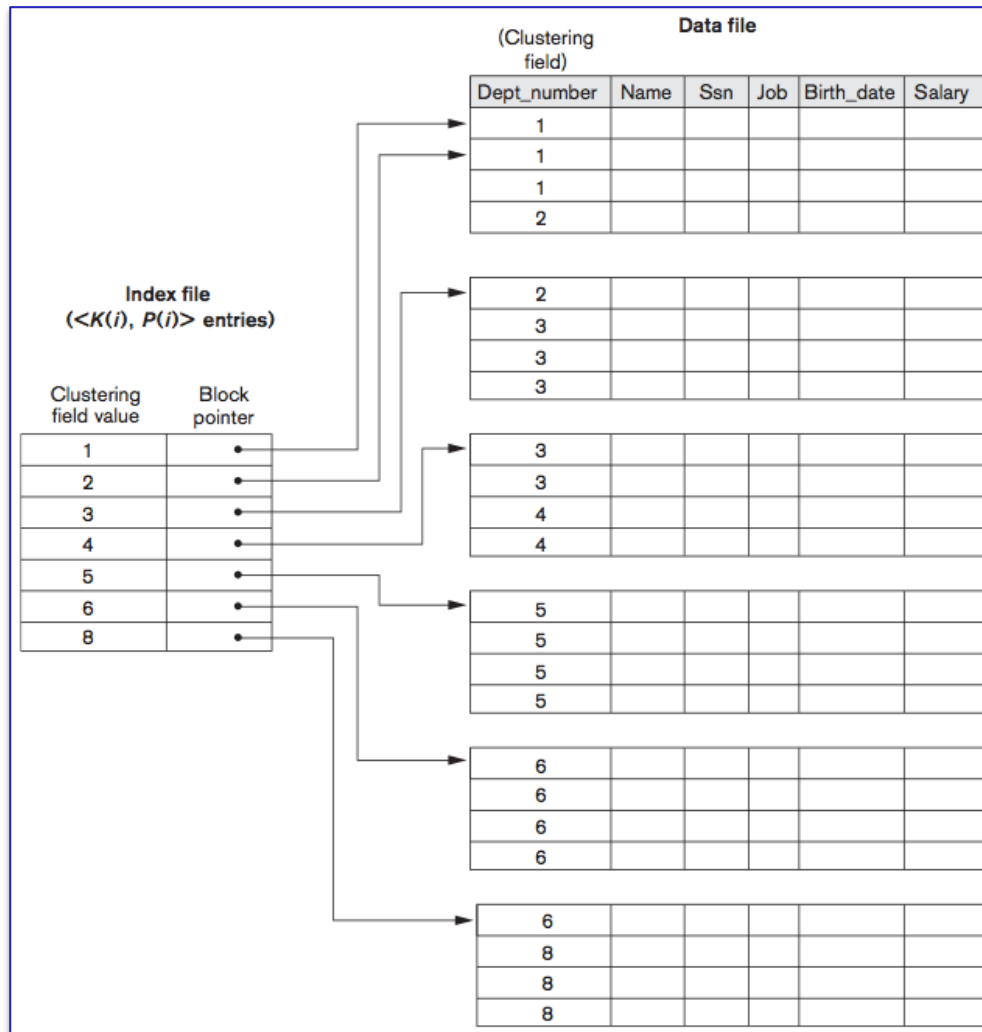
	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Nota: Só podemos ter um primary ou clustered

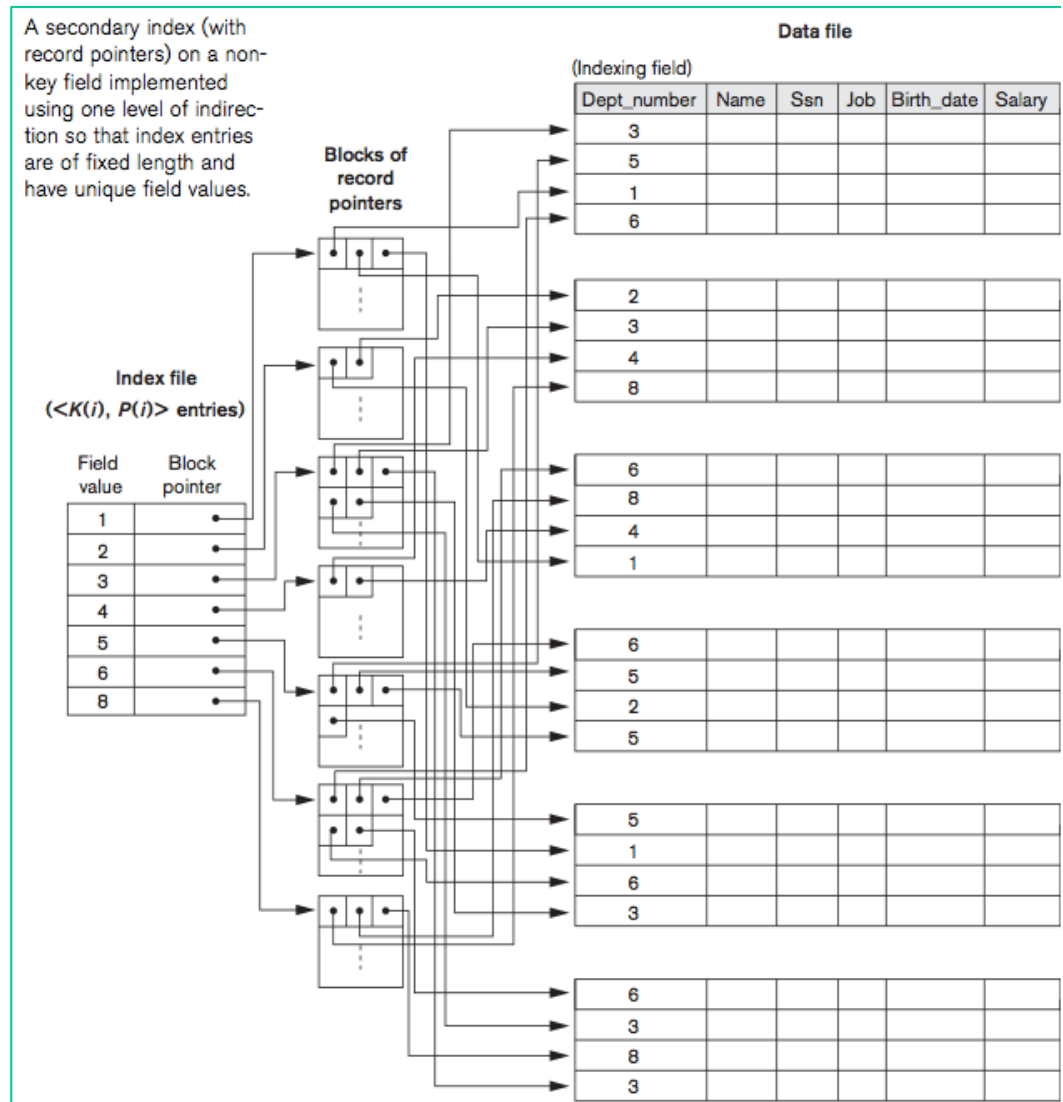
Primary Index - Exemplo



Clustered Index - Exemplos



Ponteiro para o bloco que contém o primeiro tuplo com o clustered index field

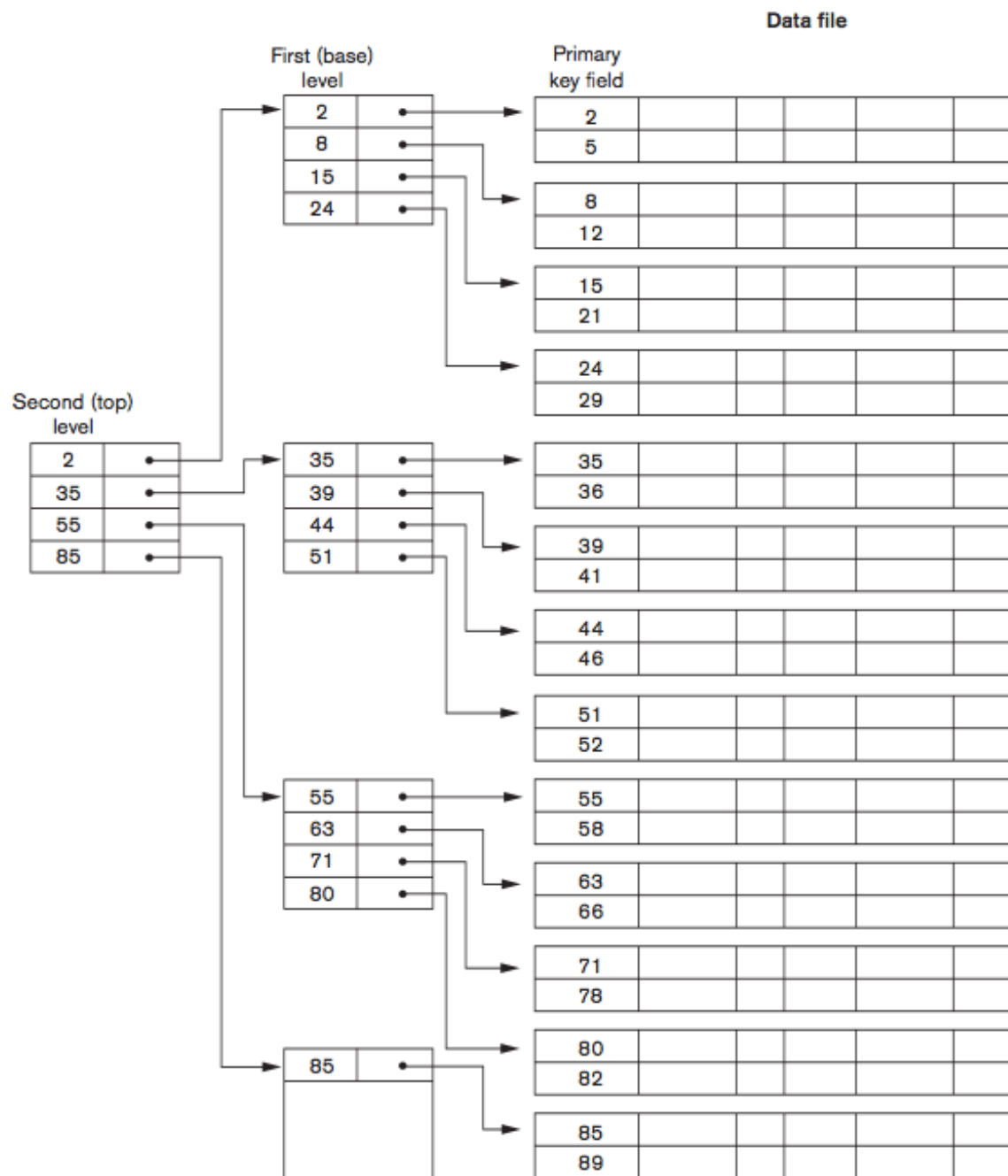


Indexing attribute with duplicate values

Multilevel Index

- 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: $\log_{f_o} b_i$
 - Fan out: f_o
 - A cada etapa do algoritmo estamos a reduzir o espaço de procura num factor f_o
 - Usualmente $f_o > 2$
- Estes índices são tipicamente implementados com estruturas em árvore balanceadas (equilibradas)
 - B-Tree

Multi-level Index - Exemplo



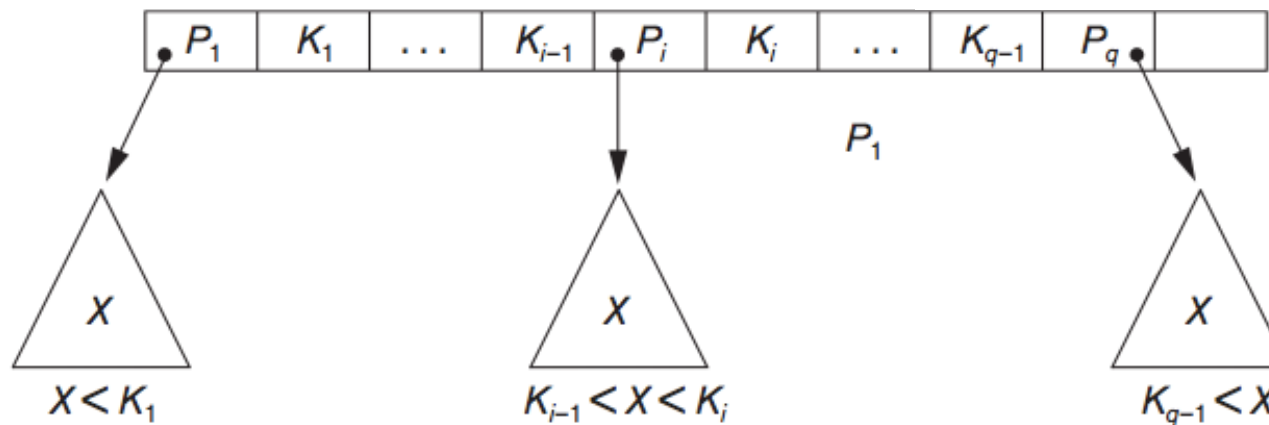
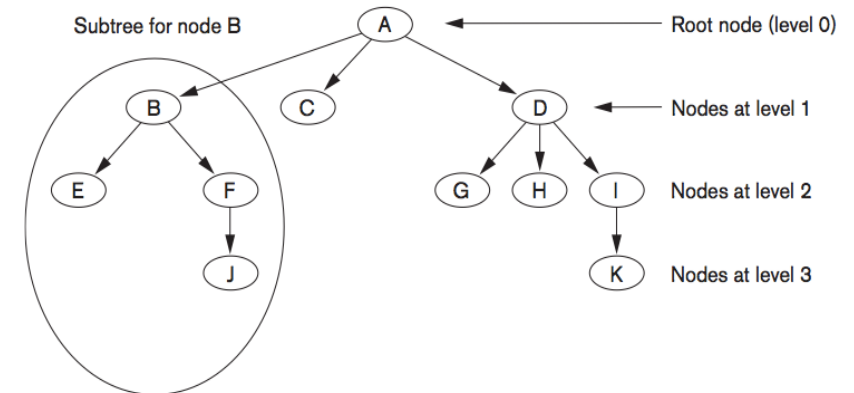
two-level
primary index

Árvore de Pesquisa

Search tree of order p

Cada nó contém, no máximo:

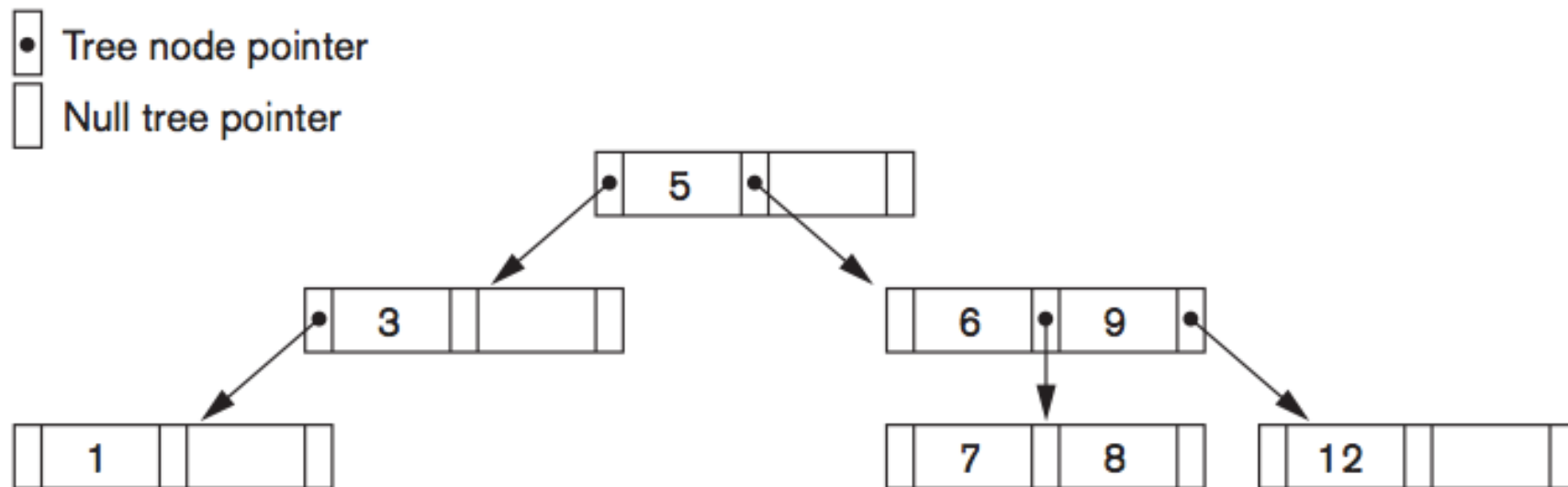
$p - 1$ valores e p ponteiros



- **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 $i = q$

Árvore de Pesquisa - Exemplo

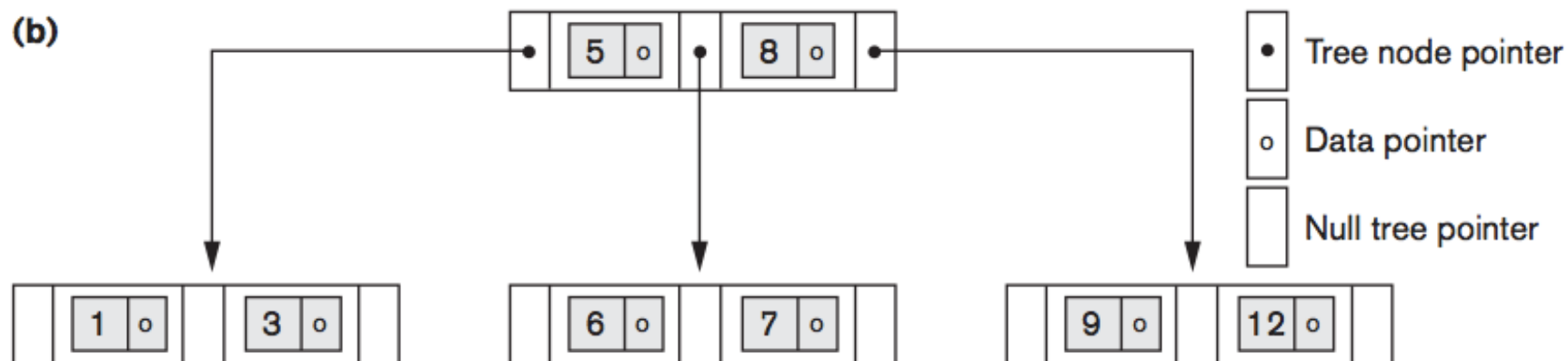
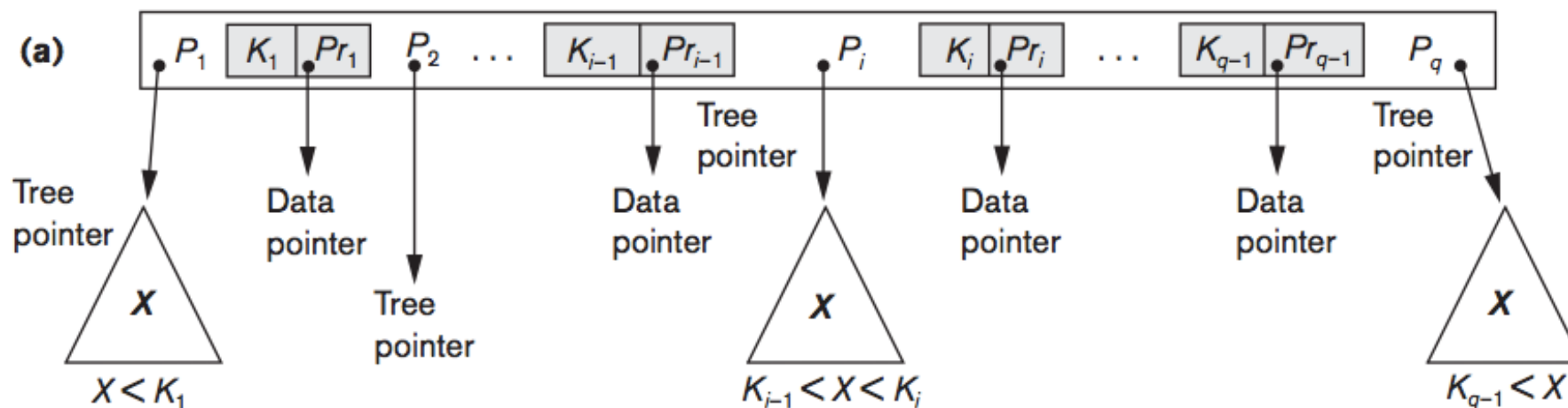
p = 3



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.

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.

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)
```

```
DROP INDEX index_name
```

```
-- Eliminar índice idxPName
```

```
DROP INDEX idxPName;
```

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

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 queries) para ajudar na decisão.

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 pesquisa¹⁹

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.

Seleção de Índices

Caso de Estudo

Cenário

- 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;                -- s constante
```

Q2: Procurar os atores de determinado filme

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

I: Inserir novo tuplo

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

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.

Tabela de Custos

Action	No Index	Star Index	Movie Index	Both Index
Q1	10	4	10	4
Q2	10	10	4	4
I	2	4	4	6
Cost Formula	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

P_1 e P_2 - fracção 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

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	$2 + 8p_1 + 8p_2$	$4 + 6p_2$	$4 + 6p_1$	$6 - 2p_1 - 2p_2$

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

Índices

SQL Server

SQL Server - Indexing

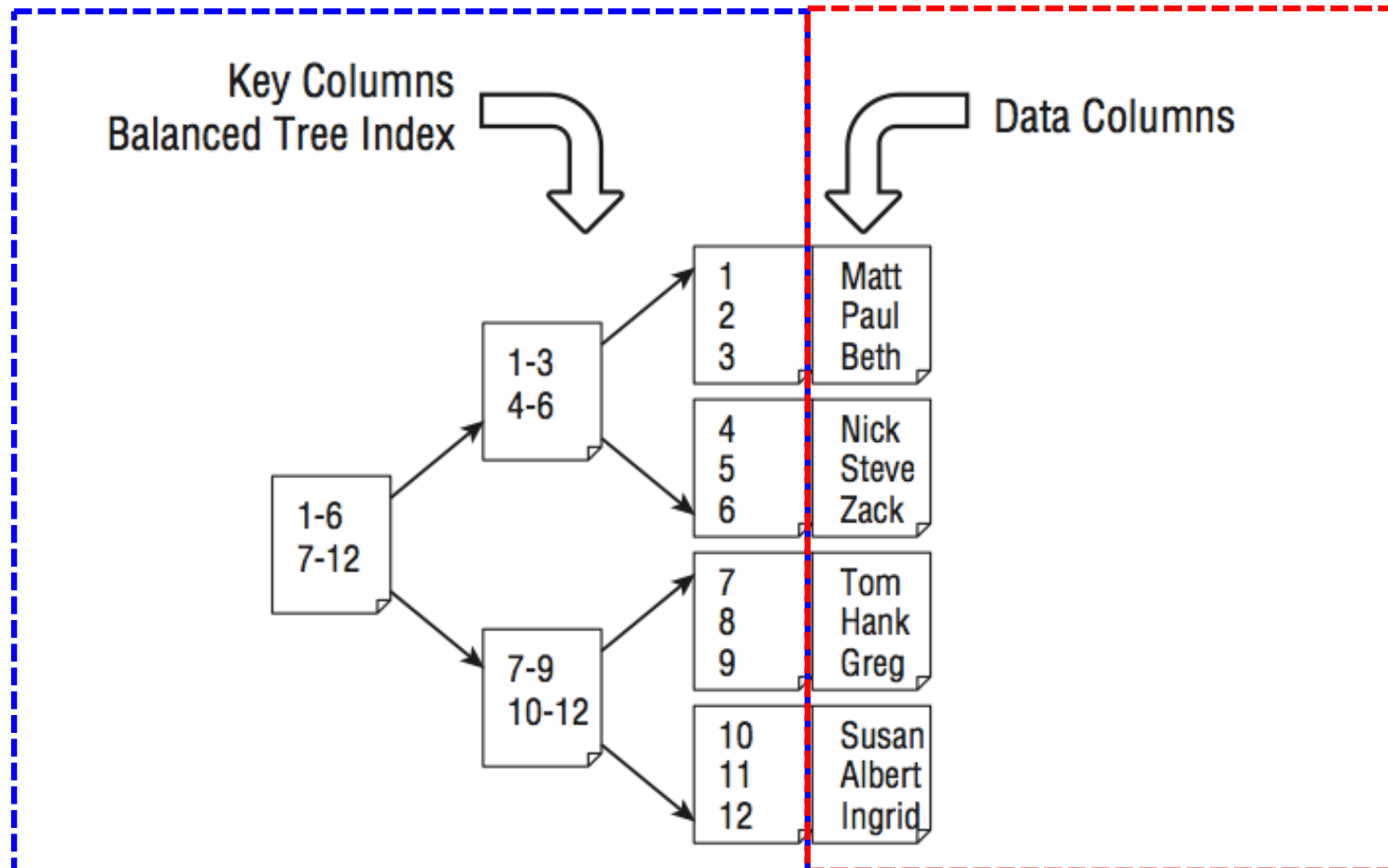
SQL Server tem dois tipos de índices*:

- clustered
 - Os nós folha contêm os próprios dados da relação
 - A tabela está ordenada pelo próprio índice
 - Só existe um por relação
 - Analogia: Agenda de contactos telefónicos
- non-clustered
 - 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

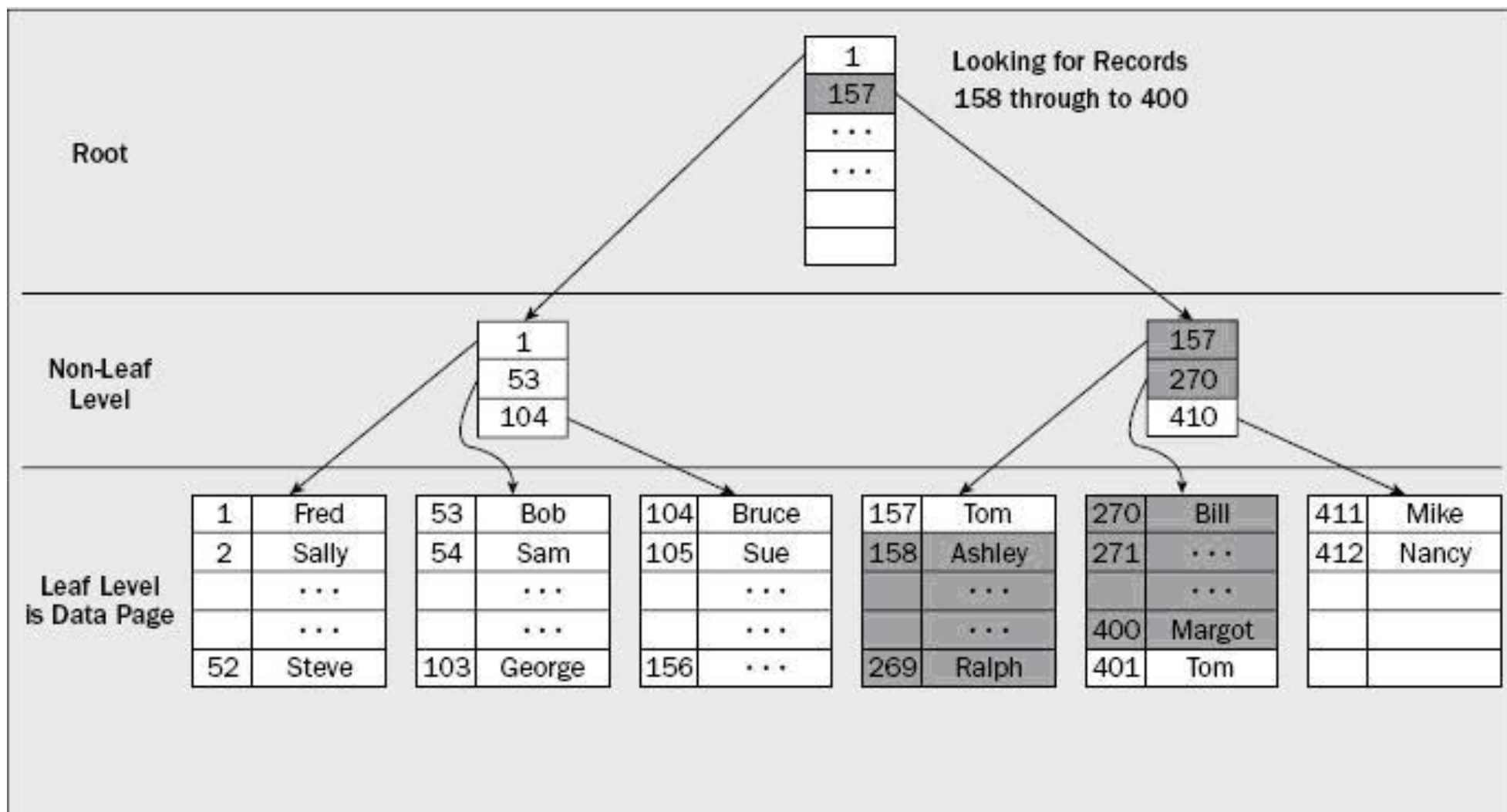
*ambos implementados com B-trees

SQL Server: Clustered index

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

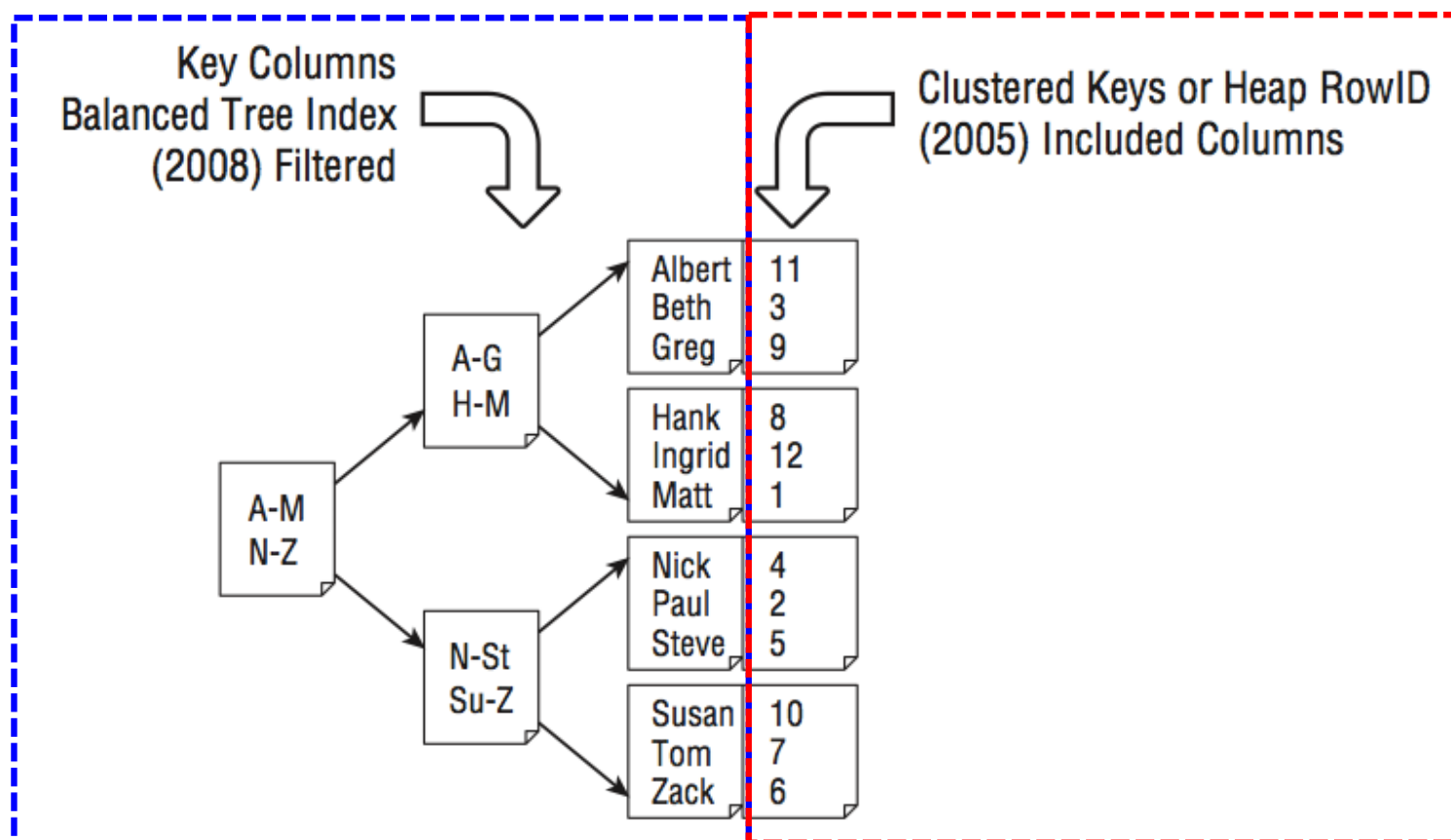


Clustered Index - Exemplo



SQL Server: Non-clustered index

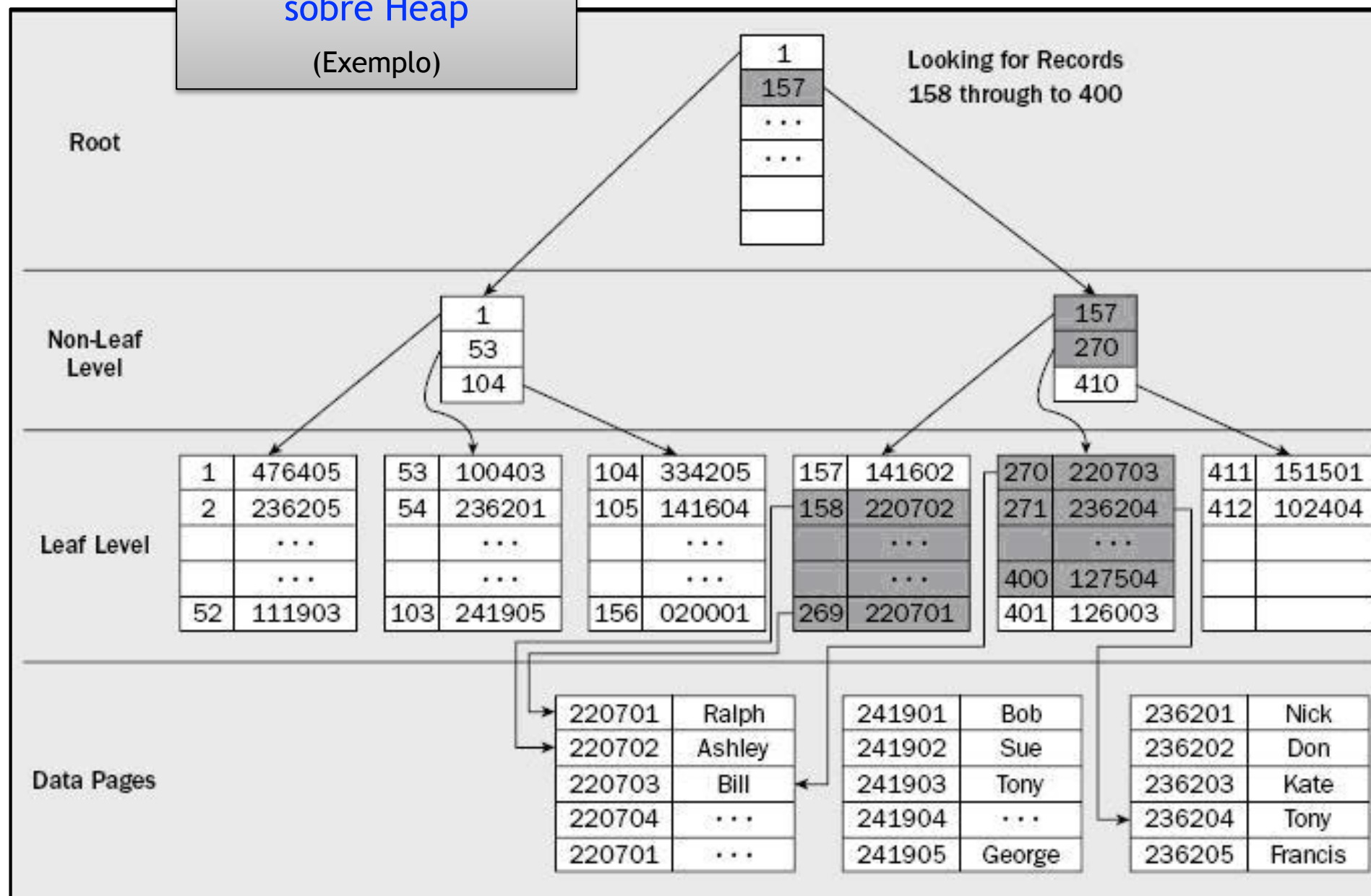
- As folhas contêm ponteiros para um **clustered index** ou **Heap RowID**.
 - Um tuplo de uma tabela non-clustered é identificado por um RowID que inclui o caminho completo (extent:página:row_offset)



Non-Clustered Indexes

sobre Heap

(Exemplo)



Select EmployeeID
who is FName like "T%"

Root

Allison	115
Fred	1
Mike	56
Ralph	74
Steve	52

Non-Leaf
Level

Allison	115
Bill	270
Charlie	23
Diane	361
Ernest	211

Steve	52
Tom	157
Zach	99

Leaf Level

Allison	115
Amy	27
...	
...	
Barbara	367

Bill	270
Bruce	104
...	
...	
Frank	171

Steve	52
Sue	105
...	
...	
Tim	102

Tom	157
Tony	209
...	
...	
Yolanda	

To Clustered
Index Point

1
157
...
...

1
53
104

157
270
410

1	Fred
2	Sally
...	
...	
52	Steve

53	Bob
54	Sam
...	
...	
102	Tim
103	George

157	Tom
158	Ashley
...	
...	
209	Tony
...	
269	Ralph

270	Bill
276	Russ
...	
...	
401	Tom

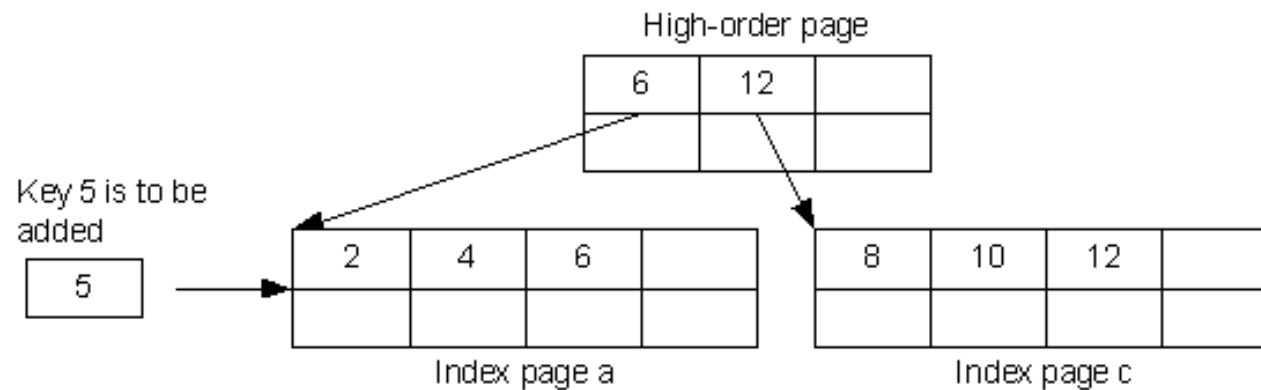
Non-Clustered Indexes
sobre
Clustered Table
(Exemplo)

B-Tree Page Split

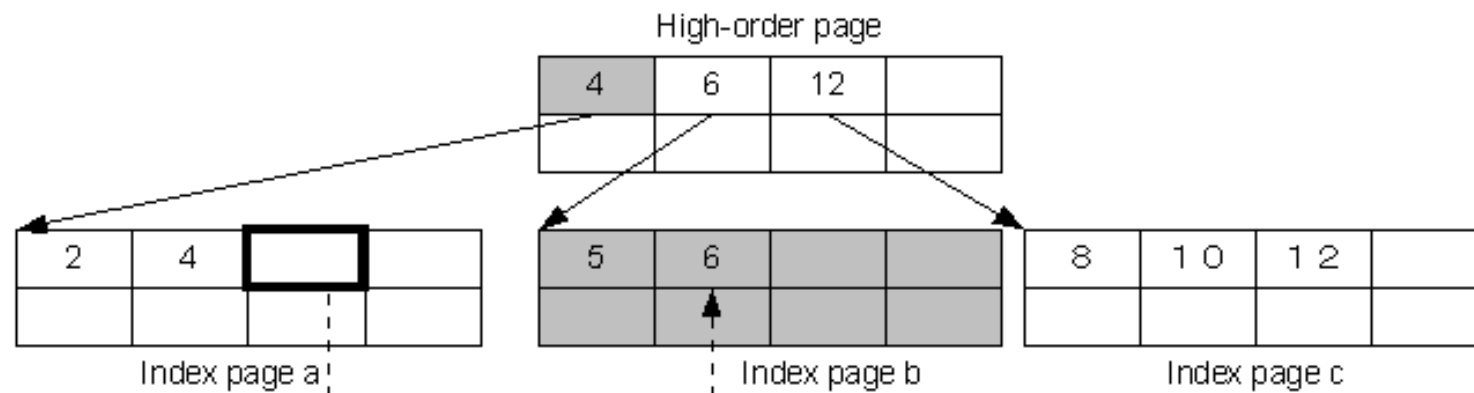
- Os **índices** da **B-Tree** devem manter-se **ordenados** pelo *key index*.
- Inserts, updates e deletes afectam os dados.
- O que acontece quando pretendemos fazer um insert e a página está cheia?
 - O SGBD divide a página cheia em duas (**page split**)
 - cria uma nova página
 - **copia** parte 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**.


B-Tree Page Split - Exemplo


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



2. Index page splitting



 : Locations where the index structure was changed by index page splitting.

 : Key that was moved to another page by index page splitting.
(Data is divided evenly between index pages a and b.)

Índices - Opções de Especialização

- Unique
 - A *index key* é única (não tem valores duplicados)
 - Por defeito, a criação de uma chave primária cria automaticamente um unique 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.

Índices - Opções de Especialização

- Filtered
 - 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

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';
```

-- Non-clustered with column include

```
CREATE INDEX ixGuideCovering ON dbo.Guide (LastName, FirstName)  
INCLUDE (Title);
```

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

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...

Escolha de um Non-Clustered Index

- São tipicamente utilizados para otimizar 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

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.
 - 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 IxOrderNumber 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
 - 65-85% se tivermos mais inserções no meio da B-Tree

Desfragmentação de Índices

- Processo de eliminação de “espaços vazios” resultantes:
 - Page Splits (1 page FULL 100% -> 2 pages ~50%)
 - Remoções de tuplos
- Regularmente devemos:
 1. 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”
 2. 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.

ou

ALTER INDEX ALL **ON** Frag **REBUILD** **WITH** (FILLFACTOR = 98)
 - reconstrói o índice completamente (equivalente a um DROP + CREATE)
 - podemos alterar as características do índice. Por exemplo, o fillfactor.

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):

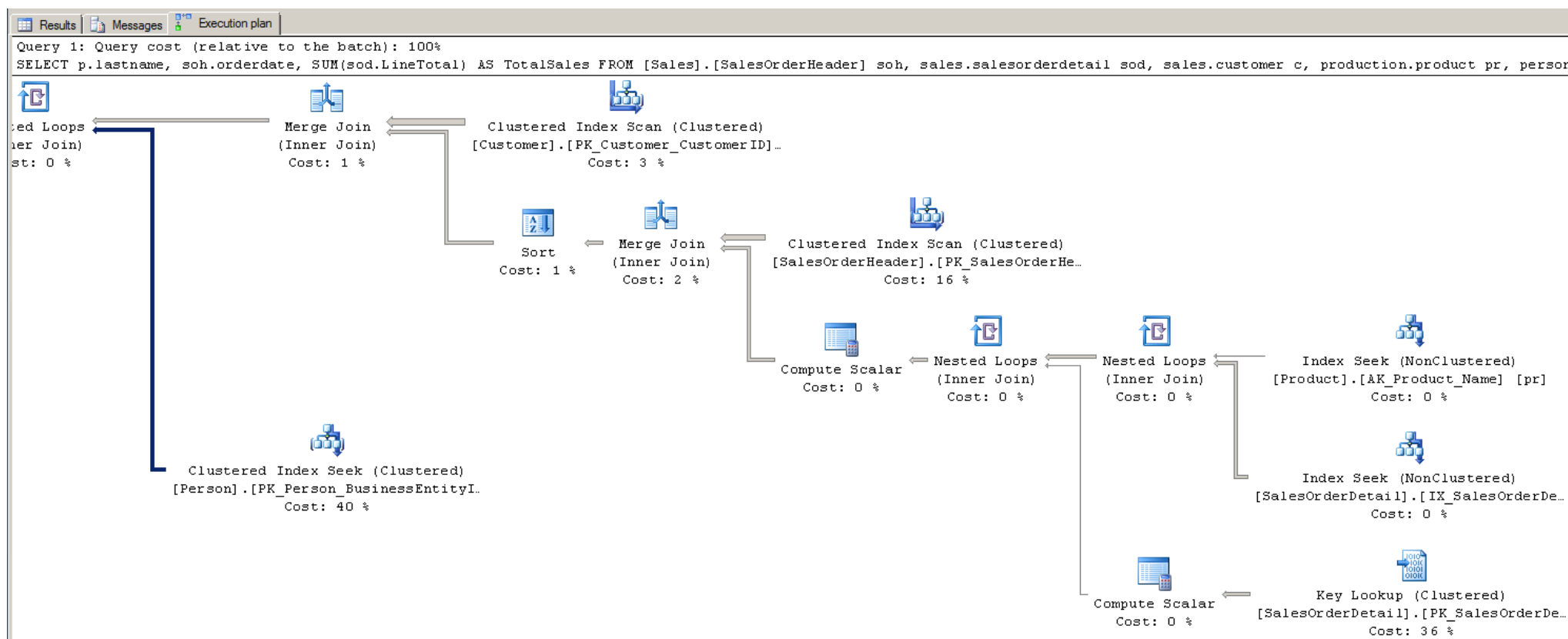
```
USE tempdb;  
ALTER INDEX PK_Frag ON Frag REORGANIZE;  
ALTER INDEX ix_col ON Frag REORGANIZE;
```

```
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
```

Problemas de Desempenho?

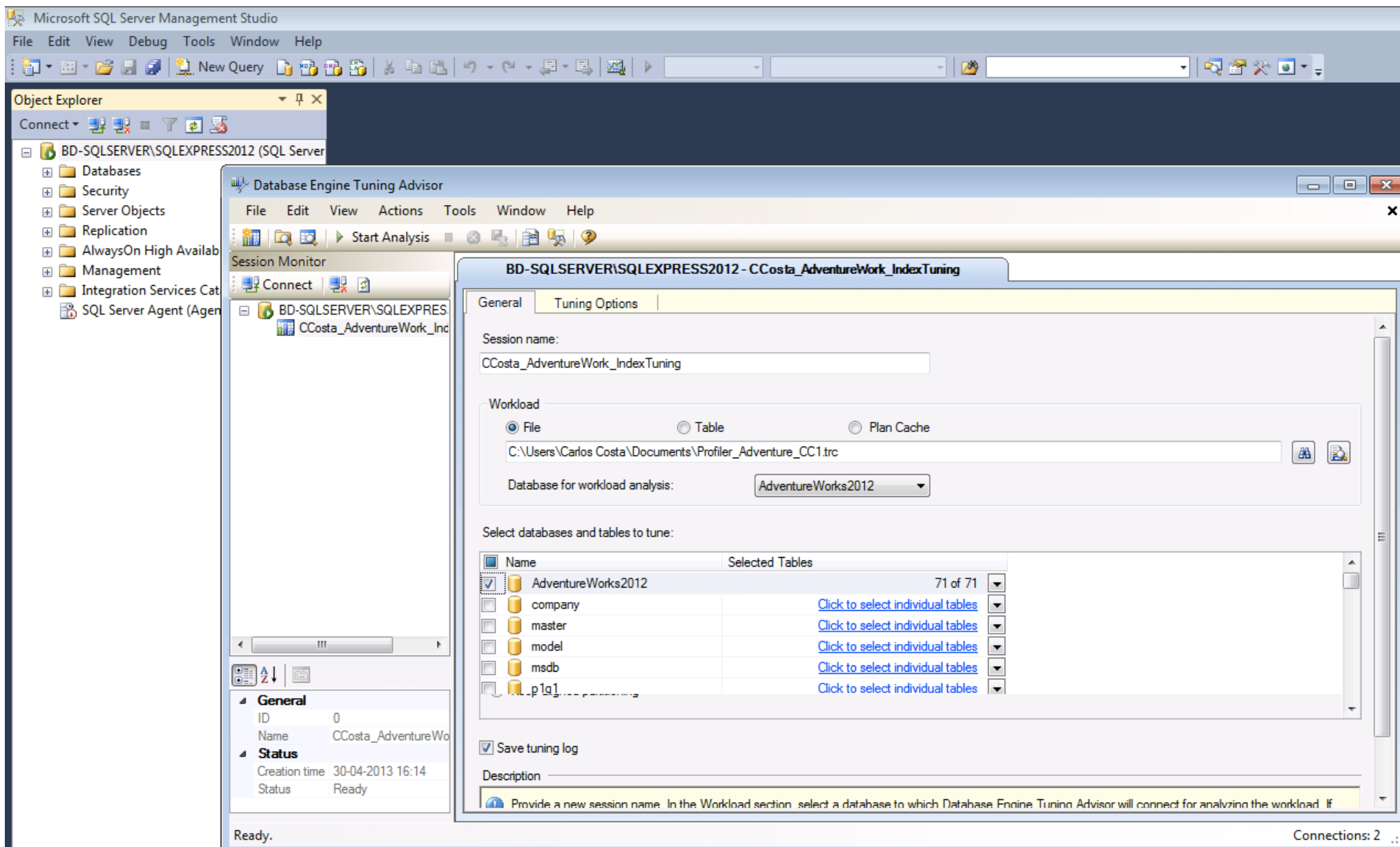
- Com uma query?
... consultar o “execution plan”



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**

Profiler + Engine Tuning Advisor



The screenshot displays the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the server structure for 'BD-SQLSERVER\SQLEXPRESS2012 (SQL Server)'. The main window is the 'Database Engine Tuning Advisor' for the session 'BD-SQLSERVER\SQLEXPRESS2012 - CCosta_AdventureWork_IndexTuning'.

The 'Tuning Options' tab is active, showing the following configuration:

- Session name:** CCosta_AdventureWork_IndexTuning
- Workload:**
 - ☒ File
 - ☐ Table
 - ☐ Plan Cache
 - File path: C:\Users\Carlos Costa\Documents\Profiler_Adventure_CC1.trc
 - Database for workload analysis: AdventureWorks2012
- Select databases and tables to tune:**

Name	Selected Tables
<input checked="" type="checkbox"/> AdventureWorks2012	71 of 71
<input type="checkbox"/> company	Click to select individual tables
<input type="checkbox"/> master	Click to select individual tables
<input type="checkbox"/> model	Click to select individual tables
<input type="checkbox"/> msdb	Click to select individual tables
<input type="checkbox"/> p1g1	Click to select individual tables
- ☒ Save tuning log
- Description:** Provide a new session name. In the Workload section, select a database to which Database Engine Tuning Advisor will connect for analyzing the workload. If

The bottom status bar indicates 'Ready.' and 'Connections: 2'.

Resumo

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

SQL Server

Tools

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'.
Updated      Rows      Rows      Steps      Density      Average
-----      -
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.