CI218: Avaliação de Consultas

Profa. Carmem Hara

Departamento de Informática/UFPR

10 de fevereiro de 2023

Catalogo do Sistema (ou Dicionario de Dados)

Contém informações sobre o SGBD e das bases de dados (metadados).

- do SGBD: tamanho do buffer, tamanho da página
- de cada tabela: nome da tabela, nome do arquivo, estrutura do arquivo
- de cada atributo: nome, tipo, ordem
- de cada índice: nome, estrutura, chave de pesquisa
- de cada visão: nome e definição
- restrições de integridade
- informações sobre autorização de acesso

Estatísticas Armazenadas no Catálogo

- `NTuplas(R)`: cardinalidade da relação R
- `NPaginas(R)`: tamanho da relação em páginas
- `NChaves(I)`: cardinalidade do índice - quantidade de valores distintos da chave de pesquisa
- `INPaginas(I)`: tamanho do índice em páginas
  para árvores B+: número de páginas folha
- `IAltura(I)`: altura da árvore B+
- `IMenor(I)` e `IMaior(I)`: domínio do índice - menor e maior valores da chave de pesquisa

Armazenamento do Catálogo

As informações do catálogo são também armazenadas em tabelas.

Tabela Cat_atributos

atrib_nome	rel_nome	tipo	posição
atrib_nome	Cat_atributos	string	1
rel_nome	Cat_atributos	string	2
tipo	Cat_atributos	string	3
posição	Cat_atributos	integer	4
idPiloto	Reserva	integer	1
idAviao	Reserva	integer	2
data	Reserva	datas	3
nomeResp	Reserva	string	4
idPiloto	Piloto	integer	1
nome	Piloto	string	2
idade	Piloto	integer	3
nivel	Piloto	integer	4

```
Reserva( idPiloto, idAviao, data, nomeResp)
Piloto ( idPiloto, nome, idade, nivel)
```

Avaliação de Operador

- A escolha do algoritmo depende de: tamanho das tabelas, existência de índices, ordenação das tuplas, tamanho do buffer, política de reposição do buffer
- Técnicas para desenvolver os algoritmos
  - **Indexação:** através da utilização de um índice
  - **Iteração:** recuperação de todas as tuplas da relação, uma por uma
  - **Particionamento:** agrupamento de tuplas por uma chave de ordenação, geralmente através de hash ou algoritmo de ordenação

Caminho de Acesso

- É uma forma de recuperar tuplas de uma relação: varredura ou índice.
- Um índice **casa** ou **satisfaz** uma condição se puder se usada para recuperar apenas as tuplas que satisfaçam a condição.
- Para uma condição na forma normal conjuntiva (CNF), onde cada termo está na forma "atrib op valor"(op $=<, \leq, =, \neq, >, \geq$):
  - índice hash: casa com a condição se todos os termos forem da forma "atrib $=$ valor"e todos os atributos da condição fazem parte da chave de pesquisa do índice
  - índice árvore B+: casa com a condição se os atributos nos termos formam um prefixo da chave de pesquisa
    Exemplo: $< a >$ e $< a, b >$ são prefixos da chave $< a, b, c >$, mas $< a, c >$ e $< b, c >$ não são.

Exemplos

- Chave de pesquisa (`nomeResp, idAviao, idPiloto`)
  - Condição:
    $nomeResp =' Jose'$ & $idAviao = 5$ & $idPiloto = 3 \Longrightarrow$ Hash casa, árvore B+ casa
  - Condição: $nomeResp =' Jose'$ & $idAviao = 5 \Longrightarrow$ Hash não casa, árvore B+ casa
- Chave de pesquisa (`idAviao, idPiloto`)
  - Condição:
    $nomeResp =' Jose'$ & $idAviao = 5$ & $idPiloto = 3 \Longrightarrow$ Hash e árvore B+ casam, mas tem que filtrar por `nomeResp`

Seletividade do método de acesso

- Corresponde ao número do páginas (de índice e de dados) recuperadas para obter todas as tuplas desejadas.
- O método **mais seletivo** é aquele que recupera o menor número de páginas.
- Exemplo:
  - Índice hash na tabela `Reserva` com chave de pesquisa `(idAviao, idPiloto)`
  - Condição: $idAviao = 100$ & $idPiloto = 5$
  - Seletividade = NPaginas(Reserva) * (1 / NChaves(H))
- O **fator de redução** é a fração do número de tuplas de uma relação que satisfazem uma determinada condição.
- Exemplo:
  - Índice árvore B+ (A) em `Reserva` com chave `(data)`
  - Condição: $data > 16/10/2005$
  - Fator de redução = $\frac{IMaior(A) - 16/10/2005}{IMaior(A) - IMenor(A)}$

Algoritmos para Operações Relacionais

Esquema exemplo:

```
Reserva( idPiloto, idAviao, data, nomeResp)
         ----------------------
Piloto ( idPiloto, nome, idade, nivel)
         --------
```

Base de dados:

- Reserva: cada página contém 100 tuplas, com total de 1000 páginas (ou seja, total de 100.000 tuplas)
- Piloto: cada página contém 80 tuplas, com total de 500 páginas (ou seja, total de 40.000 tuplas)

Seleção

- Condição: $\sigma_{nome<'C\%'}(Piloto)$
- Sem índice sobre o atributo nome: scan da relação Piloto
- Com índice B+ sobre o atributo nome
  - Existem 40.000 tuplas em Piloto (ou 500 páginas), com distribuição uniforme de valores: aprox. 10% das tuplas estão no resultado.
  - quantidade de I/O com o índice clusterizado: 50
  - quantidade de I/O com o índice não clusterizado: 4.000 no pior caso
  - Regra geral: se o número de tuplas a serem recuperadas é maior que 5% do total é melhor fazer um scan sobre a relação do que utilizar um índice não clusterizado.

Seleção (cont.)

- Condição com disjunção
  - scan da relação, união de resultados (com ou sem ordenação da entradas por id da página)
  - transformação em união de consultas
  - se forem termos sobre o mesmo atributo transformação em uma condição envolvendo IN ou bitmaps

Projeção

- pode ser implementada fazendo uma varredura da relação ou de um índice, caso exista um que contenha todos os atributos da operação
- a parte com maior custo é a remoção de duplicações.
  - implementação usando particionamento com hashing apropriado quando o tamanho do buffer não é muito pequeno com relação ao tamanho da relação (cada partição deve caber no buffer) e a distribuição com a função hash é relativamente uniforme
  - implmentação com ordenação
    a operação pode ser otimizada combinando a primeira passada do algoritmos de ordenação com a eliminação dos atributos que não estão no resultado da projeção
- se existir um índice com os atributos da projeção basta recuperar todas as entradas do índice. Eliminar duplicações é fácil porque elas estão em posições adjacentes no índice.

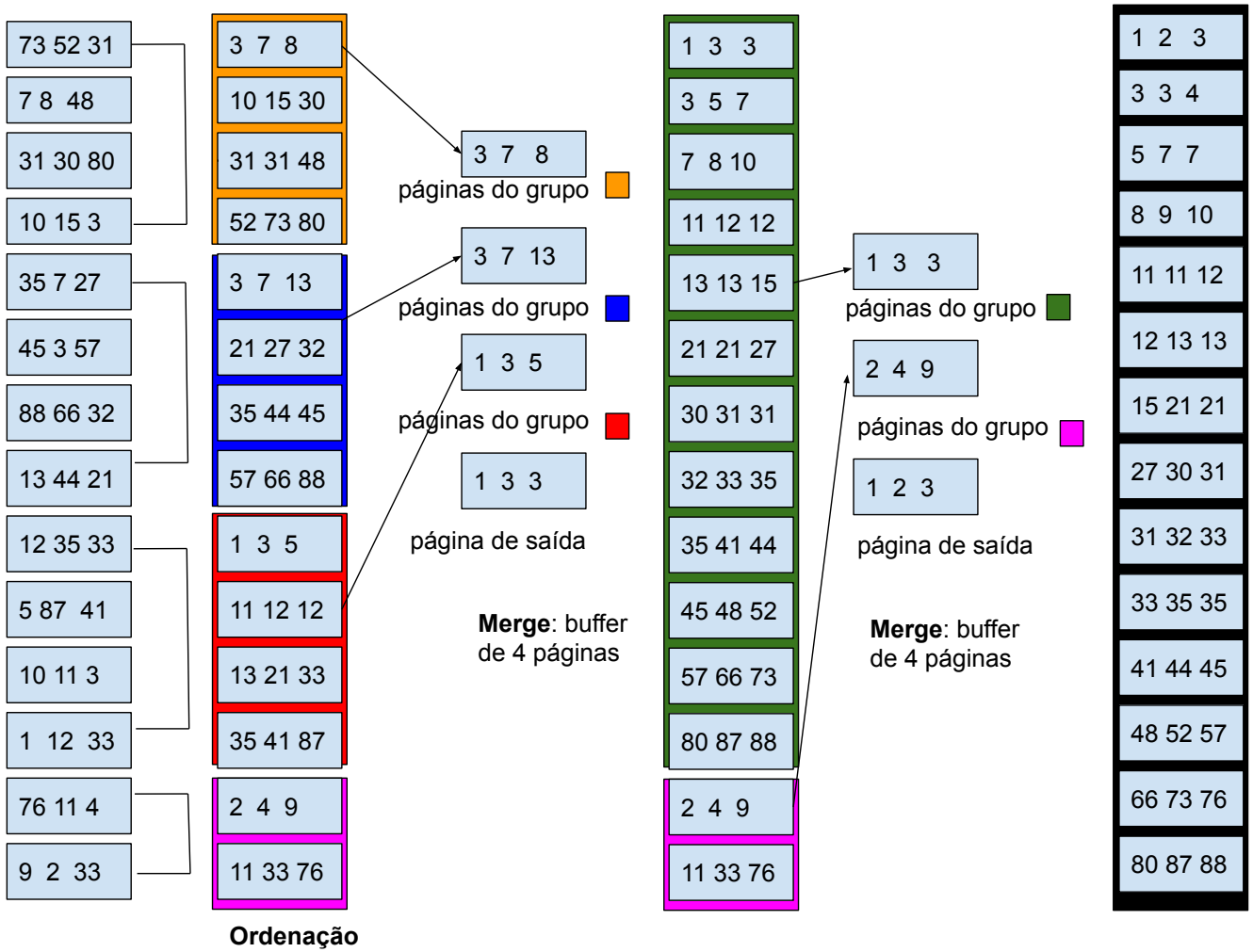
Junção

- Geralmente a operação mais cara de ser realizada
- Exemplo: *Reserva* $\bowtie$ *Piloto*
- **Alternativa 1**: junção com loop aninhado indexado (*index nested loop*)
  - Índice hash sobre `idPiloto` na relação `Piloto`:
  - Custo: 221.000 E/S
    - 1000 E/S para fazer varredura de Reserva
    - 1.2 * 100.000 E/S para busca utilizando hash (1.2 é o custo típico para tabelas hash)
    - 1 * 100.000 E/S para busca do registro de dados (se o índice não for clusterizado)

Reserva: 1000 páginas * 100 tuplas/página = 100.000 tuplas
Piloto: 500 páginas * 80 tuplas/página = de 40.000 tuplas

Alternativa 2: Junção sort-merge

Envolve ordenação das relações envolvidas na junção



Alternativa 2: Junção sort-merge

- primeiro ordena cada uma das relações e depois faz o merge
- Exemplo: buffer de 100 páginas disponíveis
- Custo: 7500 E/S
  - ordenação de `Reserva`:
    - Passo 1: 1000 páginas Reserva / 100 páginas de buffer = 10 grupos de 100 páginas ordenadas
    - Passo 2: merge de 10 grupos ordenados
    - Em cada passo há leitura e gravação das páginas de Reserva custo= 2 * 1000 páginas * 2 passos = 4000 E/S
  - ordenação de `Piloto`:
    - Passo 1: 500 páginas Piloto / 100 de buffer = 5 grupos de 100 páginas ordenadas
    - Passo 2: merge de 5 grupos ordenados
    - Em cada passo há leitura e gravação de 500 páginas custo = 2 * 500 páginas * 2 passos = 2000 E/S
  - junção de `Reserva` e `Piloto`
    1000 páginas de Reserva + 500 páginas de Piloto = 1500
  - custo total = 4000 + 2000 + 1500 = 7500 E/S

Custo do Sort-Merge

$$2 * n * (log_{b-1}(\frac{n}{b} + 1)$$

- custo em termos de quantidade de E/S
- n é o número de páginas da relação
- b é a quantidade de páginas do buffer
- $2 * n$ porque cada passo le e escreve todas as páginas
- custo do primeiro passo: $2 * n$ (daí vem o $+1$)
- o número de passos é $log_{b-1}(n/b)$ porque depois do primeiro passo existem n / b conjuntos ordenados e em cada subsequente são juntados (b-1) conjuntos porque 1 pagina é a de saída.

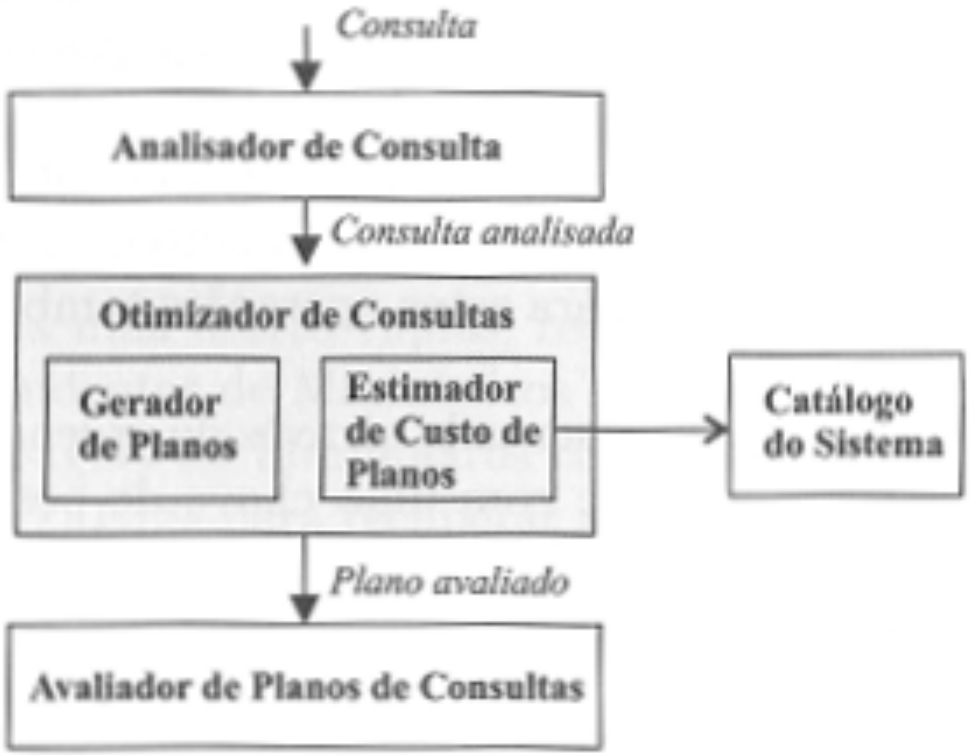
Junção aninhada indexada X Junção sort-merge

- o custo da junção sort-merge em geral é bem menor do que a junção aninhada indexada
- vantagem da junção aninhada indexada: é incremental
  - Se existir alguma seleção sobre a relação `Reserva` antes de fazer a junção, dá para evitar o processamento da junção de toda a relação.
  - Exemplo: dados dos pilotos que reservaram o avião com idAviao $= 501$
    1. seleção na tabela `Reserva` com condição $idAviao = 501$
    2. buscar na relação `Piloto` utilizando o índice

Outras Operações

- **group by, having**:
  - utiliza ordenação
  - se existir um índice com os atributos de agrupamento, pode utilizar o índice para recurar as tuplas, sem precisar da ordenação
- **operações de conjunto** (união, diferença e interseção)
  - maior custo é a eliminação de duplicações como na projeção

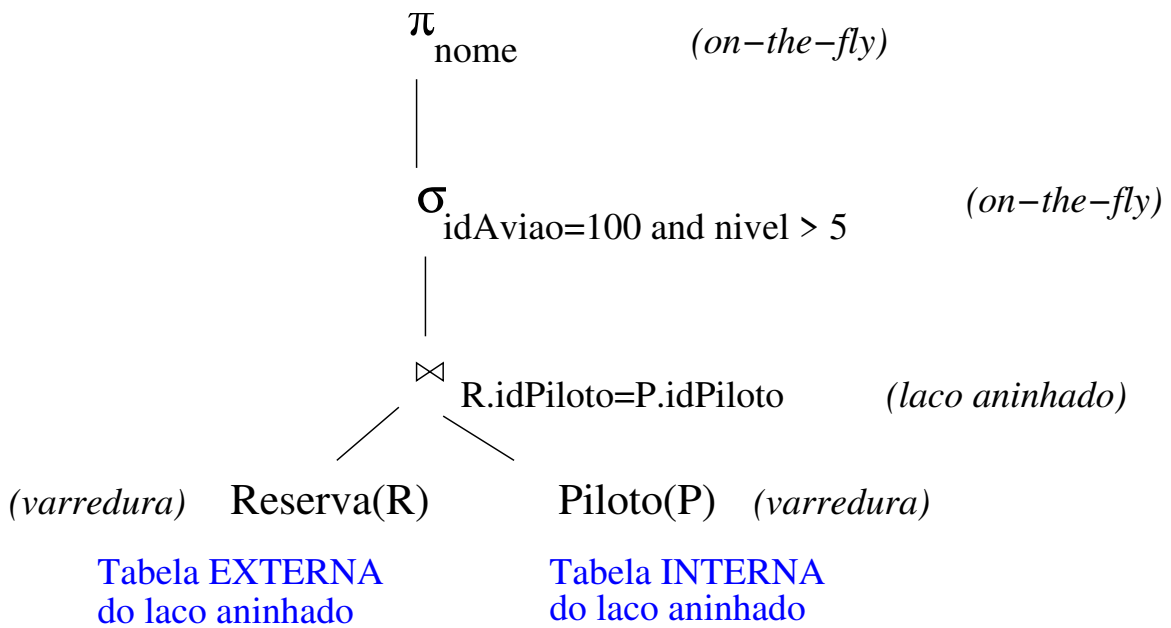
Otimização de Consultas



- o otimizador faz o planejamento da estratégia de execucao
- obter a estratégia ótima pode consumir muito tempo
- um plano de execução de consulta é uma árvore de álgebra relacional, onde cada nodo é anotado ou com o método de acesso para acesso a uma relação ou com o algoritmo utilizado para executar a operação da álgebra relacional.

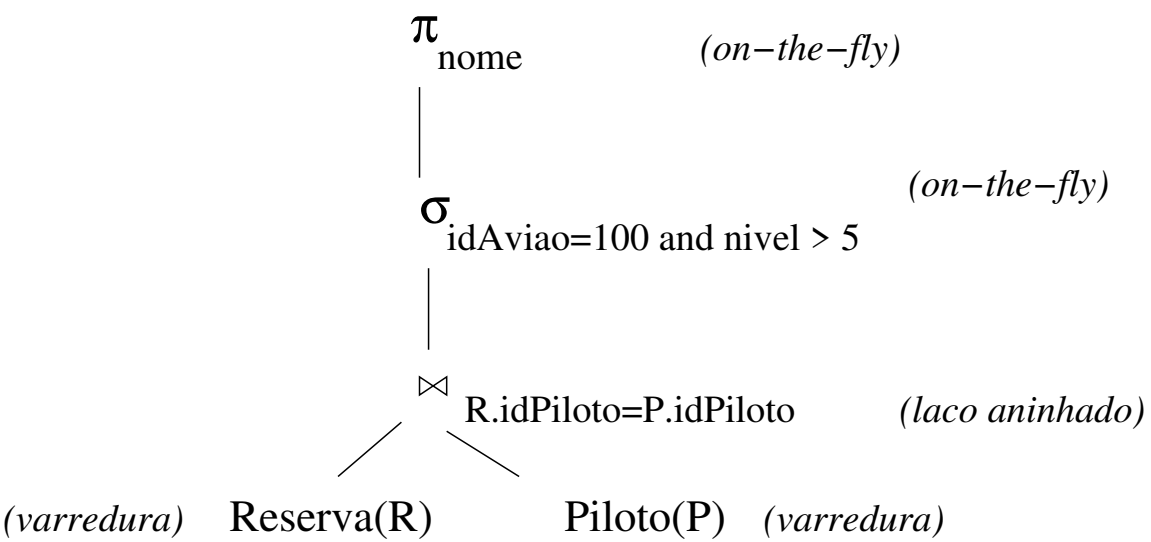
Plano de Avaliação de Consultas

```
select p.nome
rom Reserva r, Piloto p
where r.idPiloto = p.idPiloto and r.idAviao = 100 and p.nivel > 5
```



- *on-the-fly:* pipeline do resultado da operação abaixo na árvore. Caso contrário, o resultado da operação abaixo é primeiro MATERIALIZADA.

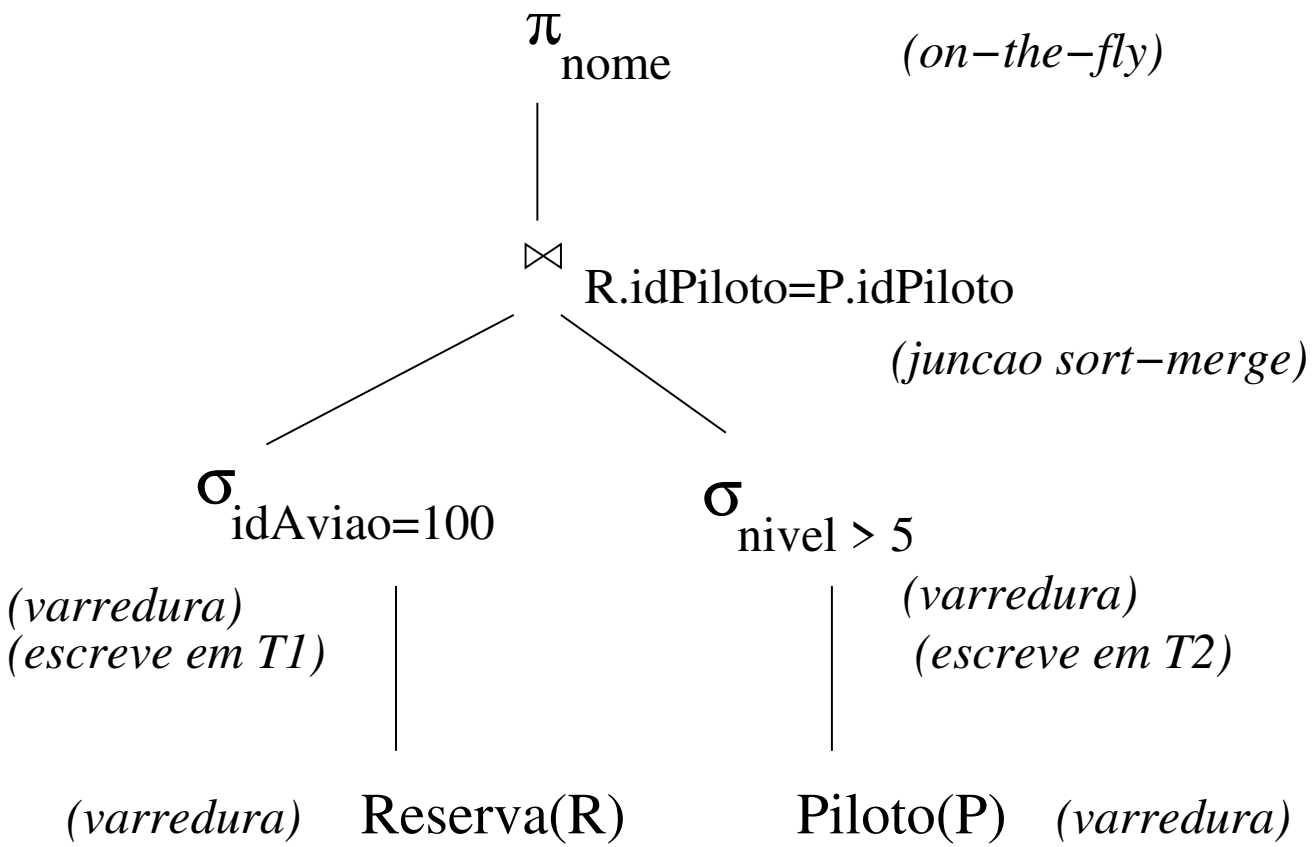
Plano de Avaliação de Consultas



Custo do plano:

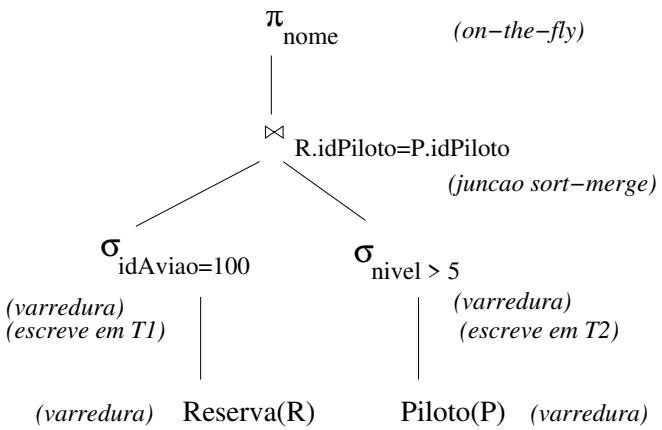
- leitura de 1000 páginas de Reserva
- 1000 * 500: para cada página de Reserva, lê-se todas as páginas de Piloto
- Total = 501.000 E/S

Alternativa 1: Empurrando Seleções



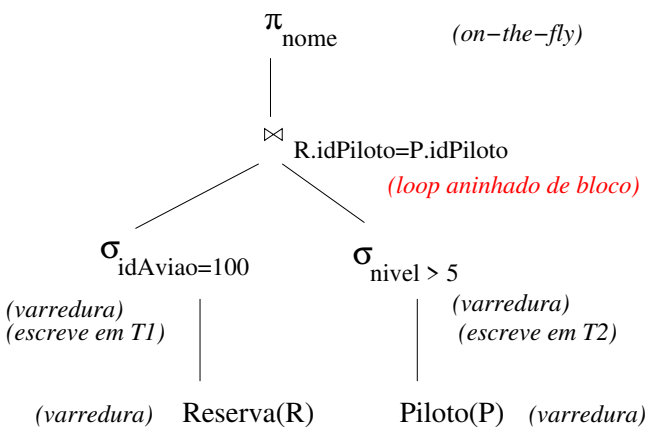
- Objetivo: diminuir o tamanho das tabelas antes da junção

Alternativa 1: Empurrando Seleções - Custo



- Suposição: existem 100 aviões e 10 níveis de piloto, com distribuição uniforme, buffer de 5 páginas
- Custo:
  - Scan de Reserva com seleção: 1000 (entrada) + 10 (saida)
  - Scan de Piloto com seleção: 500 (entrada) + 250 (saida)
  - sort-merge:
    - ordenação de T1 (em dois passos): 2 * 2 * 10 = 40
    - ordenação de T2 (em quatro passos): 2 * 4 * 250 = 2000
    - merge: 10 + 250 = 260
  - Total = 4060 E/S

Alternativa 2 - Loop Aninhado de Bloco



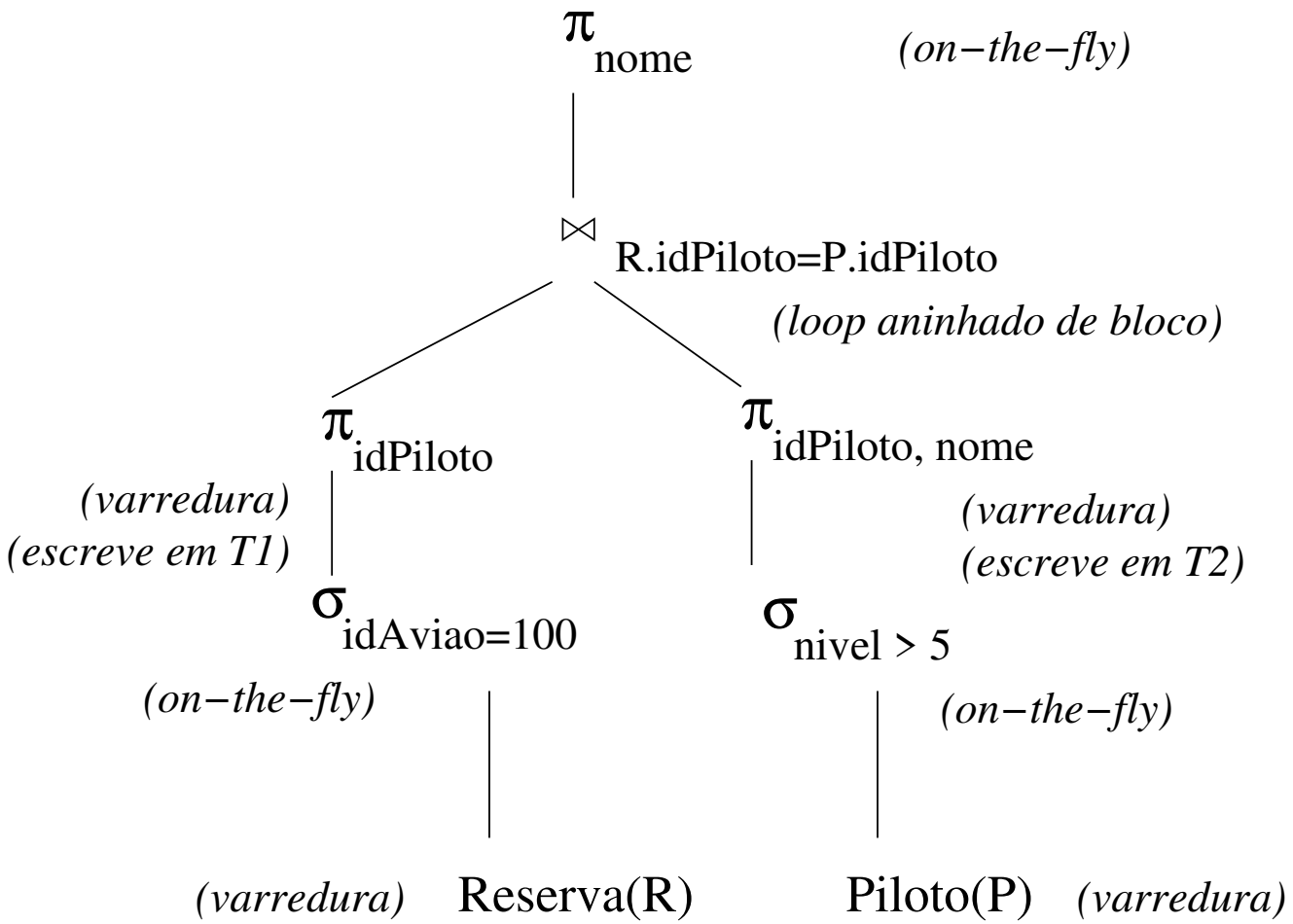
No buffer de 5 páginas:

- 3 páginas para ler T1
- 1 página para ler T2
- 1 página de saída

Custo:

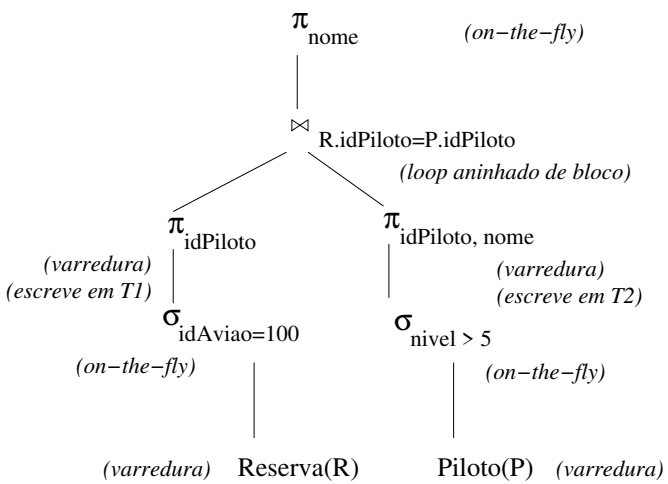
- Scan de Reserva com seleção: 1000 (entrada) + 10 (saida)
- Scan de Piloto com seleção: 500 (entrada) + 250 (saida)
- Junção: loop aninhado de bloco
  - Leitura de T1: 10
  - Junção com T2: (10 páginas de T1 / 3 páginas do buffer) * 250 páginas de T2
- Total= 2770 E/S

Alternativa 3 - Empurrando Projeções



- Objetivo: diminuir o tamanho das tabelas antes da junção

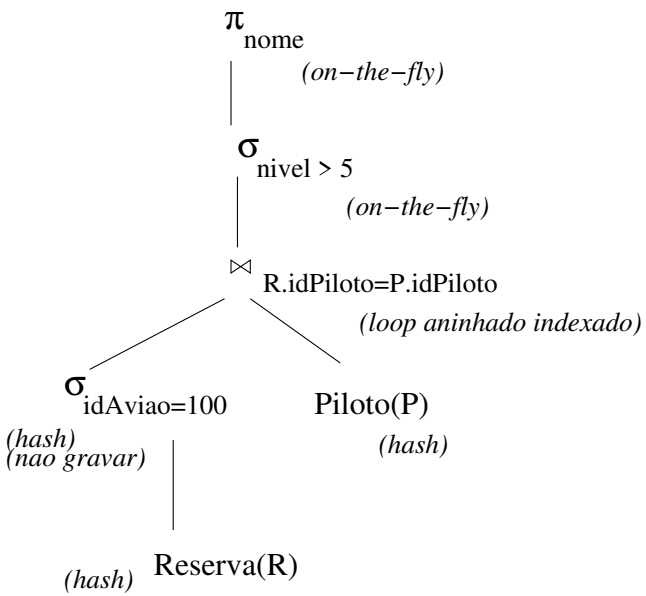
Alternativa 3: Empurrando Projeções - Custo



Custo:

- Scan de Reserva com seleção e projeção: 1000 (entrada) + 3 (saida)
- Scan de Piloto com seleção e projeção: 500 (entrada) + 240 (saida)
- Junção com loop aninhado de bloco: 3 (leitura de reserva + 240 (leitura de piloto) = 243
- Total = 1003 + 740 + 243 = 1986 E/S

Alternativa 4 - Usando Índices

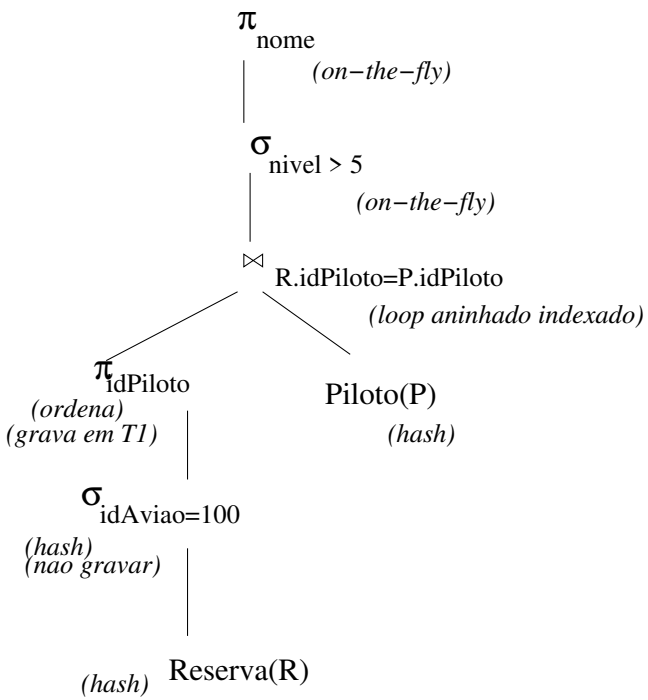


- Reserva: índice hash clusterizado em idAviao
- Piloto: índice hash em idPiloto

Custo:

- Seleção em Reserva: 10 páginas
  100.000 tuplas / 100 aviões = 1000 tuplas, 100 tuplas/página (clusterizado)
- Junção:
  - (Alternativa 1 de armazenamento)
    1000 tuplas * 1.2
    Total = 1200+10 = 1210
  - (Alternativas 2 e 3): 1000* (1.2 + 1)
    Total = 2200+10 = 2210

Alternativa 4 - Usando Índices



Possíveis melhorias:

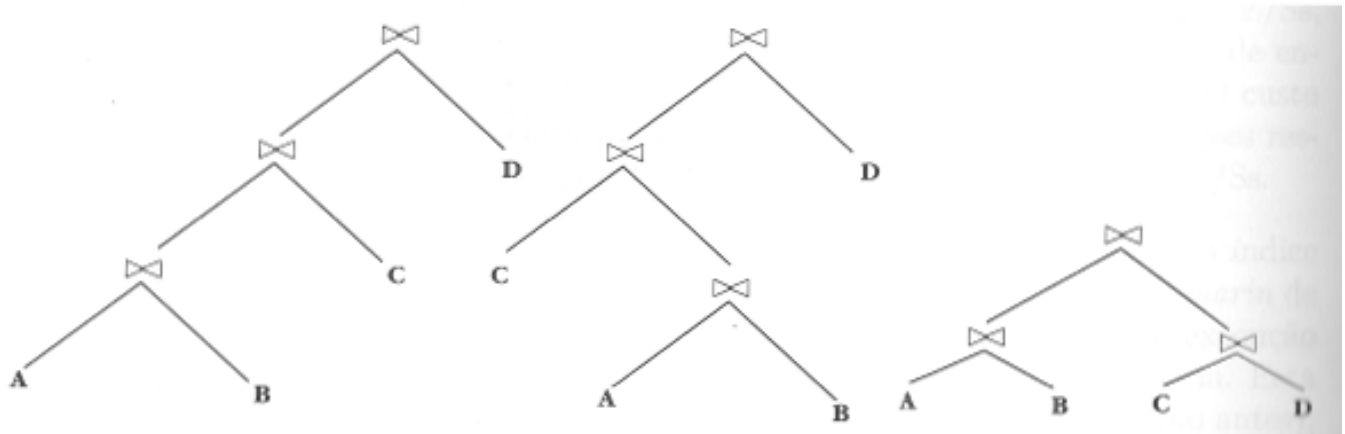
- Empurrar projeção
- Gravar resultado da seleção em T1 (3 páginas de saída)
- Ordenar T1 por idPiloto (1 passo com buffer de 5 páginas)
- Cada piloto é recuperado uma vez da tabela Piloto
- Custo: 516 E/S
  - leitura de Reserva: 10
  - escrita de T1: 3
  - ordenação de T1: 3
  - leitura de Piloto = 500

Funcionamento de um Otimizador de Consultas Típico

- Enumeração de planos de consultas alternativos que sejam equivalentes. Utiliza equivalências de expressões da álgebra relacional:
  - seleções e produto cartesiano podem ser combinados em junções
  - junções podem ser reordenadas
  - empurrando seleções e projeções para diminuir o tamanho das relações

Planos de Profundidade à Esquerda

- Em geral os otimizadores de consulta só consideram planos de profundidade à esquerda para fazer as junções
  - diminui o espaço de busca
  - permite que a consulta seja toda executada utilizando pipeline (*fully pipelined plans*)
- $A \bowtie B \bowtie C \bowtie D$



Referência

Os slides são baseados no Cap. 12 do livro "Sistemas de Gerenciamento de Banco de Dados", Terceira Edição, de Raghu Ramakrishnan & Johannes Gehrke