Arquitetura de SGBD Relacionais
— Métodos de Acesso Físico —

Caetano Traina Jr.

Grupo de Bases de Dados e Imagens
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
caetano@icmc.usp.br

18 de maio de 2011
São Carlos, SP - Brasil

Esta apresentação mostra os métodos de acesso físico usados para a execução de consultas em SGBD, como interpretar os resultados do comando *EXPLAIN QUERY* e como os métodos de de acesso exploram a estrutura de memória em disco.

Outline

Métodos de Acesso Físico em SGBD Relacionais

Introdução

- Todos os dados de uma base de dados são armazenados em disco, na estrutura <*tablespaces* – segmentos – *extents* – páginas>.
- Com isso, as diferentes formas de acesso lógico (acesso aos diferentes tipos de índices e organizações de relações) devem ser conformadas aos requisitos da estrutura de memória em disco e às formas de acesso que ela propicia.
- Conforme foi visto na aula sobre **"Organização da Memória em Disco"**, o acesso a cada registro físico envolve diversos tempos, bem maiores do que o necessário para acesso a dados na memória principal:
  - ☞ posicionar as cabeças,
  - ☞ latência rotacional,
  - ☞ taxa de transferência.

Métodos de Acesso Físico em SGBD Relacionais

Introdução

- Nesta apresentação, usaremos estimativas de custo de acesso, seguindo as mesmas considerações que o **Otimizador do Plano de Acesso** usa para fazer as previsões de custo.
- Para estimar o custo de acesso, não é necessário saber o tempo real, absoluto, de cada operação, mas apenas um tempo relativo.
- ☞ se é conhecido que uma operação é mais lenta que outra, então um plano que use a operação mais lenta será mais lento que outro cuja única mudança é a troca dessas duas operações.
- Portanto, o acesso aos x bytes de um *extent* usando a leitura de um *extent* inteiro é mais rápido do que o acesso a esses mesmos x bytes lendo página por página.

Tipos de acesso físico

- A proporção real de quanto o acesso ao *extent* inteiro é mais rápido depende de inúmeros fatores, tais como características físicas dos discos, distribuição dos *extents* no disco, dos dados em vários discos, quantas páginas por *extent*, etc.
- Considera-se para efeito de estimativa de custo, que o acesso página a página é 10 vezes mais lento que o acesso *extent* a *extent*.
- Assume-se também, para mera intuição dos tempos envolvidos, que se pode fazer, em média, acesso a 100 páginas individuais por segundo.
- Usaremos a seguinte notação para indicar números de acessos a disco:

Leitura página a página: **Aleatório** RxxxR (Read xxx R andom)
Leitura *extent* a *extent*: **Sequencial** RxxxS (Read xxx S equential)

Tipos de acesso físico

- Também é possível que algumas páginas espalhadas no disco tenham seu acesso previsto, e solicitado para leitura prévia (*page prefetch*) no *buffer* do disco.
- Esse acesso será chamado de **acesso previsível** e usaremos a notação: RxxxT (Read xxx T racked).
- O acesso previsível é mais rápido do que o aleatório, porque permite otimizar a sequência de movimentos da cabeça do disco, mas não é tão rápido quanto o acesso sequencial.
- Considera-se que o acesso previsível é em média 3 vezes mais rápido do que o acesso aleatório.

Tipos de acesso físico

Exemplo

- Para ilustrar, considere-se uma página de 2KB, e que estão sendo lidos 1MB de dados.
- Portanto, são $\frac{1.000KB}{2KB} = 500$ páginas.
- Elas podem ser lidas
  - ☞ Aleatoriamente: R500R a 100 páginas por segundo: $\frac{500}{100} = 5$ segundos.
  - ☞ Previsivelmente: R500T a 3*100 páginas por segundo: $\frac{500}{300} = 1,7$ segundos.
  - ☞ Sequencialmente: R500S a 10*100 páginas por segundo: $\frac{500}{1.000} = 0,5$ segundos.
- Esses tempos são apenas estimativas.

Tipos de acesso aos objetos

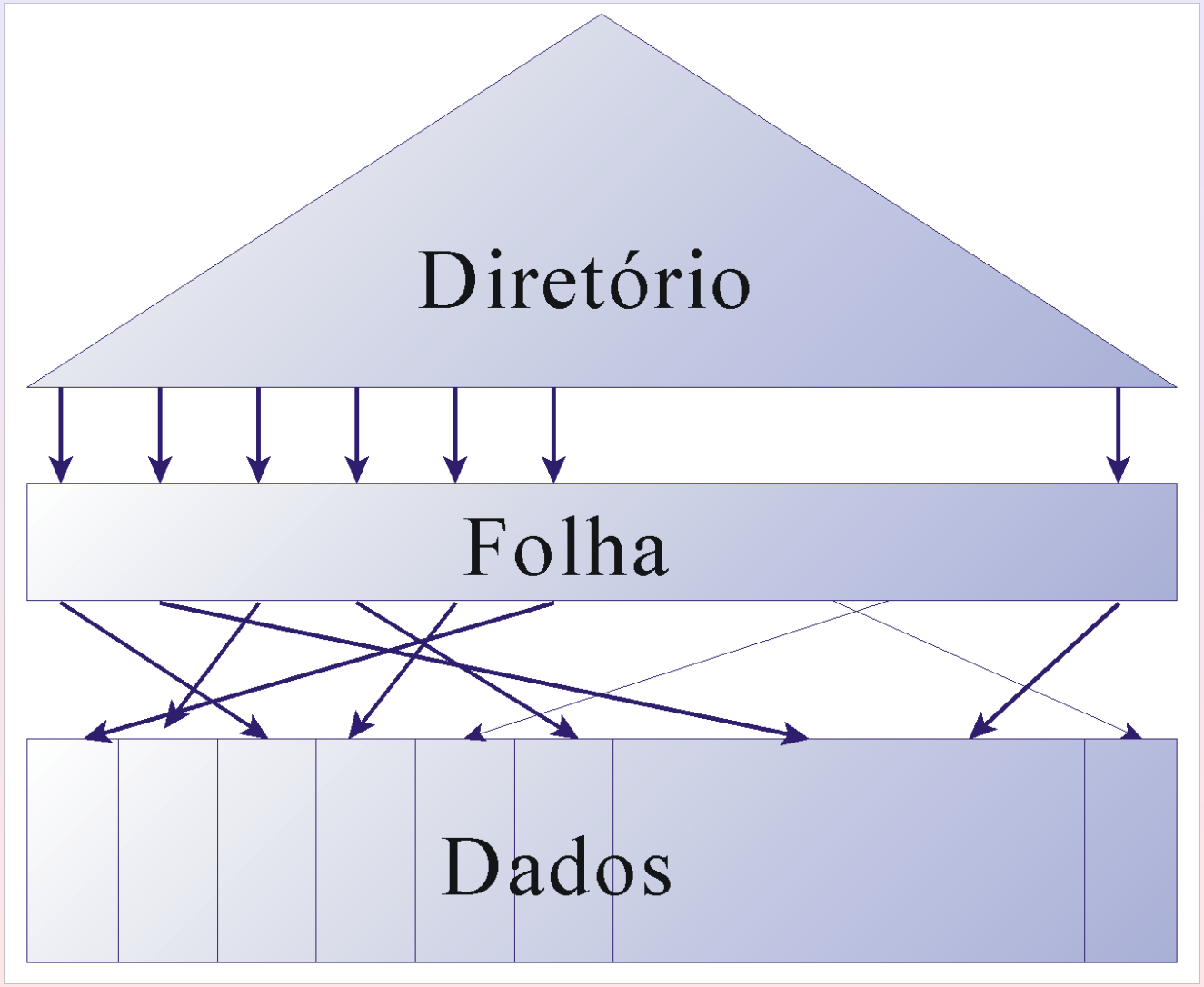
- O SGBD acessa:
  - Segmentos de dados,
  - Segmentos de índices.
- Para a maioria das consultas, os dados estão armazenados em uma estrutura em *heap*. Eles podem ser acessados:
  - em busca sequencial: o acesso é feito sequencialmente.
  - em busca indexada com ponteiro direto de um índice: o acesso é feito aleatoriamente.
  - em busca indexada a partir de uma *Rid-list*: o acesso é feito previsivelmente.

Tipos de acesso aos objetos

- Para o acesso aos índices, consideremos as formas como os índices se organizam:
  - Índice "normal" (índice + heap)
  - Índice Secundário
  - Índice Primário

Tipos de acesso aos objetos

Acesso indexado por B-tree



Tipos de acesso aos objetos

Acesso indexado por B-tree

- Diretório:

  A sequência de acesso não pode ser determinada a priori: R($H-1$)R

- Folha:

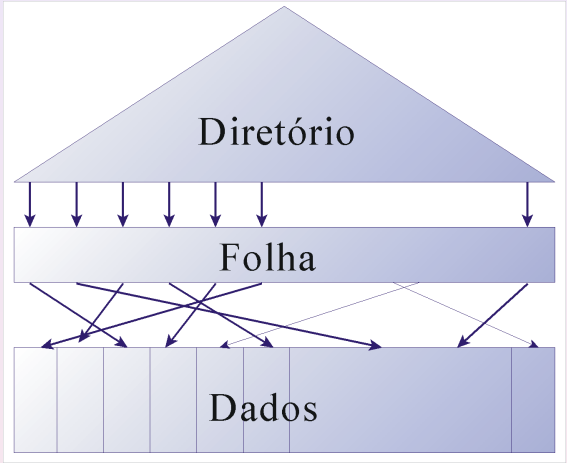
  O acesso pode seguir a sequência das chaves.

  Caso seja chave única: R1R

  Caso sejam múltiplas chaves: R(*NRid-list*)S

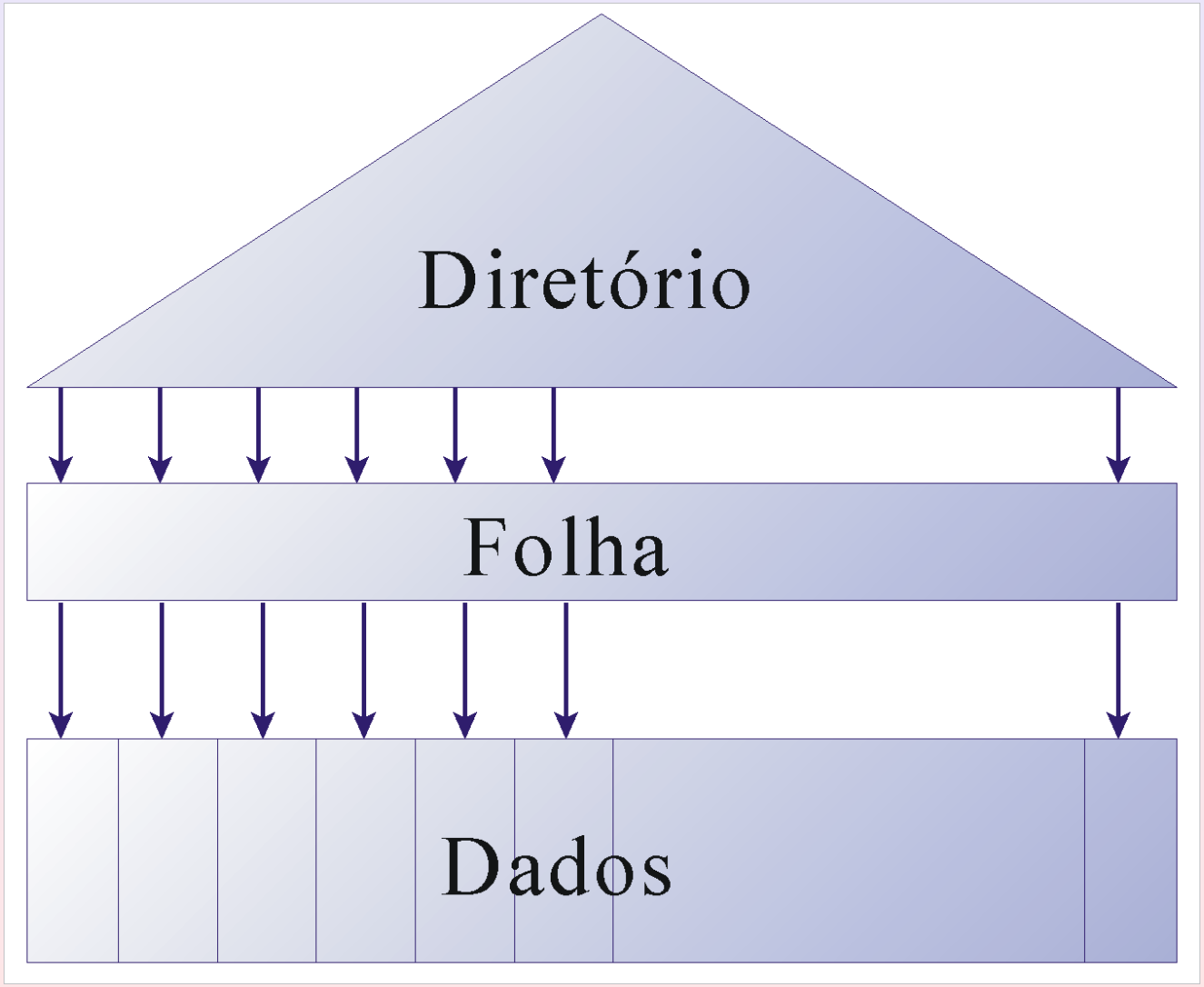
- Dados:

  A sequência de acesso é dada a priori pela *Rid-list* nas folhas: R(*NDados*)T



Tipos de acesso aos objetos

Acesso indexado por Cluster de B-tree



Tipos de acesso aos objetos

Acesso indexado por Cluster de B-tree

- Diretório:

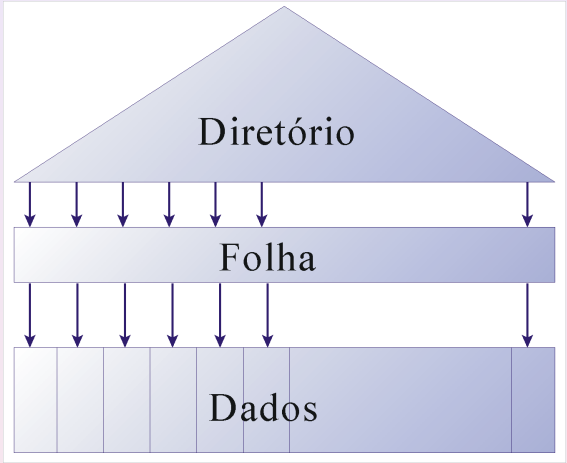
  A sequência de acesso não pode ser determinada a priori: R($H-1$)R
- Folha:

  O acesso pode seguir a sequência das chaves.

  Caso seja chave única: R1R

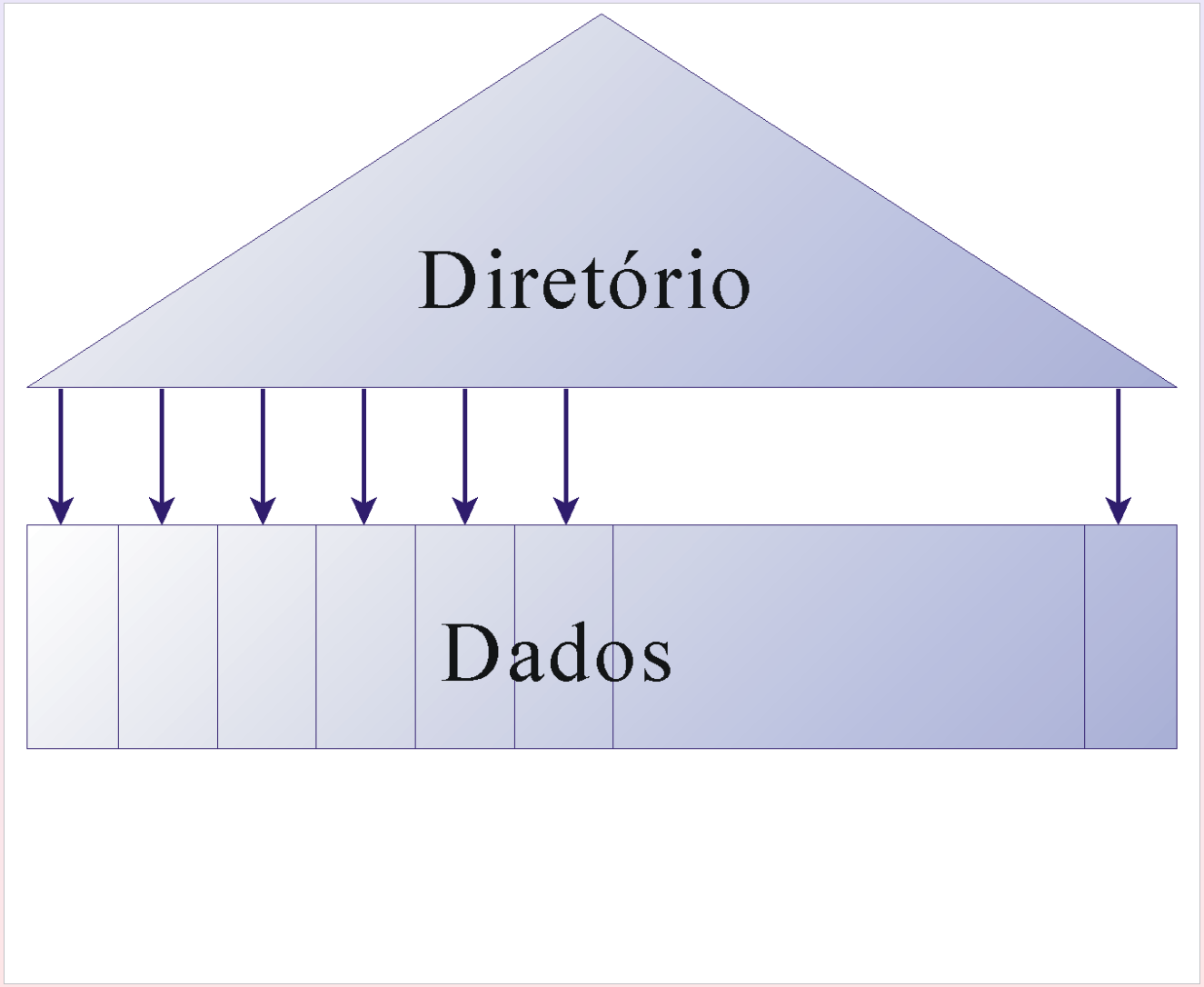
  Caso sejam múltiplas chaves: R(*NRid-list*)S
- Dados:

  O acesso pode seguir a sequência das chaves nas folhas: R(*NDados*)S



Tipos de acesso aos objetos

Acesso em Índice Primário de B-tree



Tipos de acesso aos objetos

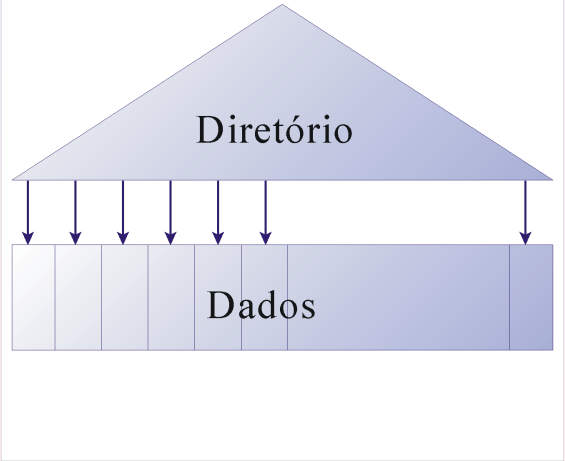
Acesso em Índice Primário de B-tree

- Diretório:

  A sequência de acesso não pode ser determinada a priori: R($H-1$)R

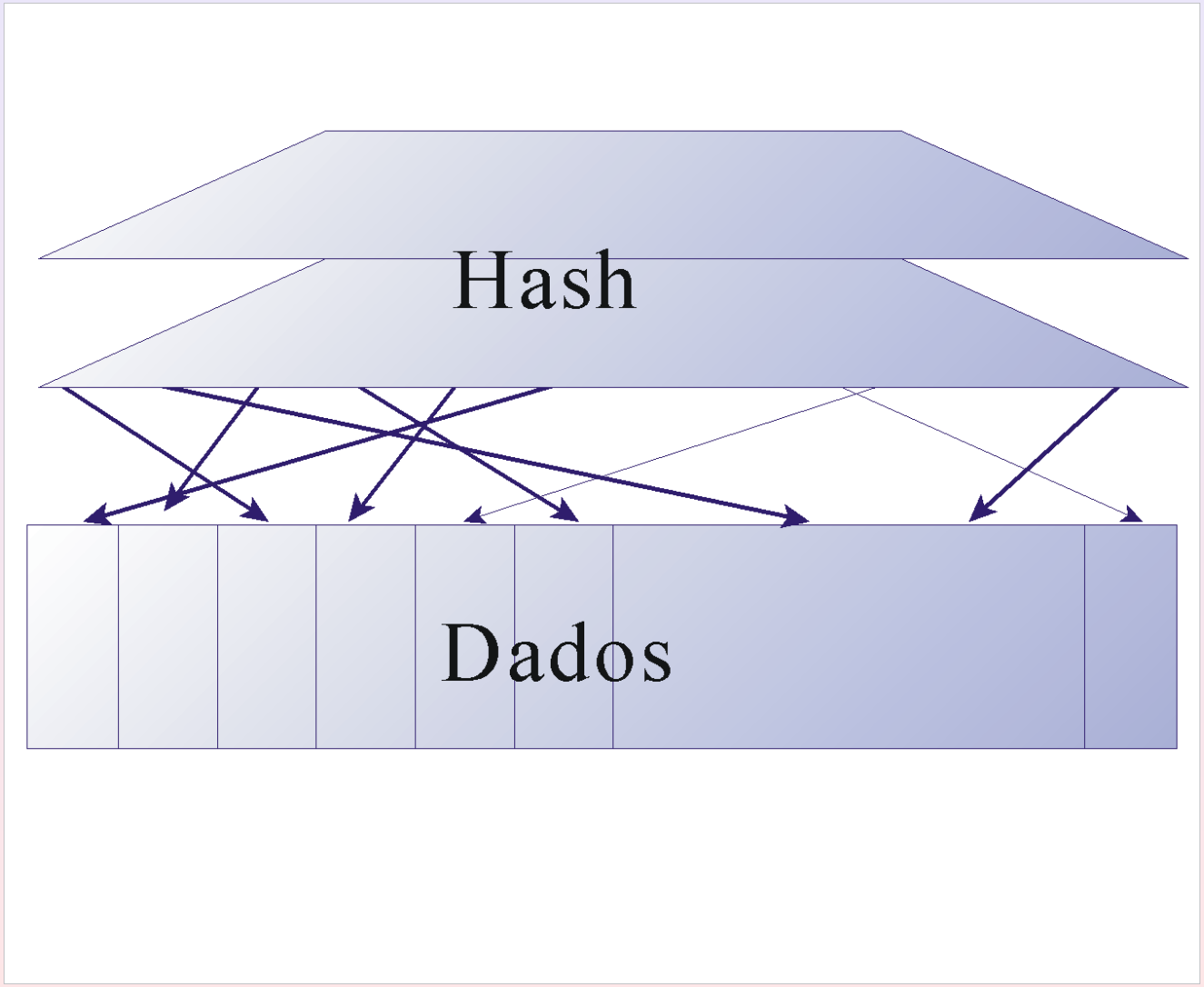
- Dados:

  O acesso pode seguir a sequência das chaves nas folhas: R($NDados$)S



Tipos de acesso aos objetos

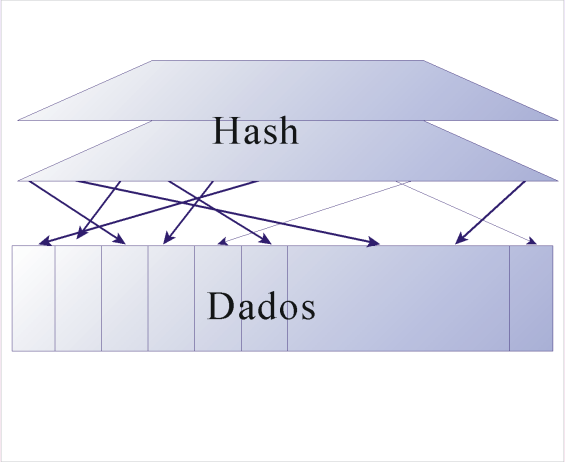
Acesso indexado por Hash



Tipos de acesso aos objetos

Acesso indexado por Hash

- Diretório de Hash:
  Se não houver colisão, o acesso é direto ao primeiro *extent*: R1R
  Se houver colisão, o acesso é aleatório, página a página, pela quantidade necessária de *links* de continuação: R(n)R
- Dados:
  Se não houver colisão, o acesso é direto ao primeiro *extent*: R1S
  Se houver colisão, o acesso é sequencial, *extent* a *extent* que armazena as páginas de continuação: R(n)S



Tipos de acesso aos objetos

Acesso indexado por Hash

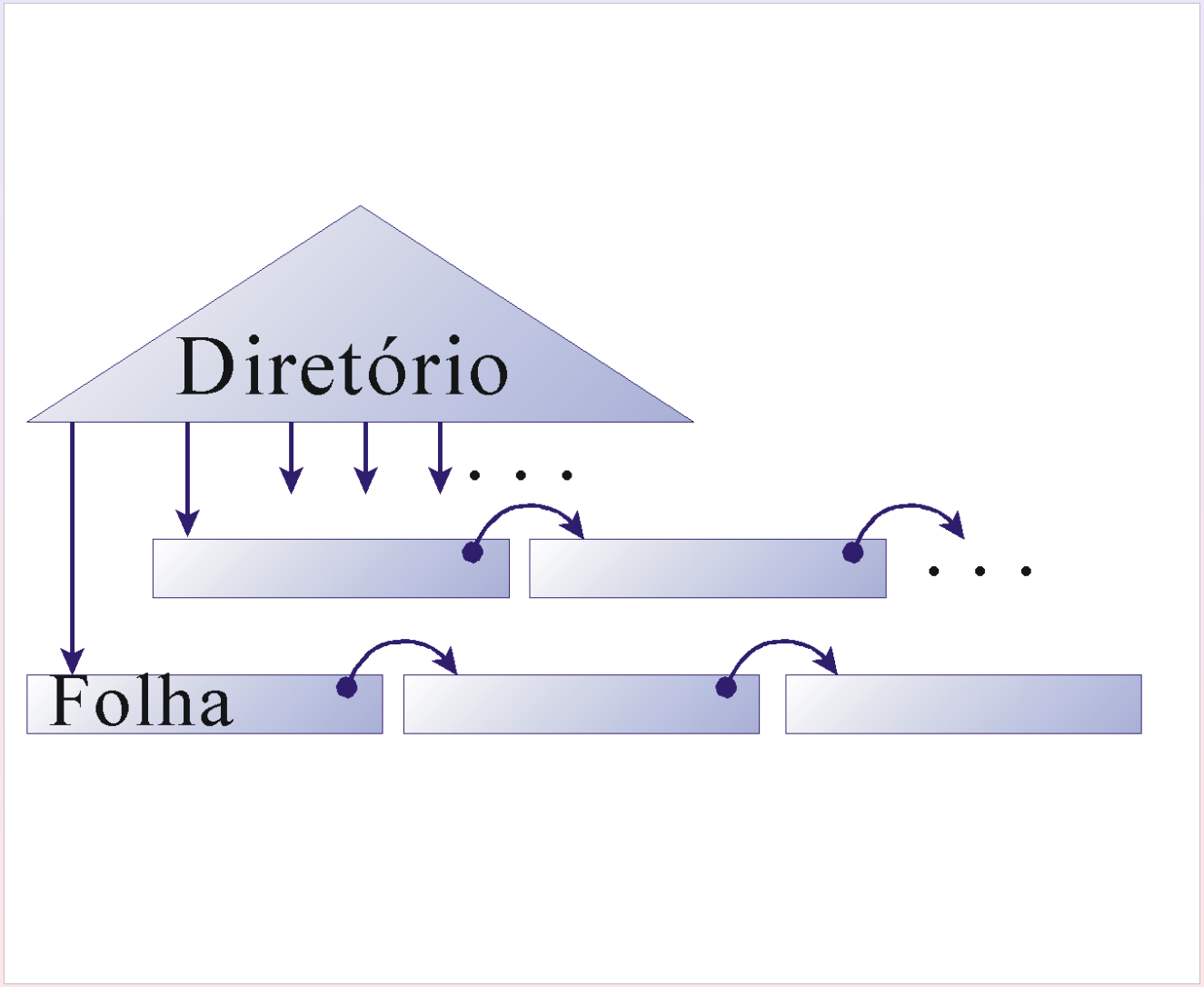
- No caso do Oracle, a estrutura Hash sempre gera um "**Cluster Hash**", portanto os dados são armazenados junto com a o próprio índice. Assim, o acesso reduz-se a:

  Se não houver colisão, o acesso é direto ao *extent* onde está o dado: R1S

  Se houver colisão, o acesso é sequencial, *extent* a *extent*: R(n)S

Tipos de acesso aos objetos

Acesso indexado por Bitmap



Tipos de acesso aos objetos

Acesso indexado por Bitmap

- Diretório do Bitmap:

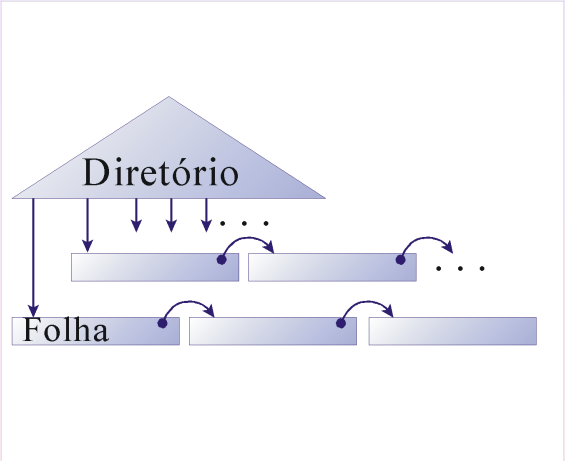
  O diretório é uma árvore B-tree, portanto é acessada com um acesso aleatório por nível

  R(H)R

- Dados:

  Os dados do índice são lidos em sequência

  R($Size$)S (Lembrando que é um bit por tupla.)



- O acesso a índices baseados em **arquivos invertidos** segue exatamente a mesma estrutura, a menos de que nas folhas são armazenadas as *Rid-lists* ao invés dos *bitmaps*

Tipos de acesso aos objetos

Exemplo

- Por exemplo, seja a relação de alunos, já definida anteriormente em Oracle como:

```
CREATE TABLE Alunos (
    NUSP CHAR(10) PRIMARY KEY,
    Nome CHAR(40),
    Idade DECIMAL(3),
    Cidade CHAR(30) )
    STORAGE INITIAL 20K NEXT 20K
    PCTFREE 10 PCTUSED 70
```

- Vamos considerar que existam 80.000 alunos,
- e também que o tamanho da página seja de 2KBytes, assumindo 2.000 bytes para dados.
- Com isso, cada *extent*, inicial e de continuação, ocupa 10 páginas.

Índices Primários

Cálculo de parâmetros da relação usada no exemplo

- Cada tupla ocupa $10 + 40 + \frac{3}{2} + 30 = 72$ bytes, mais o ponteiro do diretório da página são 74 bytes por tupla.
- Com a indicação `PCTFREE 10 PCTUSED 70`, cada página em disco fica em média $\frac{(100-\texttt{PCTFREE})+\texttt{PCTUSED}}{2} = \frac{(100-10)+70}{2} = 80\%$ ocupada.
- Portanto cada página armazena em média $\left\lfloor 0.80 \cdot \frac{2.000}{74} \right\rfloor = 21,62 = 21$ tuplas.
- A tabela `Alunos` ocupa $\left\lceil \frac{80.000}{21} \right\rceil = 3.809,52 = 3.810$ páginas.
- Como cada *extent* contém 10 páginas, a tabela requer $\left\lceil \frac{3.810}{10} \right\rceil = 381$ *extents*.
- A chave primária ocupa $10 + 2 = 12$ bytes (2 bytes do diretório da página).
- Como o índice não teve parâmetros próprios indicados, não existe reserva de espaço e a taxa de ocupação é a média teórica de 0,71%.
- Então, cada nó folha armazena $\left\lfloor 0.71 \cdot \frac{2.000-2*4}{12+6} \right\rfloor = 78,57 = 78$ chaves, e existirão $\left\lceil \frac{80.000}{78} \right\rceil = 1.025,64 = 1.026$ páginas.
- Então, cada nó diretório armazena $\left\lfloor 0.71 \cdot \frac{2.000-4}{12+4} \right\rfloor = 88,57 = 88$ chaves, e existirão $\left\lceil \frac{1.025}{88} \right\rceil = 11,65 = 12$ páginas no segundo nível, e a árvore terá $H = 3$ níveis.

Índices Primários

Estatísticas para a relação usada no exemplo

- Resumindo, temos como estatísticas:

Métricas para Tabela:

Nome	RowSIZE	CARD	MaxCARD	NPags	NExtents
Alunos	74	80.000	80.000	3.810	381
...		...			

Métricas para Atributos:

Tabela	Atrib	ColSIZE	ColCARD	Distr	Low2Key	high2Key
Alunos	NUSP	10	80.000	2	'0011223344'	'9988776655'
Alunos	Nome	40	79.500	2	'Abrão Abraão'	'Zuleica Penultima'
Alunos	Idade	2	50	2	17	80
Alunos	Cidade	30	700	2	'Adamantina'	'Zzy'
	...		...			

Métricas para Índices:

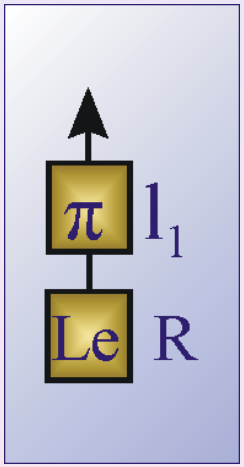
Tabela	Índice	Type	H	NLeafs	Key1Card	KeyFullCard	ClusterRatio
Alunos	AlunosPK	B-tree	3	1.026	80.000	80.000	0
	...				...		

Tipos de acesso aos objetos

Exemplo – Consulta 1 – Acesso à relação inteira.

- Seja a seguinte consulta

```
SELECT NUSP, Nome FROM Alunos
```



- Essa consulta gera a seguinte árvore de comandos:
- A leitura e a projeção podem ser executadas em uma operação só:

  Acessa sequencialmente todas as páginas que compõem a relação: R3810S

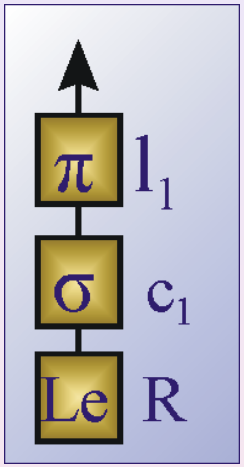
- Usando a estimativa de serem lidas 1.000 páginas sequencialmente por segundo, esse comando irá demorar $\frac{3.810}{1.000} = 3,18$ segundos.

Tipos de acesso aos objetos

Exemplo – Consulta 2 – Acesso pontual a uma tupla usando índice

- Seja a seguinte consulta

```
SELECT * FROM Alunos
    WHERE NUSP=1231231234
```



- Essa consulta gera a seguinte árvore de comandos:

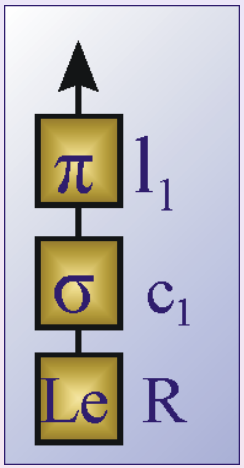
- A leitura e a seleção podem ser executadas em uma operação só:
  Acessa aleatoriamente cada nível da árvore ($H = 3$) e a página de dados na relação: R4R
- Usando a estimativa de serem lidas 100 páginas sequencialmente por segundo, esse comando irá demorar $\frac{4}{100} = 0,04$ segundos.

Tipos de acesso aos objetos

Exemplo – Consulta 3 – Acesso a uma faixa de tuplas usando índice

- Seja a seguinte consulta

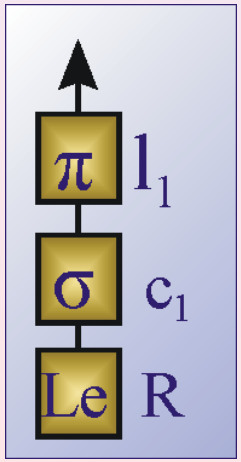
```
SELECT * FROM Alunos
    WHERE NUSP BETWEEN 2000000000 AND
        3000000000
```



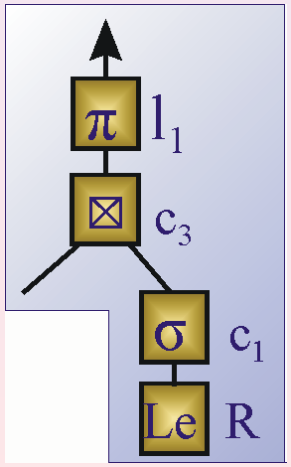
  - Essa consulta gera a mesma árvore de comandos:
- A leitura e a seleção podem ser executadas em uma operação só:
  mas o cálculo da seletividade, indica que devem ser acessadas aproximadamente $\frac{3000000000-2000000000}{9988776655-0011223344} = 0,10$, 10% das tuplas.
- Isso significa ler aleatoriamente uma página por nível da B-tree até obter a primeira folha, e dai ler sequencialmente 10% das folhas (1026/10): R3R + R103S ;
- A partir dai, cada tupla deve ser lida por acesso previsível: 10% de 80.000 corresponde a 8.000 tuplas: R8.000T .
- A estimativa de tempo é então: $\frac{3}{100} + \frac{103}{1.000} + \frac{8.000}{300} = 0,03 + 0,103 + 26,67 = 26,80$ segundos.

Métodos de Acesso Físico

- Além da otimização lógica, apoiada na Álgebra Relacional, é necessário fazer a otimização do método de acesso físico a ser executado para obter a resposta.
- Um método de acesso "empacota" um ou mais operadores algébricos.
- Por exemplo, é possível executar os operadores Le R seguido de σ c_1 e de π l_1 para recuperar todas as tuplas necessárias numa única operação de acesso aos discos.
- Existem basicamente dois tipos de métodos de acesso:



Para operações Unárias



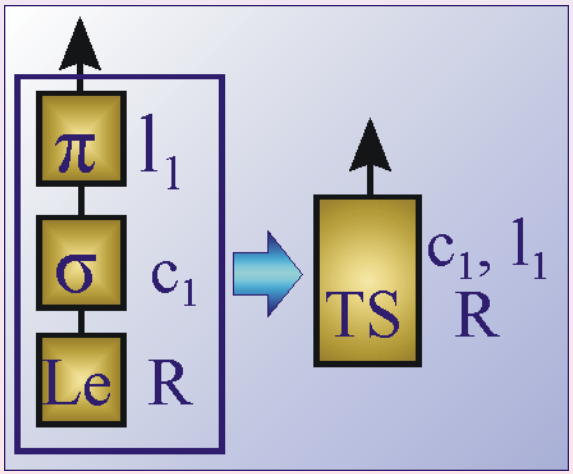
Para operações Binárias

Métodos de Acesso Físico

- Métodos de Acesso para operações unárias
  - *Table Scan*
  - *Index Scan*
  - *Clustered Matching Index Scan*
  - *Index-only Scan*
  - *Multi-Index Scan*
  - *Bitmap Heap Scan*
- Operadores de apoio
  - *Multi-Index Merger*
  - *Bitmap Sorter*
  - *Hash Table*
- Métodos de Acesso para operações binárias
  - *Neasted-loop Join*
  - *Hibrid Join*
  - *Merge Join*
  - *Hash Join*

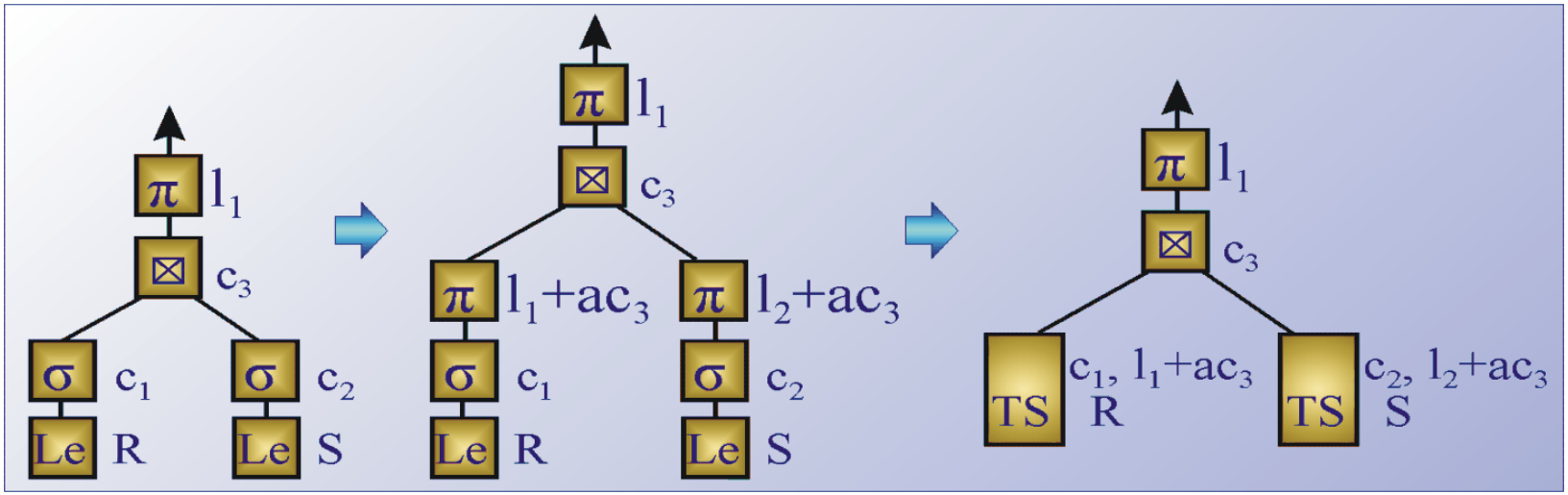
Métodos de Acesso para operações unárias

- Um Método de acesso para operações unárias empacota uma sequência de operadores Le R, σ c_1 e π l_1. Por exemplo, seja o método de acesso físico *Table Scan* (a ser estudado a seguir):



Métodos de Acesso para operações unárias

- Note-te se que a escolha por um plano lógico 'ótimo' pode levar em conta o uso que será feito das operações de acesso físico.
- Por exemplo, antecipar a projeção apenas dos atributos que efetivamente serão usados é sempre uma operação útil, dado que todos os operadores de acesso físico já provêm naturalmente a capacidade de projeção.



Métodos de Acesso para operações unárias

Table Scan

- O método de acesso **Table Scan** é o método básico para acesso a qualquer relação.
  - Quer dizer, mesmo que uma consulta não consiga usar nenhum outro método, o *table scan* sempre pode ser usado.
  - Ele não depende do uso de nenhum índice.
- O *table scan*:

  ☞ **lê** sucessivamente todos os *extents* que formam uma relação,

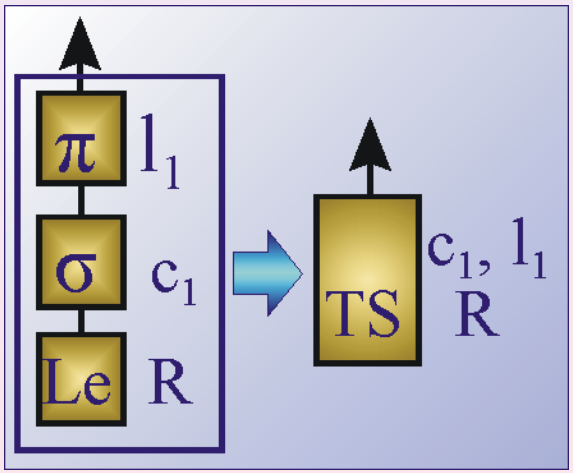
  ☞ aplica o critério combinado dos **operadores de seleção** "empacotados" para todas as tuplas de cada *extent*,

  ☞ e escreve no *buffer* de saída as tuplas projetadas segundo a lista de tuplas do **operador de projeção** "empacotado".
- **O custo** do *Table Scan* é a leitura sequencial das *NPags* que armazenam a relação: R(*NPags*)S .

Métodos de Acesso para operações unárias

Table Scan

- O **Table Scan** empacota uma sequência de operadores Le R, σ c_1 e π l_1:



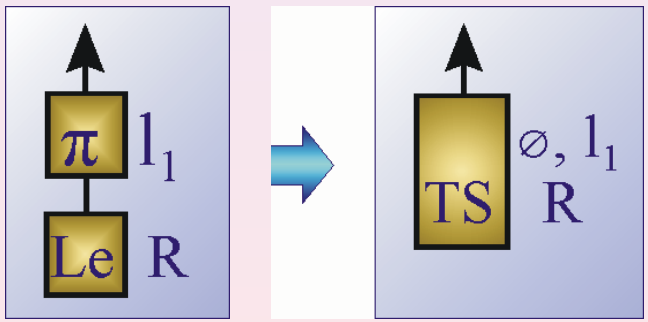
Métodos de Acesso para operações unárias

Table Scan – Exemplo Consulta 1

- Seja a consulta do exemplo 1 sobre a relação de alunos

```
SELECT NUSP, Nome FROM Alunos
```

- que gera a árvore de comandos:
- Como não pode ser usado nenhum índice, o plano de acesso físico gerado é:



- Como calculado antes, esse comando irá demorar $\boxed{\text{R}3.810\text{S}} = \frac{3.810}{1.000} = 3,81$ segundos.

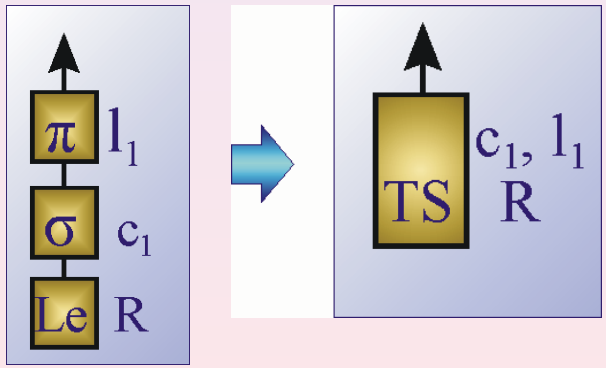
Métodos de Acesso para operações unárias

Table Scan – Exemplo Consulta 4

- Seja agora a seguinte consulta sobre a mesma relação de alunos

```
SELECT NUSP, Nome FROM Alunos
    WHERE Nome='Zezinho'
```

- Essa consulta gera a árvore de comandos:
- Como não existe nenhum índice sobre o atributo `Nome`, o plano de acesso físico gerado é:



- E da mesma maneira, esse comando irá demorar $R3.810S = \frac{3.810}{1.000} = 3,81$ segundos.

Métodos de Acesso para operações unárias

Index Scan

- O método de acesso **Index Scan** é o método básico para acessar uma relação usando um índice associado.
- Ele também empacota uma sequência de operadores  ,  e  , porém a fase de seleção faz uso de um índice.

Métodos de Acesso para operações unárias

Index Scan

- Vamos assumir que esteja sendo usado um índice B-tree:
- O *table scan*:

  ☞ Acessa o índice até a folha que contem o RowId da primeira tupla com a chave de busca,

  ☞ lê em sequência todas as folhas a partir da primeira, até achar uma chave diferente da chave de busca,

  ☞ **lê** previsivelmente todos as paginas apontadas pelas entradas das folhas lidas,

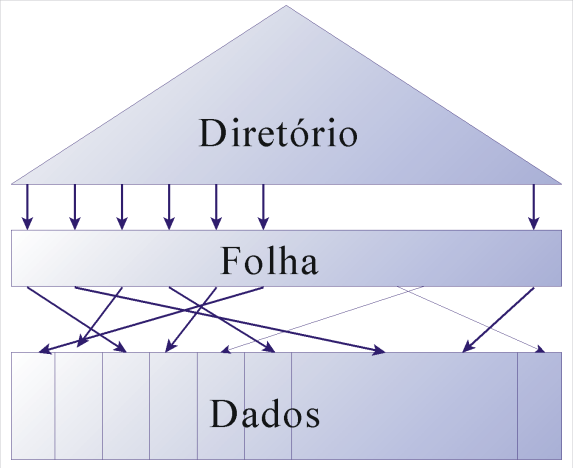
  ☞ caso haja mais critérios de busca não cobertos pelo índice, aplica o critério combinado dos demais **operadores de seleção** empacotados para todas as tuplas lidas,

  ☞ e escreve no *buffer* de saída as tuplas projetadas segundo a lista de tuplas do **operador de projeção** "empacotado".

Métodos de Acesso para operações unárias

Index Scan

- **O custo** do *Index Scan* é a leitura aleatória dos diretórios do índice (um para cada nível $H-1$ da árvore), mais a leitura sequencial das *NFolhas* que contém a chave de busca, mais a leitura previsível das n tuplas recuperadas pelo índice:
  $\mathrm{R}H\mathrm{R} + \mathrm{R}(NFolhas)\mathrm{S} + \mathrm{R}n\mathrm{T}$.



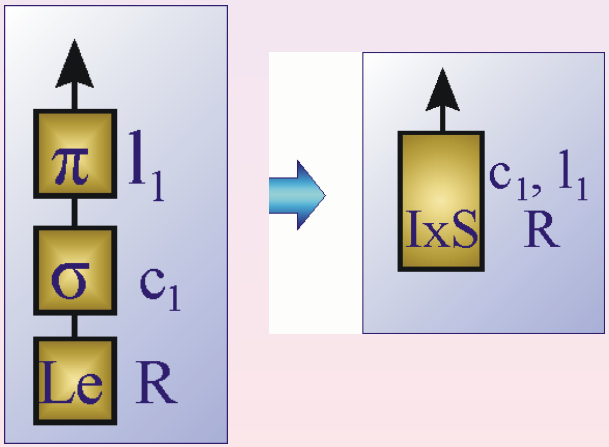
Métodos de Acesso para operações unárias

index Scan – Exemplo Consulta 2

- Seja a consulta do exemplo 2 sobre a relação de alunos:

```
SELECT NUSP, Nome FROM Alunos
    WHERE NUSP='1231231234'
```

- que gera a árvore de comandos:
- Agora o índice da chave primária pode ser usado. Portanto, o plano de acesso físico gerado é:



- Como calculado antes, esse comando irá demorar $R4R = \frac{4}{100} = 0,04$ segundos.

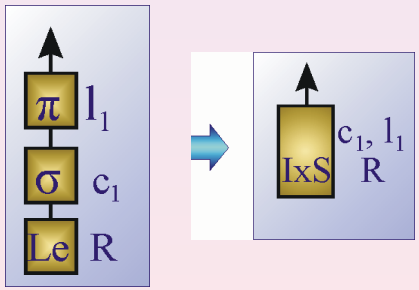
Métodos de Acesso para operações unárias

index Scan – Exemplo Consulta 3

- Nem sempre usar um índice é a melhor opção.
- Por exemplo, seja a consulta do exemplo 3 sobre a relação de alunos:

```
SELECT NUSP, Nome FROM Alunos
    WHERE NUSP BETWEEN 2000000000 AND 3000000000
```

- que gera a árvore de comandos:
- Vamos supor que é usado o índice da chave primária para responder a essa consulta. Então, o plano de acesso físico gerado é:



- Como calculado antes, esse comando irá demorar $\boxed{\text{R3R}} + \boxed{\text{R103S}} + \boxed{\text{R8.000T}} = \frac{3}{100} + \frac{103}{1.000} + \frac{8.000}{300} = 0,03 + 0,103 + 26,67 = 26,80$ segundos.

Métodos de Acesso para operações unárias

Index Scan

- Se a consulta for realizada usando um *table scan*, sem uso de índice, o tempo para acessar toda a relação é de $\boxed{\text{R3.810S}} = \frac{3.810}{1.000} = 3,81$ segundos.
- Se tentar usar o índice sobre a chave primária e um *index scan*, mesmo que ele permita acessar apenas $\frac{1}{10}$ da relação, o tempo de acesso aumenta para 26,80 segundos.
- Isso se deve a dois motivos:
  1. O acesso pelo índice gera uma "chuva de ponteiros" sobre as páginas de dados (uma mesma página pode ser lida mais de uma vez),
  2. O acesso é executado por acessos previsíveis, que usam o cache do disco mas não consegue aproveitar a proximidade das páginas em um *extent*.

Métodos de Acesso para operações unárias

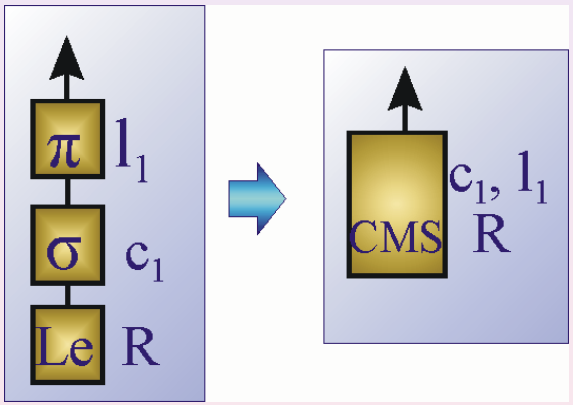
Clustered Matching Index Scan

- O método de acesso físico **Index Scan** pode ser implementado em um SGBD por mais de um algoritmo, para acessar relações que não estão *clusterizadas*:
  - Ele pode ser chamado *Equal Unique Index Lookup* quando a busca é por comparação por igualdade uma chave da relação;
  - Ele pode ser chamado *Unclustered Matching Index Scan* quando a busca é por comparação usando uma chave de busca que não é única;
  - Em geral, o acesso por *Index Scan* pode ser usado para comparação usando operadores de abrangência (*range predicates*) ou quando uma chave de busca não é única.
- Caso a relação esteja *clusterizada*, pode ser usado o método de acesso **Clustered Matching Index Scan**.

Métodos de Acesso para operações unárias

Clustered Matching Index Scan

- O método de acesso **Clustered Matching Index Scan** também empacota uma sequência de operadores Le R, σ c_1 e π l_1 usando um índice na execução do operador de seleção.



- O *Clustered Matching Index Scan* é semelhante ao *index scan*, a menos que ele é utilizado quando os dados são mantidos sincronizados com a lista de RID da folha do índice.

Métodos de Acesso para operações unárias

Clustered Matching Index Scan

- Vamos assumir que esteja sendo usado um índice B-tree. Então o *Clustered Matching Index Scan*:

  ☞ Acessa o índice até a folha que contem o RowId da primeira tupla com a chave de busca,

  ☞ lê em sequência todas as folhas a partir da primeira, até achar uma chave diferente da chave de busca,

  ☞ **lê** sequencialmente todas as paginas distintas apontadas pelas entradas das folhas lidas,

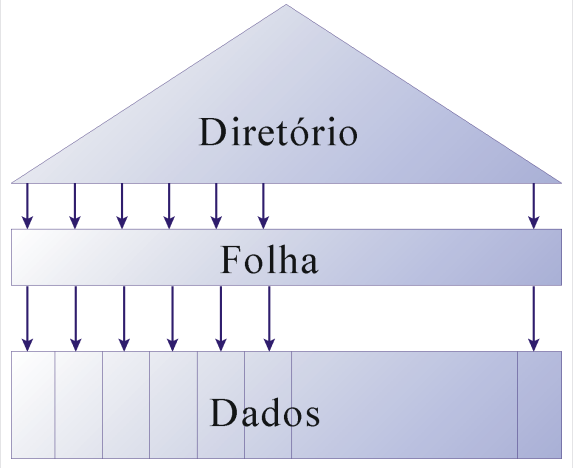
  ☞ caso haja mais critérios de busca não cobertos pelo índice, aplica o critério combinado dos demais **operadores de seleção** empacotados para todas as tuplas lidas,

  ☞ e escreve no *buffer* de saída as tuplas projetadas segundo a lista de tuplas do **operador de projeção** "empacotado".

Métodos de Acesso para operações unárias

Clustered Matching Index Scan

- **O custo** do *Clustered Matching Index Scan* é a leitura aleatória dos diretórios do índice (um para cada nível H da árvore), mais a leitura sequencial das *NFolhas* que contém a chave de busca, mais a leitura sequencial das $Sel(predicado)$ páginas recuperadas pelo índice:
$RHR + R(NFolhas)S + R(Sel(predicado) * NPags)S$.



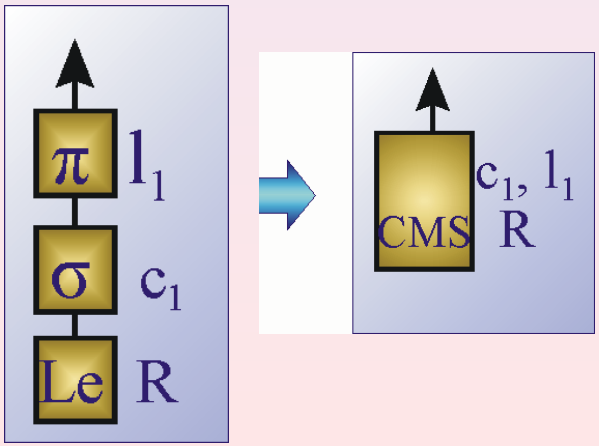
Métodos de Acesso para operações unárias

Clustered Matching Index Scan – Exemplo Consulta 2

- Vamos assumir que a relação alunos é *clusterizada* pela sua chave primária.
- Nesse caso, a consulta do exemplo 2 sobre a relação de alunos:

```
SELECT NUSP, Nome FROM Alunos
    WHERE NUSP='1231231234'
```

- que gera a árvore de comandos:
- Agora gera o plano de acesso físico:



Mas o método *Clustered Matching Index Scan* não altera o tempo de busca, pois como antes, esse comando irá demorar

$R4R = \frac{4}{100} = 0,04$ segundos.

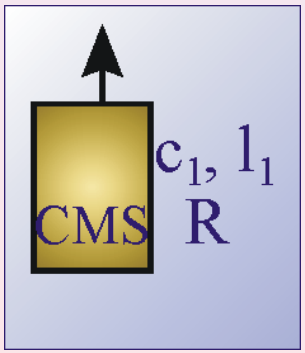
Métodos de Acesso para operações unárias

Clustered Matching Index Scan – Exemplo Consulta 3

- No entanto, a consulta do exemplo 3 sobre a relação de alunos:

```
SELECT NUSP, Nome FROM Alunos
    WHERE NUSP BETWEEN 2000000000 AND 3000000000
```

- embora gere a mesma árvore de comandos:



  - agora tem as tuplas sincronizadas com os nós-folha do índice, e portanto irá demorar $R3R + R103S + R(Sel(pred) * NTuplas)S$, onde $pred =$ (`NUSP BETWEEN 2000000000 AND 3000000000`);
  - $Sel($`NUSP BETWEEN 2000000000 AND 3000000000`$) = \frac{3000000000-2000000000}{9988776655-0011223344} \approx 0,1$
  - Portanto $\frac{3}{100} + \frac{103}{1.000} + \frac{0,1*8.000}{1.000} = 0,03 + 0,103 + 0,80 = 0,933$ segundos.

Métodos de Acesso para operações unárias

index Scan – Exemplo Consulta 3

- Comparando com o uso do *index scan* não *clusterizado*, vê-se que há um ganho de 26,80 para 0,93 segundos.
- Isso se deve a dois motivos:
  1. A "chuva de ponteiros" sobre as páginas de dados desaparece, pois os dados ficam na mesma sequência dos nós-folha;
  2. Cada página é lida apenas uma vez, pois múltiplos ponteiros que caem na mesma folha são eliminados por serem solicitados em sequência, e portanto garante-se que as páginas necessárias continuam no *buffer*.

Métodos de Acesso para operações unárias

Index-only Scan

- Uma relação somente pode ser *clusterizada* pela sua chave primária, portanto não é possível sincronizar as tuplas com outro índice.
- No entanto, é possível um efeito parecido com a seguinte observação:

É possível construir um índice que contém, sozinho, todos os dados necessários para responder a uma dada consulta.

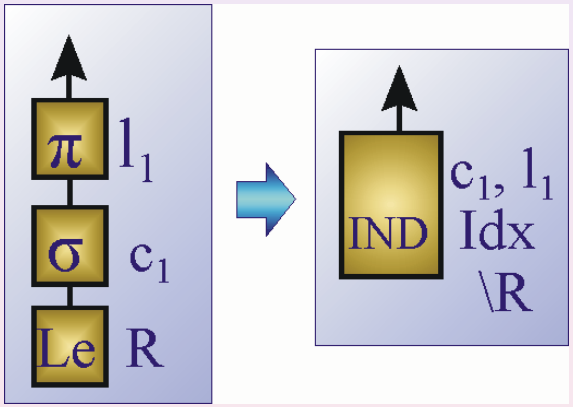
- Nesse caso, não é necessário o acesso aos dados: basta obter no índice os dados solicitados.
- O método de acesso que não faz a leitura dos dados, apenas do índice é chamado **Index-only Scan**.

☞ Veja que esse índice é equivalente ao *index scan* ou ao *clustered matching index scan*, pois como não é feito o acesso aos dados, é indiferente se os dados estão sincronizados ou não.

Métodos de Acesso para operações unárias

Index-only Scan

- O método de acesso *Index-only Scan* também empacota uma sequência de operadores Le R, σ c_1 e π l_1:



- No entanto, o *Index-only Scan* muda bastante a estrutura da árvore de comandos da consulta, pois ele altera a leitura básica, que deixa de ser feita sobre os dados e passa a ser feita apenas no índice.

Métodos de Acesso para operações unárias

Index-only Scan

- Vamos assumir que esteja sendo usado um índice B-tree. Então o *Index-Only Scan*:

  ☞ Acessa o índice até a folha que contem o RowId da primeira tupla com a chave de busca,

  ☞ lê em sequência todas as folhas a partir da primeira, até achar uma chave diferente da chave de busca,

  ☞ monta cada chave lida como se fosse uma tupla,

  ☞ caso haja mais predicados cobertos pelo índice, aplica o critério combinado dos demais **operadores de seleção** empacotados para filtrar todas as tuplas montadas,

  ☞ e escreve no *buffer* de saída as tuplas montadas já projetadas segundo a lista de tuplas do **operador de projeção** "empacotado".

- A aplicação de múltiplos critérios de busca cobertos pelo índice é tratada de maneira especial:

Métodos de Acesso para operações unárias

Index-only Scan — Aplicação de múltiplos critérios de busca cobertos pelo índice

- Apenas **predicados indexáveis** podem ser utilizados como chave de busca em um método de acesso *Index-Only Scan*.
- Outros predicados que podem ser comparados no índice somente podem ser usados como um predicado para *screening*,
- e são usados como **predicados de restrição** para eliminar tuplas que passam pelos predicados indexáveis.

Métodos de Acesso para operações unárias

Index-only Scan — Aplicação de múltiplos critérios de busca cobertos pelo índice

- Por exemplo, suponha que a relação de alunos tem os seguintes atributos:
  `Alunos={NUSP, Nome, NomeMae, DataNasc, Altura, Cidade}`
- é criado o índice:
  `CREATE INDEX NomeNomeMae ON Alunos(Nome, NomeMae, DataNasc)`
- e é feita a seguinte consulta:

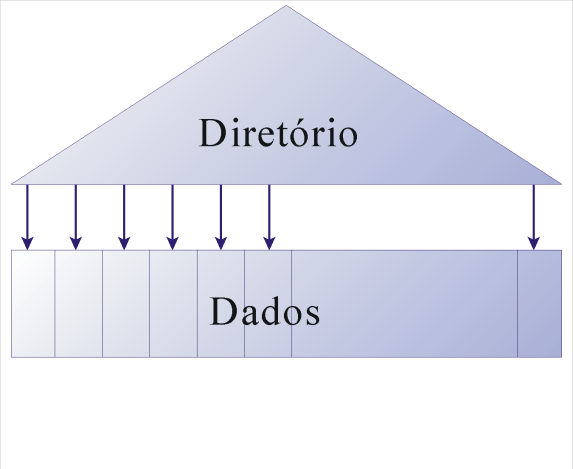
```
SELECT Nome, NomeMae FROM Alunos
    WHERE Nome=:nome AND DataNasc=:data;
```

- Como `NomeMae` não é atributo indexável (não existe predicado que o use), `DataNasc` não pode ser usado para busca,
- mas o predicado `DataNasc=:data` pode ser usado como predicado de restrição no próprio método de acesso *Index-Only Scan*.

Métodos de Acesso para operações unárias

Index-only Scan

- O efeito geral do *index-only scan* é do acesso a uma estrutura de dados primária.
- **O custo** do *INDex-only scan* é a leitura aleatória dos diretórios do índice (um para cada nível $H-1$ da árvore), mais a leitura sequencial das *NFolhas* que contém a chave de busca:
  $R(H-1)R + R(NFolhas)S$.



Métodos de Acesso para operações unárias

Exemplo – Consulta 4 – Acesso indexado apenas por índice

- Suponha que é frequente solicitar a `Idade` de um aluno dado seu `Nome`, e que é frequente solicitar os `Nomes` de todos os alunos de uma dada `Cidade`,
- então os seguintes índices podem ser criados:

```
CREATE INDEX NomeIdade ON Alunos(Nome, Idade);
CREATE INDEX CidadeNome ON Alunos(Cidade, Nome);
```

- Como previsto, agora se quer saber a idade de um aluno:

```
SELECT Nome, Idade
    FROM Alunos
    WHERE Nome=:nome;
```

- Basta acessar o índice `NomeIdade`, obter todos os pares <Nome, Idade> cujas chaves têm valor do `Nome` dado, e retornar o resultado (sem acessar o segmento de dados).
- Assumindo que essa árvore ainda terá altura $H = 3$ e que o número médio de tuplas repetidas por chave é menor do que dois, o custo dessa consulta é $R(2)R + R(2)S \approx \frac{4}{100} = 0,04$s (acessar duas páginas em sequência é equivalente ao acesso aleatório).

Métodos de Acesso para operações unárias

Exemplo – Consulta 4 – Acesso indexado apenas por índice

- Vamos agora considerar a segunda consulta: Obter os `Nomes` de todos os alunos de uma dada `Cidade`, dada por:

```
SELECT Nome
    FROM Alunos
    WHERE Cidade=:cidade;
```

- Novamente, basta acessar o índice `CidadeNome`, obter todos os pares `<Cidade, Nome>` cujas chaves têm valor da `Cidade` dado, e retornar o resultado (novamente sem acessar o segmento de dados).
- Assumimos novamente que essa árvore ainda terá altura $H = 3$ mas o número médio de tuplas repetidas por chave é dado por: $\frac{Sel(pred) * Ntuplas}{TuplasPorPagina}$. Para isso, é necessário estimar o número de cidades distintas que têm alunos na relação. Usando as estimativas feitas no inicio da apresentação, temos que $Sel(\text{Cidade} = cte) * 80.000 = \frac{80.000}{700} = 114,29 \approx 114$.
- O tamanho e cada chave é $dir + 15 + 15 = 32$, então o número de tuplas por página é $\left\lfloor \frac{2.000 - 2*4}{32} \right\rfloor = 62.25 = 62$.
- Portanto, o custo dessa consulta é $R(2)R + R(114/62)S = \frac{2}{100} + \frac{1,84}{1000} \approx 0,04s$ novamente.

Métodos de Acesso para operações unárias

Exemplo – Consulta 4 – Acesso indexado apenas por índice

- Veja que se não existisse o índice `CidadeNome`, seria necessário realizar um *table scan* sobre toda a relação, ao custo (já calculado no exemplo 1) de 3,81 segundos.
- Caso houvesse um índice sobre `Cidade` mas que não incluisse o `Nome`, seria necessário utilizar um *index scan* nesse índice, ao custo de $\boxed{\mathrm{R}\mathit{H}\mathrm{R}} + \boxed{\mathrm{R}(\mathit{NFolhas})\mathrm{S}} + \boxed{\mathrm{R}\mathit{NTuplas}\mathrm{T}} = \frac{3}{100} + \frac{80.000}{62*700*1.000} + \frac{114}{300} = 0,03 + 0,0018 + 0,38 = 0,41$ segundos,
- ou seja, aproximadamente 10 vezes mais demorado do que utilizar o índice `CidadeNome`.

Métodos de Acesso para operações unárias

Multi-Index Scan

- Se na consulta houver algum outro operador de seleção que envolva um atributo que não está no índice, o método *index-only scan* não pode ser usado.
- No entanto, uma outra observação pode ajudar novamente:

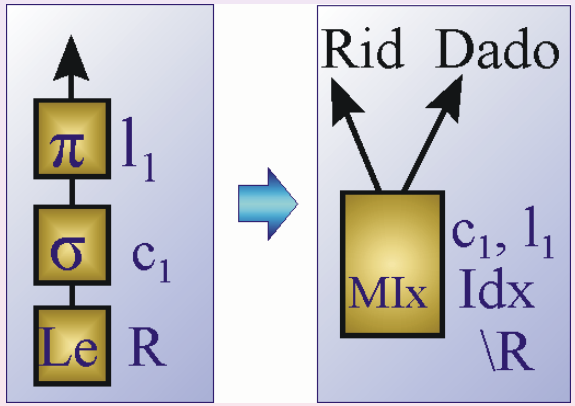
Todo índice, mesmo que seja usado apenas para *index-only scan*, tem o *RowId* para as tuplas de dados correspondentes.

- Isso significa que se houver dois ou mais índices que, juntos, contenham todos os dados necessários a uma consulta, é possível integrar os dados de todos eles usando o *RowId* como chave de junção, e não é necessário acessar os segmentos de dados.
- Mas para isso, o método que faz a leitura de um índice tem que recuperar os dados e os *RowId* como duas informações separadas.
- O método de acesso do índice recuperando as chaves e os *RowId*, sem fazer a leitura dos dados, é chamado **Multi-Index Scan**.

Métodos de Acesso para operações unárias

Multi-Index Scan

- O método de acesso *Multi-Index Scan* também empacota uma sequência de operadores Le R, σ c_1 e π l_1:



- No entanto, o *Multi-Index Scan* gera dois conjuntos de saída: uma relação de tuplas e uma Lista de *RowId*, a *Rid-list*.

Métodos de Acesso para operações unárias

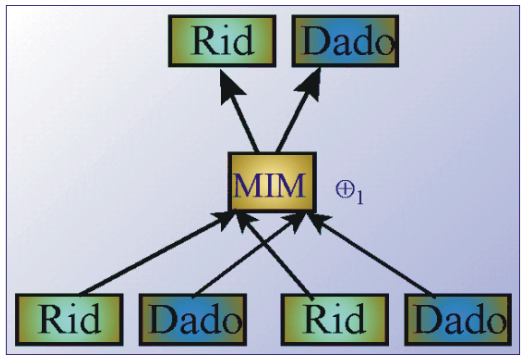
Multi-Index Scan

- Vamos assumir que esteja sendo usado um índice B-tree. Então o *Multi-Index Scan*:
  - ☞ Acessa o índice até a folha que contem o *RowId* da primeira tupla com a chave de busca,
  - ☞ lê em sequência todas as folhas a partir da primeira, até achar uma chave diferente da chave de busca,
  - ☞ monta cada chave lida como se fosse uma tupla,
  - ☞ e cria uma lista de *RowId*, a *Rid-list*,
  - ☞ caso haja mais critérios de busca cobertos pelo índice, aplica o critério combinado dos demais **operadores de seleção** empacotados para filtrar todas as tuplas montadas,
  - ☞ escreve no *buffer* de saída as tuplas montadas já projetadas segundo a lista de tuplas do **operador de projeção** "empacotado",
  - ☞ ordena a *Rid-list* e a escreve em uma área específica para *Rid-list* na estrutura de *buffers*.

Métodos de Acesso para operações unárias

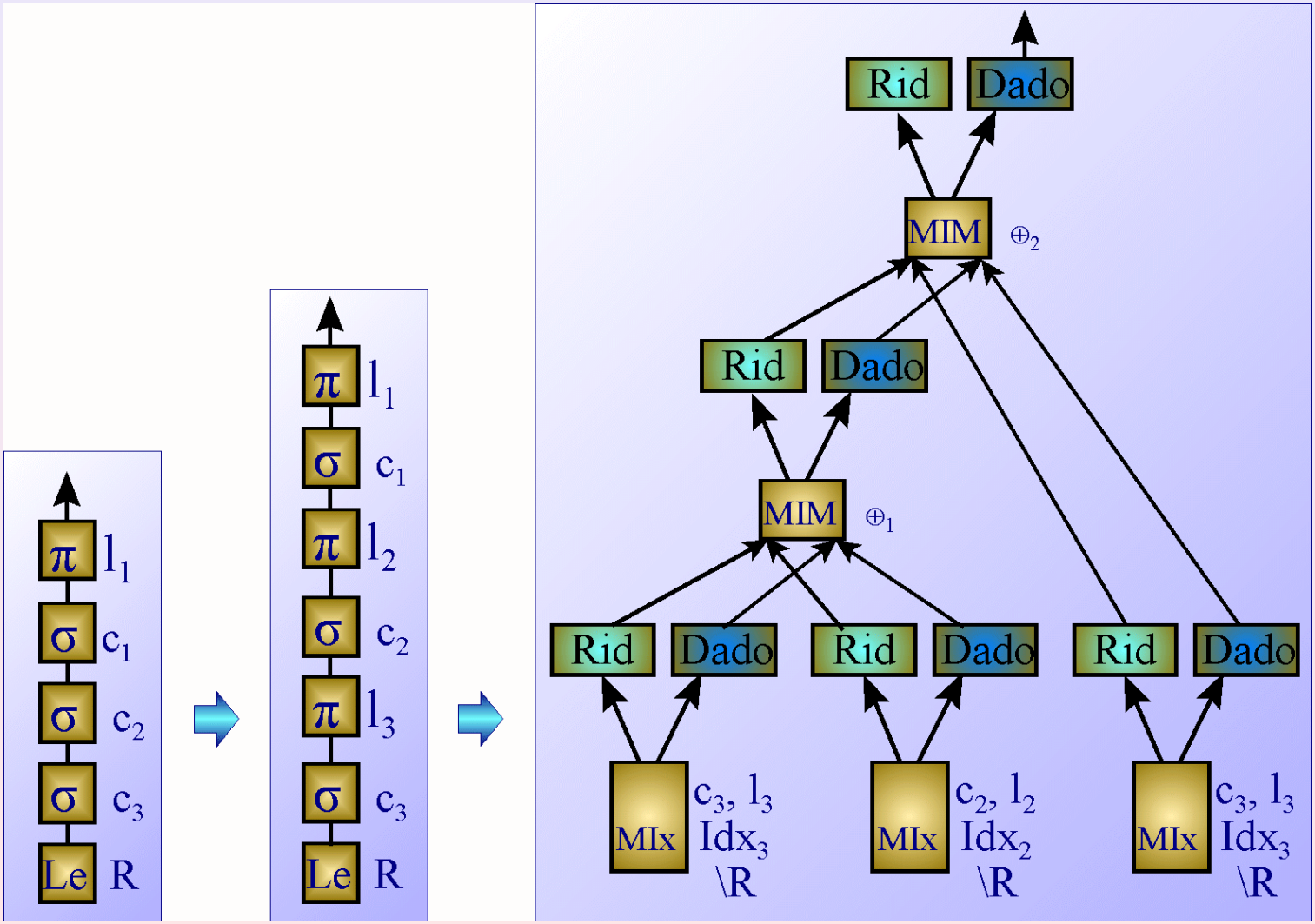
Multi-Index Scan

- O método *Multi-Index Scan* somente é usado quando existe mais de um índice que pode ser usado para acesso *index-only*.
- Somente podem existir duas áreas para *Rid-list* simultaneamente.
- Isso significa que sempre deve ser aplicado um operador de integração de duas *Rid-list*, chamado **Multi-Index Merger** antes que uma terceira possa ser aplicada:



Métodos de Acesso para operações unárias

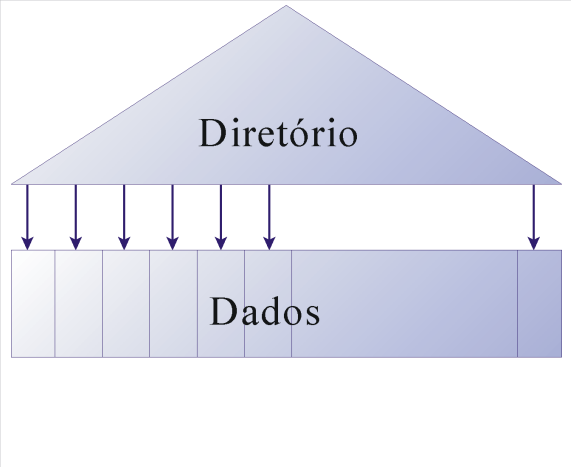
Multi-Index Scan



Métodos de Acesso para operações unárias

Multi-Index Scan

- **O custo** do *INDex-only scan* é a leitura aleatória dos diretórios do índice (um para cada nível $H-1$ da árvore), mais a leitura sequencial das *NFolhas* que contém a chave de busca:
  $R(H-1)R + R(NFolhas)S$.



Métodos de Acesso para operações unárias

Multi-Index Scan

- Para poder utilizar o método de acesso físico *multi-index scan*, algumas condições devem ser seguidas:
- A previsão de espaço necessário para a ordenação dos métodos *multi-index scan* e do método *multi-index merge* intermediário deve indicar que ela pode ser completamente realizada em memória, caso contrário o *multi-index scan* não é escolhido.
- A seguir deve-se calcular a seletividade composta de todos os predicados indexáveis para cada índice.
- Sempre executa-se primeiro os índices que têm a maior seletividade composta de todos os predicados indexáveis desse índice.
- A medida que os índices vão sendo processados (vão sendo previstos para entrar no plano de acesso), vai se calculando a seletividade composta de todos os índices já colocados no plano, e calcula-se o custo do plano sendo gerado até esse ponto, o custo de acesso ao índice e o custo de acesso ao dado;
- quando o custo do plano sendo gerado até esse ponto torna-se inferior ao custo de acesso ao dado, esse índice e os demais de seletividade ainda menor são descartados e acessa-se as tuplas para obter os dados remanescentes.

Métodos de Acesso para operações unárias

Exemplo – Consulta 5 – Acesso indexado por múltiplos índices

- Suponha que os seguintes índices foram criados sobre a relação `Alunos`:

```
CREATE INDEX IdadeNome ON Alunos(Idade, Nome);
CREATE INDEX CidadeNome ON Alunos(Cidade, Nome);
```

- É solicitada a consulta sobre quais são os nomes dos alunos com menos de 20 anos que vêm de Rio Claro:

```
SELECT Nome, Idade
    FROM Alunos
    WHERE Cidade='Rio Claro' AND Idade<20;
```

- Primeiro calcula-se a seletividade de todos os predicados indexáveis. Para isso, usam-se as estatísticas disponíveis como anteriormente mostrado na tabela:
  - $Sel(\text{Cidade} = \text{'Rio Claro'}) = \frac{1}{700}$
  - $Sel(\text{Idade} = 20) = \frac{1}{50}$
- Executa-se primeiro os índices que têm a maior seletividade composta de todos os predicados indexáveis de um índice.
- Neste caso processa-se primeiro o índice `CidadeNome` e depois o índice `IdadeNome`.

Métodos de Acesso para operações unárias

Exemplo – Consulta 5 – Acesso indexado por múltiplos índices

- O custo do índice `CidadeNome` é: $H = 3$, Número de chaves por folha$= \left\lfloor \frac{2.000-2*4}{2+15+15} \right\rfloor = 62,25 = 62$, número de tuplas repetidas por chave$NTuplas = \frac{80.000}{700} = 114,29 = 114$, número de folhas a serem lidas $= NFolhas = \left\lceil \frac{114}{62} \right\rceil = 1.84 = 2$.
- O custo para acessar o índice `CidadeNome` é: $\boxed{\text{R}(H-1)\text{R}} + \boxed{\text{R}(NFolhas)\text{S}} = \boxed{\text{R}(2)\text{R}} + \boxed{\text{R}(2)\text{S}} \approx \frac{4}{100} = 0,04$ segundos.
- o Custo para acessar os dados depois de filtrado esse índice é: $\boxed{\text{R}(\text{NTuplas})\text{T}} = \frac{114}{300} = 0,38\text{s}$.

Métodos de Acesso para operações unárias

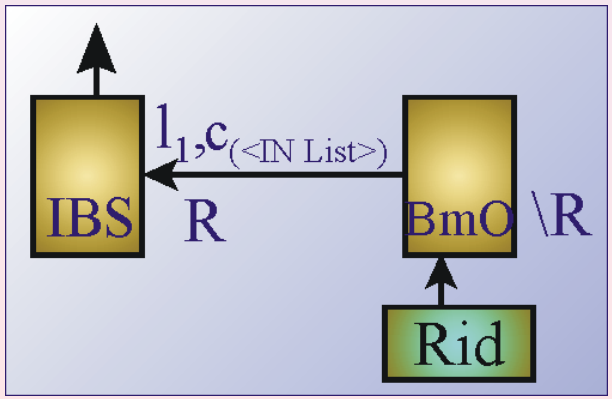
Exemplo – Consulta 5 – Acesso indexado por múltiplos índices

- O custo do índice `IdadeNome` é: $H = 3$, Número de chaves por folha$= \lfloor \frac{2.000-2*4}{2+15+2} \rfloor = 104,84 = 104$, número de tuplas repetidas por chave$NTuplas = \frac{80.000}{50} = 114,29 = 1.600$, número de folhas a serem lidas $= NFolhas = \lceil \frac{1.600}{104} \rceil = 14,38 = 15$.
- O custo para acessar o índice `IdadeNome` é: $R(H-1)R + R(NFolhas)S =$ $R(2)R + R(15)S = \frac{2}{100} + \frac{15}{1000} = 0,035$ segundos.
- A seletividade composta desses dois índices é $\frac{1}{700} \cdot \frac{1}{50} = \frac{1}{35.000}$, portanto aproximadamente $\frac{1}{35.000} \cdot 80.000 = 2.29$ tuplas serão recuperadas em média.
- o Custo para acessar os dados depois de filtrado pelos dois índices é: $R(NTuplas)T = \frac{3}{300} = 0,01s$.
- Mas como todos os dados necessários já estão obtidos ao custo de $0,04 + 0,035 = 0,075s$, não é necessário nenhum acesso aos dados.
- Finalmente, veja que as ordenações que devem ser executadas em memória corresponde a ordenar 114 *RowIds* para o índice `CidadeNome`, depois ordenar os 1.600 *RowIds* para o índice `IdadeNome` e fazer o *merging* de ambas *Rid-lists*.

Operadores de apoio aos Métodos de Acesso

Bitmap Sorter

- Além dos métodos de acesso propriamente ditos, que de fato acessam os dados, existem alguns outros operadores de apoio aos métodos.
- Um deles é o operador de integração **Multi-Index Merger**, já estudado.
- Outro operador de apoio interessante é o **Bitmap Sorter**:
- ele transforma uma *Rid-list* em um predicado do tipo `IN (lista)` para ser usado em um outro método de acesso aos dados.



Operadores de apoio aos Métodos de Acesso

Exemplo – Consulta 6 – Acesso por múltiplos índices e aos dados

- Por exemplo, suponha que os mesmos índices criados na Consulta 5 do exemplo anterior existam sobre a relação `Alunos`:

```
CREATE INDEX IdadeNome ON Alunos(Idade, Nome);
CREATE INDEX CidadeNome ON Alunos(Cidade, Nome);
```

- Mas agora é solicitada a consulta sobre quais são os nomes e número USP dos alunos com menos de 20 anos que vêm de Rio Claro:

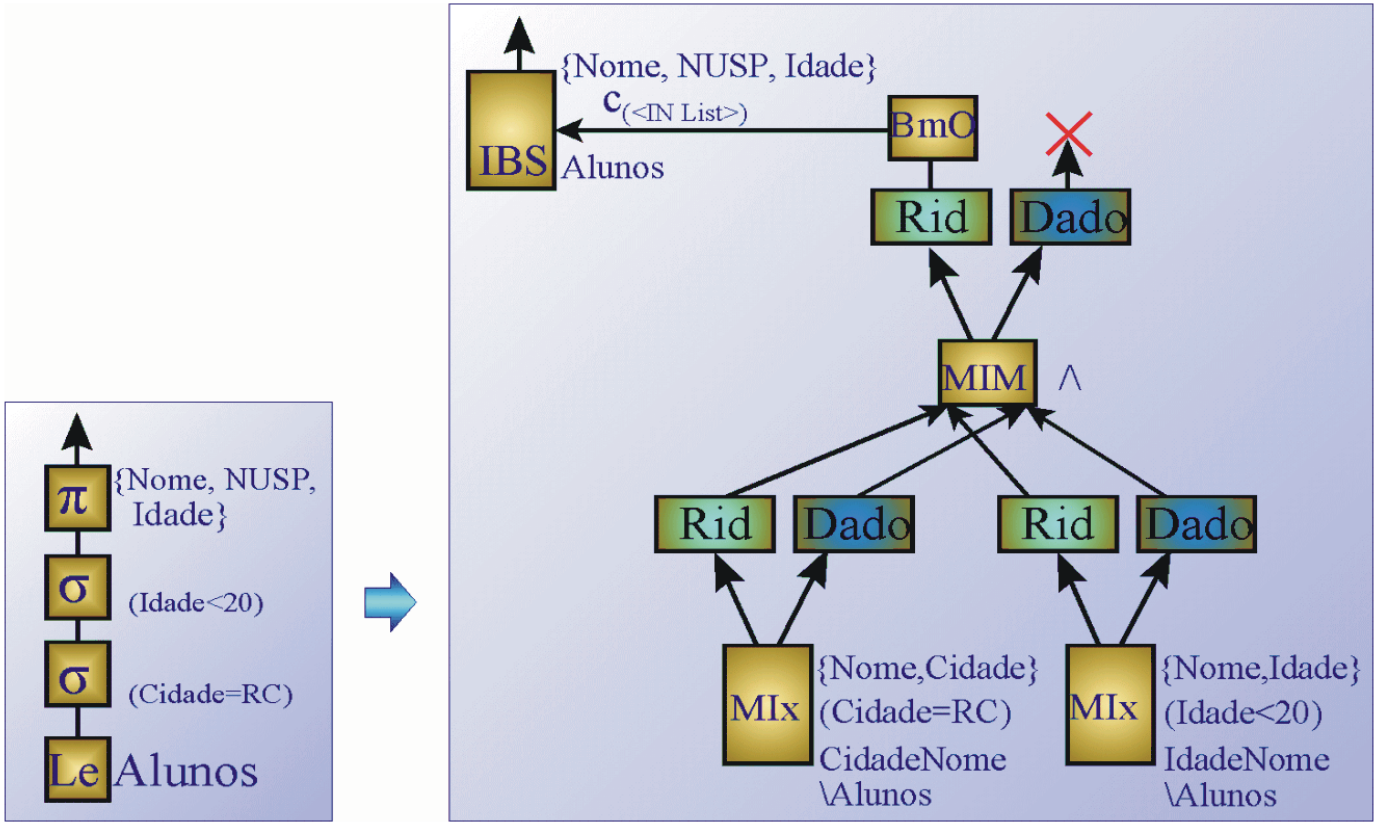
```
SELECT Nome, NUSP, Idade
    FROM Alunos
    WHERE Cidade='Rio Claro' AND Idade<20;
```

- Nesse caso, nenhuma combinação de índice de busca pode suprir todos os dados necessários.
- Embora exista uma chave primária sobre o atributo `NUSP`, ela não pode ser usada como chave de busca porque não tem um predicado sobre ela.
- No entanto, o uso combinado dos dois índices `CidadeNome` e `IdadeNome`, como se viu no exemplo anterior, permite restringir o acesso necessário a uns poucos dados.
- Então pode-se usar os índices para obter a *Rid-list* das tuplas que devem ser lidas, e acessar só elas.

Operadores de apoio aos Métodos de Acesso

Bitmap Sorter

```
SELECT Nome, NUSP, Idade
    FROM Alunos
    WHERE Cidade='Rio Claro' AND Idade<20;
```



Operadores de apoio aos Métodos de Acesso

Bitmap Sorter

- De fato, o operador *Bitmap Sorter* gera:
  - a *Rid-list* das tuplas que precisam ser acessadas,
  - junto com um *bitmap* dos *extents* do segmento de dados que precisam ser acessados.
- Com o *bitmap* dos *extents*, o método de acesso usado pode ordenar os acessos, e acessar uma única vez cada *extent* que contém qualquer quantidade de tuplas.
- Portanto, o operador *Bitmap Sorter* permite eliminar a " chuva de ponteiros" sobre as tuplas, a partir de qualquer índice utilizado.

Operadores de apoio aos Métodos de Acesso

Exemplo – Consulta 6 – Acesso por múltiplos índices e aos dados

- Prosseguindo com o exemplo anterior, os mesmos acessos já calculados serão utilizados, o que corresponde ao custo de $0,04 + 0,035$ segundos para os índices `CidadeNome` e `IdadeNome` respectivamente, sem contar ainda o acesso aos dados.
- Como nenhum dos índices usáveis tem o atributo `NUSP`, ele é obtido direto nas tuplas.
- Como o operador *Multi-Index Merger* já gerou a *Rid-list* de todas as tuplas necessárias, o operador *Bitmap Sorter* extrai os *extents* necessários e passa essa lista para um método de acesso, que lê os dados, faz a projeção final e responde a consulta.
- O custo para o acesso aos dados será o de acessar aleatoriamente as 3 tuplas que foram identificados anteriormente, correspondente a $\boxed{\text{R3R}} = \frac{3}{100} = 0,03$.
- Portanto o custo total dessa consulta será $0,04 + 0,035 + 0,03 = 0,105$ segundos.

Operadores de apoio aos Métodos de Acesso

Exemplo – Consulta 6 – Acesso por múltiplos índices e aos dados

- Vamos considerar um plano de acesso alternativo que não use o método *Multi-Index Scan*.
- Nesse caso, somente um índice pode ser utilizado. Vamos escolher o mais seletivo: `CidadeNome`.
- Seria então utilizado o método *Index Scan*, a um custo de: $\mathrm{R}\mathit{H}\mathrm{R} + \mathrm{R}(\mathit{NFolhas})\mathrm{S} + \mathrm{R}n\mathrm{T} = \frac{3}{100} + \frac{2}{1.000} + \frac{114}{300} = 0,412$ segundos, quatro vezes mais demorado.
- O problema desse acesso é a chuva de ponteiros que o índice `CidadeNome` causa sobre as páginas de dados.
- Além disso, o predicado `(Idade<20)` somente pode ser usado para filtragem, pois não podem ser usados dois índices para acesso indexado.

Operadores de apoio aos Métodos de Acesso

Operador **Hash Table**

- Um outro operador de apoio aos métodos de acesso é o chamado **Hash Table**, ou simplesmente **Hash**.
- Ele é aplicado entre dois métodos de acesso:
  - O método anterior deve recuperar tuplas que contém os valores de alguns atributos que deverão ser usados para filtragem;
  - O método posterior filtra as tuplas pelos valores recuperados.
- Esse operador é útil quando são executadas operações de junção, e será discutido oportunamente.

Métodos de Acesso para operações unárias

Uma variante para os métodos de acesso físicos indexados

- Os métodos de acesso físico baseados em um índice estudados até agora fazem o acesso aos dados utilizando apenas o índice que provê a maior seletividade combinada dos predicados indexáveis que ele permite usar.
- Os demais predicados, mesmo que haja outros índices disponíveis, não podem ser usados.
- Então apenas um índice é acessado, e o método de acesso usado já filtra as tuplas usando todos os demais predicados solicitados na consulta, mas sem usar outros índices.
- Isso significa que as tuplas indicadas pelo índice único têm que ser efetivamente lidas e então filtradas, sem apoio de outro índice.
- Uma variante é a seguinte:

Cada método de acesso tem uma variante que recupera apenas a *Rid-List* das tuplas que podem conter as tuplas necessárias.

Variantes para os métodos de acesso físicos indexados

- A *Rid-list* pode ser mantida ou como uma lista de *RowID*, ou como um *bitmap* sobre as páginas (e a seguir de *extents*) que contém as tuplas de interesse.
- Em ambos os casos, conforme os índices vão sendo acessados, a *Rid-List* vai sendo montada por operadores *Multi-Index Merger*, e apenas ao final é utilizada a variante do método de acesso para acessar os dados e gerar a resposta final.
- Por exemplo, considerando a variante que gera um *bitmap*, existirá:
  - *Table Scan* ☞ *Bitmap Table Scan*
  - *Index Scan* ☞ *Bitmap Index Scan*
  - *Clustered Matching Index Scan* ☞ *Clustered Matching Bitmap Index Scan* e
  - *Index-only Scan* ☞ *Bitmap Index-only Scan*
  - *Multi-Index Scan* ☞ *Bitmap Multi-Index Scan*

Variantes para os métodos de acesso físicos indexados

- Um ponto importante a destacar é que ordenar a *Rid-List* permite o acesso às páginas na máxima velocidade possível, tornando no pior caso o acesso equivalente à leitura sequencial da relação inteira.
- No entanto, a etapa de ordenação, além do próprio custo da ordenação, causa o seguinte efeito colateral nem sempre admissível:

A ordenação somente pode ser feita depois que todas as tuplas foram recuperadas. Assim não é possível fornecer respostas parciais incrementais.

- Portanto o uso dos operadores de *bitmap* prejudica a concorrência na paginação de resultados,
- e eles são desabilitados quando a consulta tem a cláusula:

SQL:2008

```
OFFSET start ROW | ROWS
FETCH FIRST | NEXT [<count>] ROW | ROWS ONLY
```

Métodos de Acesso para operações binárias

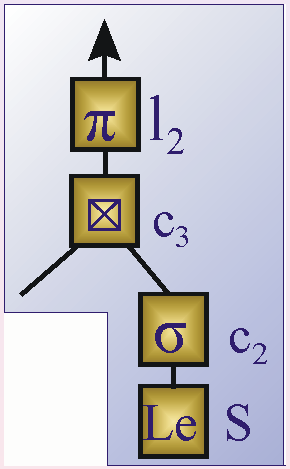
Idéia geral

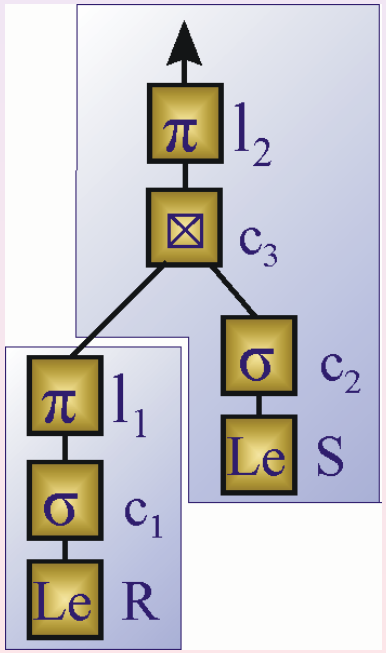
- Os métodos de acesso físicos estudados até agora resolvem apenas os operadores $\boxed{\text{Le}}\ \text{R}$, $\boxed{\sigma}\ c_1$ e $\boxed{\pi}\ l_1$.
- Para executar as operações de junção, existem métodos de acesso físicos específicos, chamados Binários, ou operadores SPJ (operadores "Seleção Projeção Junção").
- Todos eles empacotam os operadores $\boxed{\text{Le}}\ \text{R}$, $\boxed{\sigma}\ c_1$ aplicados a uma relação, mais o operador $\boxed{\boxtimes}\ c_j$ e a operação $\boxed{\pi}\ l_1$ final resultante da junção.

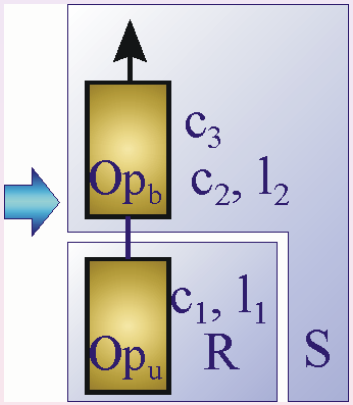
Métodos de Acesso para operações binárias

Idéia geral

- Note-se que um método de acesso físico sempre processa uma única relação, mesmo que envolva um operador binário, como a junção:
- Portanto uma junção é respondida tipicamente com dois operadores:







Métodos de Acesso para operações binárias

Idéia geral

- A execução de uma junção requer o uso de dois operadores que operam como se fossem dois *loops* aninhados:
  1 Para cada tupla t_R da relação R:
  2 Para cada tupla t_S da relação S:
  3 Se a condição de junção for verdadeira, escreva $< t_R, t_S >$ no resultado.
- A relação R é chamada externa ou da esquerda, e a relação S é chamada interna ou da direita para a junção.
- A tática usual é usar um método de acesso adequado para obter as tuplas necessárias da relação externa (o que corresponde ao primeiro passo do algoritmo),
- e para cada tupla obtida, obter as tuplas necessárias da relação interna (o que corresponde ao segundo passo do algoritmo).

Métodos de Acesso para operações binárias

Idéia geral

- O método usado para obter a relação externa pode ser qualquer método de acesso unário.
- O método usado para obter a relação interna deve ser um método de acesso binário.
- Caso existam outras junções a serem executadas em sequência, o resultado da primeira junção é tratado como o resultado do *loop* externo, e um método de acesso binário executa a segunda junção, e assim sucessivamente.

Operações Relacionais Binárias

Junção

- Na discussão a seguir, considere-se que a operação a ser executada é expressa como: $\boxed{\sigma_{(c_1)} R \overset{(c_{junc})}{\bowtie} \sigma_{(c_2)} S}$
- onde a condição de Junção (c_{junc}) é uma conjunção da forma: $(c_{junc})_1 \wedge (c_{junc})_2 \wedge \ldots \wedge (c_{junc})_n$,
  onde $(c_{junc})_i$ é uma comparação entre um atributo de cada relação, da forma: $\boxed{AtR_i \theta_i AtS_i}$, onde
  - AtR_i é um atributo da relação externa R,
  - AtS_i é um atributo da relação interna S,
  - AtR_i e AtS_i têm o mesmo domínio, e
  - θ_i é uma operação de comparação válida nesse domínio.
- As condições c_1 e c_2 são usadas nas operações de seleção, onde cada uma envolve apenas atributos das relações R e S respectivamente.

Métodos de Acesso para operações binárias

Neasted-loop Join

- O método de acesso para junção **Neasted-loop Join** é o método básico para executar um operador de junção.
  - Quer dizer, mesmo que uma consulta não consiga usar nenhum outro método de junção, o *Neasted-loop Join* sempre pode ser usado.
  - Ele não depende do uso de nenhum índice...,
  - ...mas pode usar qualquer índice disponível para acessar tanto a relação interna quanto a externa

Métodos de Acesso para operações binárias

Neasted-loop Join

- O *Neasted-loop Join* opera assim:
  1. ☞ Para cada tupla da relação externa R que atende à $\sigma_{(c_1)}R$, recupera os valores dos atributos t_{AtR} usados na junção;
  2. ☞ considera t_{AtR} como uma constante, e submete como critério de busca para o método de acesso usado para recuperar as tuplas necessárias da relação interna;
  3. ☞ para cada tupla recuperada de S, verifica se ela atende à $\sigma_{(c_2)}S$. Se atender, escreve o par $< t_R, t_S >$ no resultado.
- O processo de escolha das relações que devem ocupar o *loop* externo ou externo prioriza que a relação externa deve caber na memória disponível.
- Se ambas couberem, ou nenhuma couber, é indiferente qual assume cada posição.

Métodos de Acesso para operações binárias

Neasted-loop Join

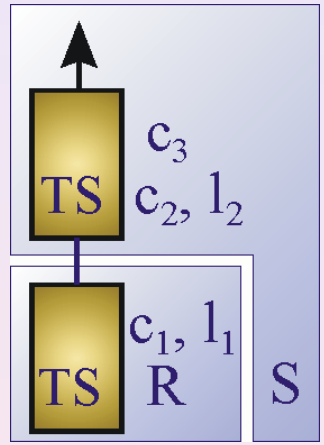
- Métodos de acesso unários podem ser combinados para resolver uma junção da forma $\sigma_{(c_1)}R \overset{(c_{junc})}{\bowtie} \sigma_{(c_2)}S$:
  - Acesso sequencial nas duas relações;
  - Acesso indexado na relação externa e sequencial na interna;
  - Acesso indexado na relação interna e sequencial na externa;
  - Acesso indexado nas duas relações;
- O método de acesso usado depende de haver índices adequados nas respectivas relações envolvidas.

Métodos de Acesso para operações binárias

Neasted-loop Join com acesso sequencial nas duas relações

☞ Acesso sequencial nas duas relações

- Primeiro deve ser lida a relação externa, de preferência armazenada inteira na memória, ao custo de $R(NExtent(R))S$.
- Conforme cada *extent* é lido, apenas as tuplas que atendem à condição c_1 são mantidas no `buffer`.



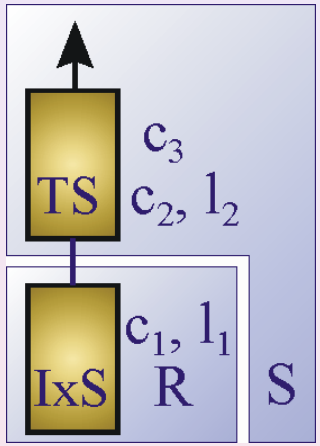
- A seguir, deve ser lida a relação interna, ao custo de $R(NExtent(S))S$. Apenas as tuplas que atendem à condição c_2 são mantidas no `buffer`.
- Note-se que mesmo que a relação externa não caiba na memória, o custo da leitura não se altera.
- As tuplas que atendem à condição c_3 são colocadas no resultado.

Métodos de Acesso para operações binárias

Neasted-loop Join com acesso indexado na relação externa e sequencial na interna

☞ Acesso indexado na relação externa e sequencial na interna

- Primeiro deve ser lida a relação externa, de preferência armazenada inteira na memória. Para isso, a condição c_1 deve existir. Supondo que haja um índice B-tree que atende a c_1, seu custo será
$R(H-1)R + R(NFolhas)S + R(N)T$.

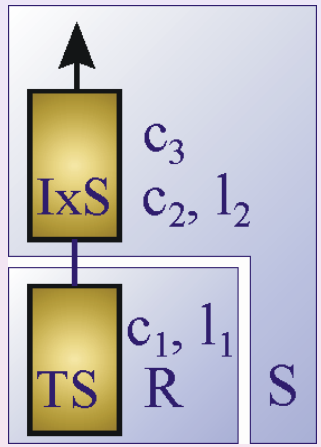


- A seguir deve ser lida a relação interna sequencialmente, ao custo de $R(NExtent(S))S$. Apenas as tuplas que atendem à condição c_2 são mantidas no `buffer`.
- As tuplas que atendem à condição c_3 são colocadas no resultado.

Métodos de Acesso para operações binárias

Neasted-loop Join com acesso sequencial na relação interna e indexado na externa

☞ Acesso sequencial na relação interna e indexado na externa



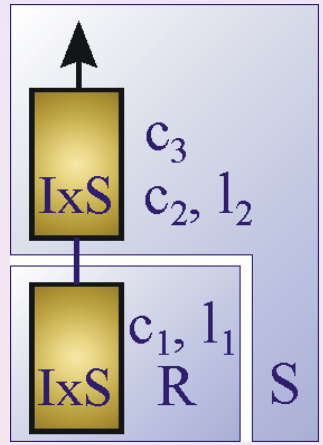
- Primeiro deve ser lida a relação externa, de preferência armazenada inteira na memória, ao custo de $R(NExtent(R))S$.
  Apenas as tuplas que atendem à condição c_1 são mantidas no `buffer`.
- A seguir deve ser lida a relação interna usando um índice adequado. Isso é feito analisando cada tupla da relação externa, e realizando a busca pelo predicado de junção c_3 como se fosse constante para cada operação de busca.
  - Supondo que haja um índice B-tree que atende a c_3, seu custo será $R(H-1)R + R(NFolhas)S + R(N)T$ vezes o número de tuplas da relação externa que atende à condição c_1.
  - Caso a condição c_2 exista, ela é usada como condição de *screening* da consulta.
  - As tuplas que atendem à condição c_3 são colocadas no resultado.

Métodos de Acesso para operações binárias

Neasted-loop Join com acesso indexado nas duas relações

☞ Acesso indexado nas duas relações

- Primeiro deve ser lida a relação externa, de preferência armazenada inteira na memória. Para isso, a condição c_1 deve existir. Supondo que haja um índice B-tree que atende a c_1, seu custo será $R(H-1)R + R(NFolhas)S + R(N)T$.



- A seguir deve ser lida a relação interna usando um índice adequado. Supondo que haja um índice B-tree que atende a c_3, seu custo será $R(H-1)R + R(NFolhas)S + R(N)T$ vezes o número de tuplas da relação externa que atende à condição c_1.
- Caso a condição c_2 exista, ela é usada como condição de *screening* da consulta.
- As tuplas que atendem à condição c_3 são colocadas no resultado.

Métodos de Acesso para operações binárias

Exemplo

Por exemplo, seja as relações já anteriormente definidas em Oracle:

```
CREATE TABLE Alunos (
    NUSP CHAR(10) PRIMARY KEY,
    Nome CHAR(40),
    Idade DECIMAL(3),
    Cidade CHAR(30) )
    STORAGE INITIAL 20K NEXT 20K
    PCTFREE 10 PCTUSED 70
```

```
CREATE TABLE Matriculas (
    Disciplina CHAR(8) NOT NULL,
    NUSP CHAR(10) NOT NULL,
    Nota DECIMAL(4,2),
    PRIMARY KEY (NUSP, Disciplina) )
    STORAGE INITIAL 20K NEXT 20K
    PCTFREE 10 PCTUSED 70
```

- Assumimos que existam 80.000 alunos, matriculados em 12.000 disciplinas, cada aluno se matriculando numa média de 8 disciplinas.
  Então haverá $80.000 * 8 = 640.000$ tuplas na relação `Matrículas`.
- O tamanho da página é de 2KBytes, portanto 2.000 bytes para dados, e cada *extent* inicial e de continuação ocupa 10 páginas.

Métodos de Acesso para operações binárias

Estatísticas para as relações usadas nos exemplos

Métricas para Relação:

Nome	RowSIZE	CARD	MaxCARD	NPags	NExtents
Alunos	74	80.000	80.000	3.810	381
Matriculas	22	640.000	640.000	5.689	569
...		...			

Métricas para Atributos:

Relação	Atrib	ColSIZE	ColCARD	Distr	Low2Key	high2Key
Alunos	NUSP	10	80.000	2	'0011223344'	'9988776655'
Alunos	Nome	40	79.500	2	'Abrão Abraão'	'Zuleica Penultima'
Alunos	Idade	2	50	2	17	80
Alunos	Cidade	30	700	2	'Adamantina'	'Zzy'
Matriculas	Disciplina	8	12.000	2	'AAA-1001'	'ZOO-8228'
Matriculas	NUSP	10	80.000	2	'0011223344'	'9988776655'
Matriculas	Nota	2	100	2	0,1	9,9
	...		...			

Métodos de Acesso para operações binárias

Estatísticas para as relações usadas nos exemplos

Métricas para Índices:

Relação	Índice	Type	H	NLeafs	Key1Card	KeyFullCard	ClusterRatio
Alunos	AlunosPK	B-tree	3	1.026	80.000	80.000	0
Matriculas	MatriculasPK	B-tree	3	8.422	80.000	640.000	0
	...				...		

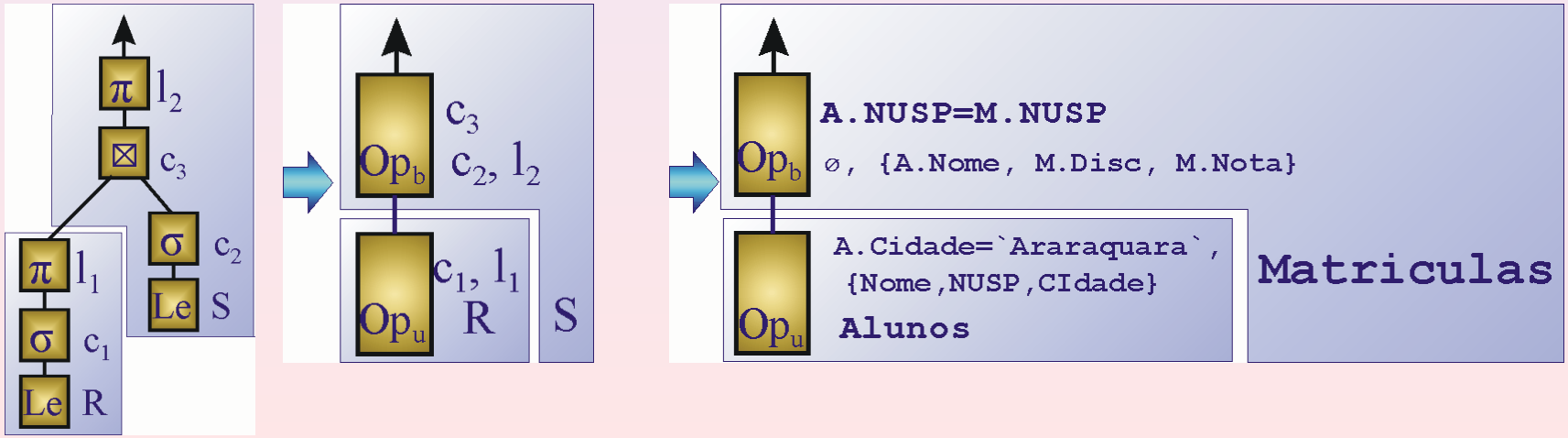
Métodos de Acesso para operações binárias

Neasted-loop Join – Exemplo Consulta 7

- Seja a consulta sobre as relações de alunos e de matrículas:
  Obtenha as notas de cada disciplina onde alunos de "Araraquara" se matricularam.

```
SELECT A.Nome, M.Disciplina, M.Nota
    FROM Alunos A, matriculas M
    WHERE A.Cidade='Araraquara' AND
        A.NUSP=M.NUSP;
```

- Ela gera a árvore de comandos: que pode ser empacotada como:



Métodos de Acesso para operações binárias

Neasted-loop Join – Exemplo Consulta 7

- As chaves primárias de `Alunos` e `Matriculas` são indexadas em B-tree: a junção tem a forma $\sigma_{(\texttt{A.Cidade='Araraquara'})}\texttt{Alunos} \overset{\texttt{A.NUSP=M.NUSP}}{\bowtie} \texttt{Matriculas}$:
  - Acesso sequencial nas duas relações:
    $R(NExtent(Alunos))S + R(NExtent(Matriculas))S = R3.810S + R5.689S$
    $= \frac{3.810}{1.000} + \frac{5.689}{1.000} = 9,5$ segundos.
  - Acesso indexado na relação externa e sequencial na interna:
    `Alunos`: $R(H-1)R + R(NFolhas)S + R(N)T + R(NExtent(Matriculas))S =$
    $R3R + R\frac{1.026}{700}S + R\frac{80.000}{700}T + R5.689S = \frac{3}{100} + \frac{2}{1.000} + \frac{114}{300} + \frac{5.689}{1.000} = 6,1$
    segundos.
  - Acesso indexado na relação interna e sequencial na externa:
    $R(NExtent(Alunos))S +$
    $Quantos(\texttt{A.Cidade='Araraquara'}) \cdot (R(H-1)R + R(NFolhas)S + R(N)T) =$
    $R3.810S + \frac{80.000}{700} \cdot (R3R + R\frac{8.422}{80.000}S + R\frac{640.000}{80.000}T) =$
    $\frac{3.810}{1.000} + \frac{80.000}{700} \cdot (\frac{3}{100} + \frac{1}{1.000} + \frac{8}{300}) = 3,81 + 114 \cdot (0,058) = 10,42$ segundos.

Métodos de Acesso para operações binárias

Neasted-loop Join – Exemplo Consulta 7

- As chaves primárias de `Alunos` e `Matriculas` são indexadas em B-tree: a junção tem a forma $\sigma_{(\texttt{A.Cidade='Araraquara'})}\texttt{Alunos} \overset{\texttt{A.NUSP=M.NUSP}}{\bowtie} \texttt{Matriculas}$:
  - Acesso indexado nas duas relações:

    $\texttt{Alunos}: \boxed{R(H-1)R} + \boxed{R(NFolhas)S} + \boxed{R(N)T} +$

    $Quantos(\texttt{A.Cidade='Araraquara'}) \cdot (\boxed{R(H-1)R} + \boxed{R(NFolhas)S} + \boxed{R(N)T}) =$

    $\boxed{R3R} + \boxed{R\frac{1.026}{700}S} + \boxed{R\frac{80.000}{700}T} + \frac{80.000}{700} \cdot (\boxed{R3R} + \boxed{R\frac{8.422}{80.000}S} + \boxed{R\frac{640.000}{80.000}T}) =$

    $\frac{3}{100} + \frac{2}{1.000} + \frac{114}{300} + 114 \cdot (\frac{3}{100} + \frac{1}{1.000} + \frac{8}{300}) = 0,412 + 114 \cdot 0,058 = 7,02$ segundos

Métodos de Acesso para operações binárias

Hybrid Join

- Uma causa importante de perda de desempenho do uso de índice na relação interna no *Neasted-loop Join* quando existem muitas tuplas na relação externa, é que a mesma página pode precisar ser lida para comparações com tuplas da relação externa distintas.
- o Hybrid Join elimina essa necessidade de leitura repetida, da seguinte maneira:
  - Quando o índice de busca da relação interna é usado, é feita apenas a leitura do segmento de índice, e não dos dados;
  - das folhas do índice, recuperam-se todas as *Rid-lists*, e ordena-se eliminando as duplicatas;
  - Além disso, as páginas mais acessadas do índice são naturalmente mantidas no *buffer* – por exemplo, a raiz e o primeiro nível de uma B-tree.

Métodos de Acesso para operações binárias

Hybrid Join

- Com isso, o custo de acesso da relação interna cai pela probabilidade de que uma mesma página da relação interna contenha chaves de duas tuplas da relação externa.
- Portanto, o custo de acesso da relação interna cai de $Card(R) \cdot (\ \mathrm{R}(H-1)\mathrm{R} + \mathrm{R}(NFolhas)\mathrm{S} + \mathrm{R}(N)\mathrm{T}\ )$ para $Card(R) \cdot (\ \mathrm{R}(H-3)\mathrm{R} + \mathrm{R}(NFolhas)\mathrm{S} + (1 - Sel^2(c_3)) \cdot \mathrm{R}(N)\mathrm{T}\ )$.

Métodos de Acesso para operações binárias

Merge Join

- O método *Merge Join* obtém as tuplas que atendem às condições c_1 e c_2 das duas relações, idealmente em memória, e ordena as duas relações pelos atributos do predicado de junção c_3. A seguir percorre as duas relações de maneira sincronizada.
- A consulta `SELECT * FROM R, S WHERE R.r1=k1 AND S.s2=k2 AND R.r3=S.s3;` usando o operador *Merge Join* é executada assim:
  1. `SELECT * FROM R WHERE R.r1=k1 ORDER BY R.r3 INTO TempR;`
  2. `SELECT * FROM S WHERE S.s2=k2 ORDER BY S.s3 INTO TempS;`
  3. Percorre sincronizadamente `TempR` e `TempS`, e quando `TempR.r3=TempS.s3` escreva $<t_{TempR}, t_{TempS}>$ no resultado.

Métodos de Acesso para operações binárias

Merge Join

- Percorrer sincronizadamente `TempR` e `TempS` significa fazer uma única varredura nas duas relações, comparando tupla a tupla.
- Sempre que `R.r3` $<$ `S.s3` avança o cursor de `R`, e sempre que `R.r3` $>$ `S.s3` avança o cursor de `S`.
- Quando `R.r3` $=$ `S.s3`, escreve $<t_{TempR}, t_{TempS}>$ no resultado e avança os dois cursores.
  - ☞ Se houver n_s tuplas com um mesmo valor na relação `R` e n_s tuplas com esse mesmo valor na relação `S`, devem ser geradas as $n_r \cdot n_s$ tuplas correspondentes ao cartesiano dessas tuplas.
- O custo de acesso da operação *Merge Join* é linear na soma $N + M$ das cardinalidades das relações, $N \cdot \log N + M \cdot \log M$ para o processamento da ordenação das relações e linear na comparação $N + M$ entre os atributos das duas relações,
  onde $N = Card(R)$ e $M = Card(S)$ (a menos da geração do cartesiano de valores repetidos, que em geral corresponde a muito poucas tuplas).

Métodos de Acesso para operações binárias

Merge Join

- Caso algumas das relações `R` ou `S` tenham um índice ordenado sobre o atributo de junção, esse índice é usado para obter as tuplas ordenadas e a etapa de ordenação é suprimida.
- Um índice com os atributos `<R.r1, R.r3>` sobre a relação `R`, ou com os atributos `<S.s2, S.s3>` sobre a relação `S` certamente agiliza muito uma junção com o operador *Merge Join*, pois permite que se use o atributo `<R.r1>` (ou `<S.s2>`) para a busca da relação temporária e já obtém as tuplas ordenadas.

Métodos de Acesso para operações binárias

Hash Join

- O método *Hash Join* obtém as tuplas que atendem às condições c_1 e c_2 em duas relações `TempR`, e `TempS`, idealmente em memória, e cria uma estrutura *hash* para a relação `TempR` para o atributo de junção.
- A seguir, percorre a relação `TempS`, usando o atributo de junção como chave de busca na relação `TempR`. Sempre que uma tupla com valor igual for achado, escreve $<t_{TempR}, t_{TempS}>$ no resultado.
  - ☞ Note-se que se houver n_r tuplas com um mesmo valor na relação `TempR`, elas estarão acessíveis pela mesma chave de *hash*, e todas devem ser escritas no resultado.
  - ☞ Se houver n_s tuplas com um mesmo valor na relação `TempS`, mesmo que fora de ordem, elas irão naturalmente gerar as tuplas corretas no resultado.

Métodos de Acesso para operações binárias

Hash Join

- O custo do *Hash Join* tende a ser menor do que o do *Merge Join* sempre que houver a necessidade de ordenar as tuplas e é possível colocar toda a relação `TempR` em estrutura *hash* em memória.
- Na realidade, o método *Hash Join* é a estrutura preferida quando a junção não pode contar com estruturas de índice de apoio nas relações-base.

Como analisar os planos de consulta gerados pelo SGBD

Postgres

SQL provê o comando `EXPLAIN query` para que o analista possa verificar como cada comando SQL é transformado no plano físico de acesso.

EXPLAIN query – Postgres

```
EXPLAIN [ANALYZE] <comando>
```

- `<comando>` pode ser qualquer comando da DML.
- O comando não é executado, a menos que a cláusula `ANALYZE` seja solicitada.
- O comando `EXPLAIN` mostra o plano de acesso escolhido de acordo com as estatísticas atuais.

Como analisar os planos de consulta gerados pelo SGBD

Postgres

Por exemplo:

```
EXPLAIN SELECT * FROM Alunos;
```

```
                QUERY PLAN
-------------------------------------------------------
Seq Scan on Alunos (cost=0.00..381.00 rows=80000 width=72)
```

- Este resultado indica que o plano é constituído apenas do método 'Seq Scan' sobre a relação Alunos,
- não precisa esperar nada para começar a obter resultados, custa 381 unidades de tempo,
- e estima-se que serão obtidas 80.000 tuplas com média de 72 bytes cada.

Como analisar os planos de consulta gerados pelo SGBD

Oracle

SQL provê o comando `EXPLAIN query` para que o analista possa verificar como cada comando SQL é transformado no plano físico de acesso.

EXPLAIN query – Oracle

```
EXPLAIN PLAN
     [SET STATEMENT_ID = <valor>]
     [INTO <relacao>]
     FOR <comando>;
```

- `<comando>` pode ser qualquer comando da DML.
- Se a cláusula `SET STATEMENT_ID` não for colocada, o valor usado é `null`.
- Se a cláusula `INTO <relacao>` não for colocada, é usada a tabela `PLAN_TABLE` do esquema corrente.

Como analisar os planos de consulta gerados pelo SGBD

Oracle

Por exemplo:

```
EXPLAIN PLAN SET STATEMENT_ID ='Primeiro' FOR
    SELECT * FROM Alunos;

SELECT id, operation, options, object_name, parent_id,
       cost, bytes
    FROM plan_table
    WHERE statement_id = 'Primeiro'
    ORDER BY id;
```

```
  ID OPERATION    OPTIONS OBJECT_NAME PARENT_ID COST     BYTES
-------------------------------------------------------------
   0 SELECT STATEMENT                         2 5.760.000
   1 TABLE ACCESS FULL    Alunos              0    2 5.760.000
```

Como analisar os planos de consulta gerados pelo SGBD

Oracle

```
  ID OPERATION    OPTIONS OBJECT_NAME PARENT_ID COST     BYTES
-------------------------------------------------------------
   0 SELECT STATEMENT                         2 5.760.000
   1 TABLE ACCESS FULL    Alunos              0    2 5.760.000
```

- Este resultado indica que o plano é constituído apenas do método 'TABLE ACCESS' sobre a relação Alunos, lida inteira (FULL),
- estima-se que ele custa 2 unidades de tempo,
- e serão obtidas tuplas de 5.760.000 bytes cada.

☞ Diversos *scripts* para formatação de resultados na tabela PLAN_TABLE estão disponíveis em *Oracle*, como por exemplo UTLPLS.SQL.

☞ O *script* UTLXPLAN.SQL também é útil para criar uma tabela com a estrutura da PLAN_TABLE.

Arquitetura de SGBD Relacionais
— Métodos de Acesso Físico —

Caetano Traina Jr.

Grupo de Bases de Dados e Imagens
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo - São Carlos
caetano@icmc.usp.br

18 de maio de 2011
São Carlos, SP - Brasil

FIM