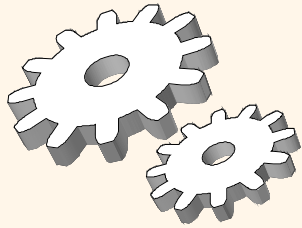
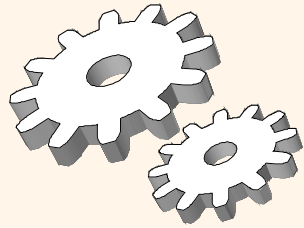


# *Resumo de Avaliação de Consultas*

## Capítulo 12

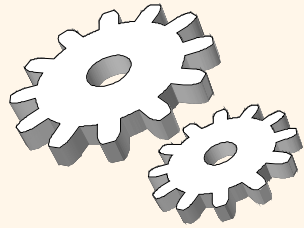


# *Esquema para Exemplos*

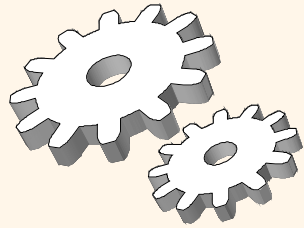
Sailors (*sid*: integer, *sname*: string, *rating*: integer, *age*: real)
Reserves (*sid*: integer, *bid*: integer, *day*: dates, *rname*: string)

- Similar ao esquema anterior; *rname* adicionado para variações.
- Reserves:
  - Cada tupla tem 40 bytes, 100 tuplas por página, 1000 págs.
- Sailors:
  - Cada tupla tem 50 bytes, 80 tuplas por página, 500 págs.

# *Visão Geral Sobre Realização de Consultas*



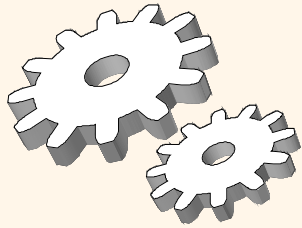
- *Plano: Árvore de Operadores da A.R., com a escolha do algoritmo para cada operador.*
  - Cada operador é tipicamente implementado usando-se uma interface: quando um operador é requisitado pelas próximas tuplas de saída, tais tuplas são computadas a partir de dados de entrada com um interface específica.
- Duas questões fundamentais na otimização de consultas:
  - Dada uma consulta, quais planos serão considerados?
    - Algoritmo para buscar o espaço de planos para o plano mais barato (estimado).
  - Como estimar o custo de um plano?
- Gostariamos: Achar o melhor plano. Na prática: Evitar os piores planos!
- Estudaremos as idéias utilizadas no Sistema R da IBM.



# *Algumas Técnicas Comuns*

- Os algoritmos para avaliação de operadores relacionais usam idéias simples extensivamente:
  - Indexação: Pode-se usar as condições da cláusula WHERE para se obter um menor conjunto de tuplas.
  - Iteração: Percorrer todas as tuplas da relação. (Às vezes podemos percorrer os campos nos índices ao invés de na própria tabela.)
  - Particionamento: Utilizando ordenação ou hashing, podemos particinonar as tuplas de entrada e trocar uma operação cara por operações similares em conjuntos de entrada menores.

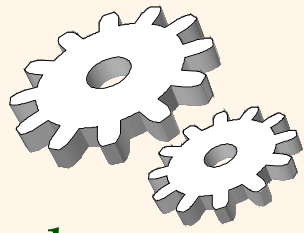
** Vamos ver estas técnicas enquanto discutimos consultas!*



# *Catálogo e Estatísticas*

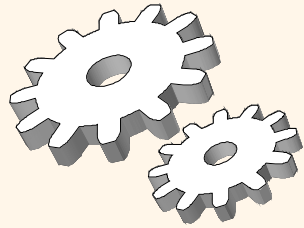
- Precisamos de informações sobre as relações e indexações envolvidas. ***Catálogos*** tipicamente contém pelo menos:
  - # tuplas (NTuples) e # páginas (NPages) para cada relação.
  - # valores distintos de chaves (NKeys) e NPages para cada índice.
  - Altura do índice, menor/maior valor de chaves (Low/High) para cada índice em árvore.
- Catálogos são atualizados periodicamente.
  - Atualizações sob todas alterações são muito caras; usa-se aproximações. Desta forma um há um pouco de inconsistência.
- Existem informações mais detalhadas (ex: histogramas dos valores de algum campo) que podem ser armazenadas.

# *Caminhos de Acesso*



- Um caminho de acesso é um método de recuperação de tuplas:
  - varredura de arquivo, ou índice que casam com uma seleção dada na consulta.
- Um índice em árvore *casa* com (uma conjunção de) termos que envolvem somente atributos em um *prefixo* da chave de busca do índice.
  - Ex: Índice em árvore em <*a, b, c*> casa com a seleção *a=5 AND b=3*, e *a=5 AND b>6*, mas não *b=3*.
- Um índice em hash *casa* com (uma conjunção de) termos que têm *atributo=valor* para todo atributo na chave de busca do índice.
  - Ex: Índice em <*a, b, c*> casa *a=5 AND b=3 AND c=5*; mas não *b=3, ou a=5 AND b=3, ou a>5 AND b=3 AND c=5*.

# *Uma Nota em Seleções Complexas*



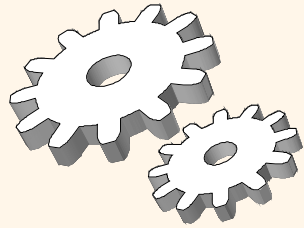
*(day<8/9/94 AND rname='Paul') OR bid=5 OR sid=3*

- Seleções são primeiramente convertidas para *forma normal conjuntiva* (FNC):

  *(day<8/9/94 OR bid=5 OR sid=3 ) AND*
  *(rname='Paul' OR bid=5 OR sid=3)*

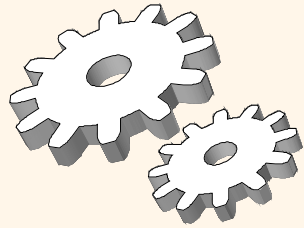
- Discutimos apenas casos sem ORs; consulte o texto para curiosidades sobre o caso mais geral.

# *Um método para realizar Seleções*



- Ache o *caminho de acesso mais seletivo*, usando-o para recuperar tuplas, e aplique os termos remanescentes que não casam com o índice:
  - *Caminho de acesso mais seletivo:* Um índice ou busca de arquivo que estimamos ter o menor número de E/S de páginas.
  - Os termos que casam com o índice reduzem o número de tuplas recuperadas; outros termos (não indexados) são usados para descartar mais algumas destas tuplas, mas não alteram o número de tuplas/páginas consultadas.
  - Considere *day<8/9/94 AND bid=5 AND sid=3*. Uma árvore B+ indexada no dia pode ser usada; então, *bid=5* e *sid=3* deve ser checado para cada tupla recuperada. Da mesma forma, uma hash indexada em <*bid, sid*> pode ser usada; mas *day<8/9/94* deve checado posteriormente.

# *Usando um Índice para Seleções*

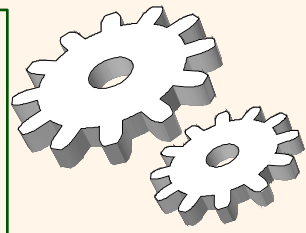


- Custo depende do #tuplas qualificadas, e do uso de *clustering*.
  - Custo para achar dados de entrada qualificados (tipicamente pequeno) mais o custo de recuperar registros (pode ser alto sem clustering).
  - Por exemplo, assumindo distribuição uniforme de nomes, com 10% de tuplas qualificadas (100 paginas, 10000 tuplas). Com um índice em cluster, o custo é pouco mais que 100 I/Os; se não é, teremos 10000 I/Os!

```
SELECT  *
FROM    Reserves R
WHERE   R.rname < 'C%'
```

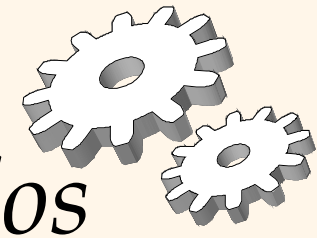
# *Projeção*

```
SELECT   DISTINCT
         R.sid, R.bid
FROM     Reserves R
```



- A parte cara é a remoção de duplicatas.
  - Sistemas SQL não removem duplicatas a menos que a opção DISTINCT seja especificada em uma consulta.
- Técnica de Ordenação: Ordene em <sid, bid> e remova duplicatas. (Pode-se otimizar descartando dados não necessários durante a ordenação.)
- Técnica de Hashing : Fazemos um hash em <sid, bid> criando-se assim partições. Carregam-se as partições para a memória uma por vez, fazendo-se uma estrutura de hash na memória, e eliminando-se duplicatas.
- Se há um índice com ambos R.sid e R.bid na chave de busca, pode ser mais barato ordenar os dados de entrada!

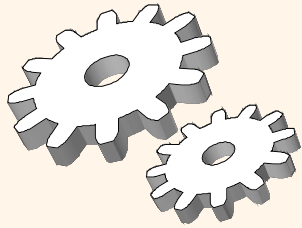
# *Junção: Índices em Laços Aninhados*



foreach tuple r in R do
    foreach tuple s in S where $r_i$ == $s_j$ do
        add <r, s> to result

- Se há um índice na coluna de união de uma das relações (S por ex.), podemos explorar o índice no laço interno.
  - Custo: M + ( (M*$p_R$) * custo de se achar tuplas em S que casam)
  - M=núm. de páginas de R; $p_R$= núm. de tuplas de R por página
- Para cada tupla de R, custo de explorar o índice de S é algo como 1.2 para índices em hash e 2-4 para árvores B+. Custo de então encontrar as tuplas de S depende de clustering.
  - Índices em cluster: 1 I/O (tipicamente), sem cluster: até 1 I/O por tupla de S que casa.

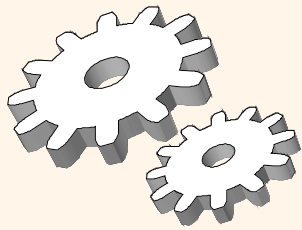
# *Exemplos de Índice em laços aninhados*



```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid
```

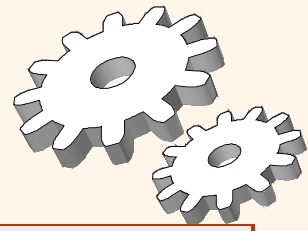
- Hash em *sid* de Sailors (como laço interno):
  - Varredura em Reserves: 1000 I/Os (páginas), 100*1000 tuplas.
  - Para cada tupla de Reserves: 1.2 I/Os para acessar os dados no índice, mais 1 I/O para achar a tupla (exatamente uma) que casa em Sailors. Total: 221,000 E/S. => 1000 + (100000 * (1.2 + 1))
- Hash em *sid* de Reserves (como laço interno):
  - Varredura em Sailors: 500 I/Os (páginas), 80*500 tuplas.
  - Para cada tupla de Sailors: 1.2 I/O para achar o índice da página com os dados, mais custo de recuperar tuplas de Reserves que casam. Assumindo distribuição uniforme , 2.5 reservas por navegador (100,000 / 40,000). Custo será 1 ou 2.5 I/Os dependendo se o índice é em *cluster*. Total: (500 + 40,000*1.2) + (40,000 *(1 ou 2.5)) = 88,500 ou 148,500.

# *Junção:Sort-Merge* ($R \bowtie_{i=j} S$)



- Ordene R e S na coluna de junção, depois percorra estas relações para junta-las (na col. de junção), e devolva as tuplas resultantes.
  - Avance em R até que R-col >= S-col corrente. Avance em S até S-col >= corrente R-col. Faça isso até termos R-col = S-col.
  - Neste ponto, todas tuplas de R com o mesmo valor em Ri (grupo corrente de *R*) e todas tuplas de S com mesmo valor em Sj (grupo corrente de *S) casam*; devolva todos os pares <r, s> de tais tuplas.
  - Continue percorrendo R e S como acima.
- R é percorrida uma vez; cada grupo de S é percorrido uma vez para cada tupla que casa de R. (Múltiplas varreduras de um grupo de S provavelmente encontram as páginas necessárias no buffer.)

# *Exemplo de Junção Sort-Merge*

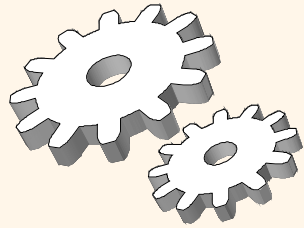


| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

| sid | bid | day | rname |
|---|---|---|---|
| 28 | 103 | 12/4/96 | guppy |
| 28 | 103 | 11/3/96 | yuppy |
| 31 | 101 | 10/10/96 | dustin |
| 31 | 102 | 10/12/96 | lubber |
| 31 | 101 | 10/11/96 | lubber |
| 58 | 103 | 11/12/96 | dustin |

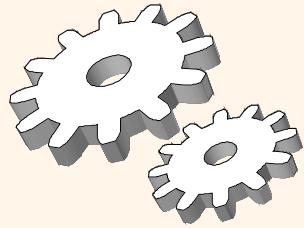
- Custo: M log M + N log N + (M+N)
  - O custo de percorrer, M+N, poderia ser M*N (ruim!)
- Com 35, 100 ou 300 páginas de buffer, Reserves e Sailors podem ser ordenados em 2 passos; custo total da junção: 7500 = (2*2*1000 + 2*2*500 + 1000 + 500).

# *Otimizador do Sistema R*



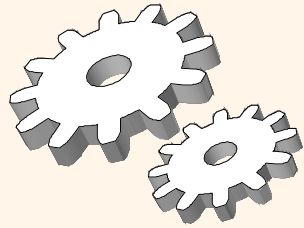
- Impacto:
  - Mais utilizado atualmente; trabalha bem para <10 junções.
- Estimativa de custo: Aproxima pelo melhor.
  - Estatísticas, mantidas em catálogos, são utilizados para estimar custos de operações e tamanhos de resultados.
  - Considera combinação de custos de CPU e I/O.
- Conjunto de Planos: Muito grande, deve ser podado.
  - Somente o espaço de planos com profundidade à esquerda *"left-deep"* são considerados.
    - Planos com profundidade a esquerda permitem que seja feito um *pipeline* com o próximo operador sem armazenamento em uma relação intermediária.
  - Produtos cartesianos são evitados.

# *Estimativa de Custos*



- Para cada plano considerado deve-se estimar o custo:
  - Deve-se estimar o custo de cada operação na árvore do plano.
    - Depende da cardinalidade da entrada.
    - Já foi discutido como estimar o custo de operações (percorrer de forma seqüencial e indexado, junções, etc.)
  - Deve-se também estimar o tamanho do resultado para cada operação na árvore!
    - Usam-se informações sobre as relações de entrada.
    - Para seleções e junções, assumimos a independência dos predicados.

# *Estimativa de Tamanho e Fatores de Redução*

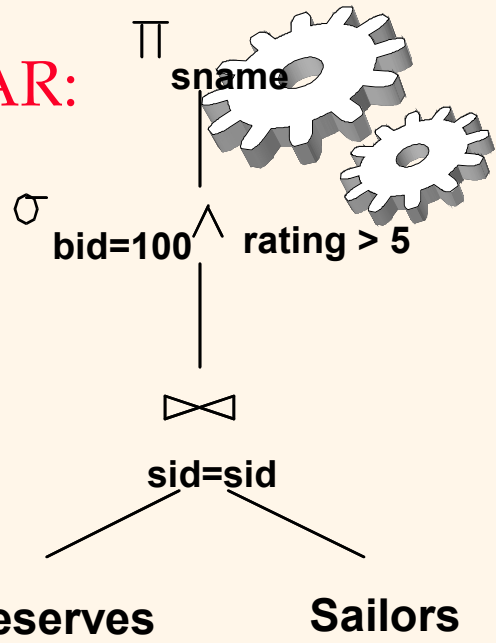


```
SELECT  attribute list
FROM  relation list
WHERE  term1 AND ... AND termk
```

- Considere a consulta:
- O número máximo de tuplas resultante é o produto das cardinalidades das relações na cláusula FROM.
- *Fator de Redução (FR)* associado com cada *termo* reflete o impacto do *termo* na redução do tamanho do resultado. *Cardinalidade Resultado* = Max # tuplas * produto de todos FR's.
  - Assumimos implicitamente que os *termos* são independentes!
  - Termo *col=value* tem FR *1/NKeys(I)*, dado que temos índice I em *col*
  - Termo *col1=col2* tem FR *1/MAX(NKeys(I1), NKeys(I2))*
  - Termo *col>value* tem FR *(High(I)-value)/(High(I)-Low(I))*

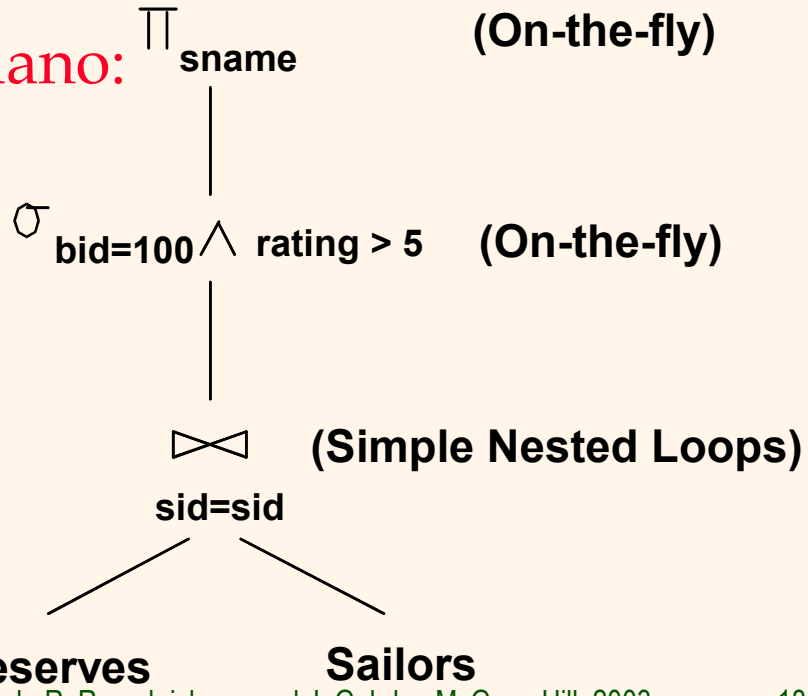
# *Exemplo*

Árvore AR:

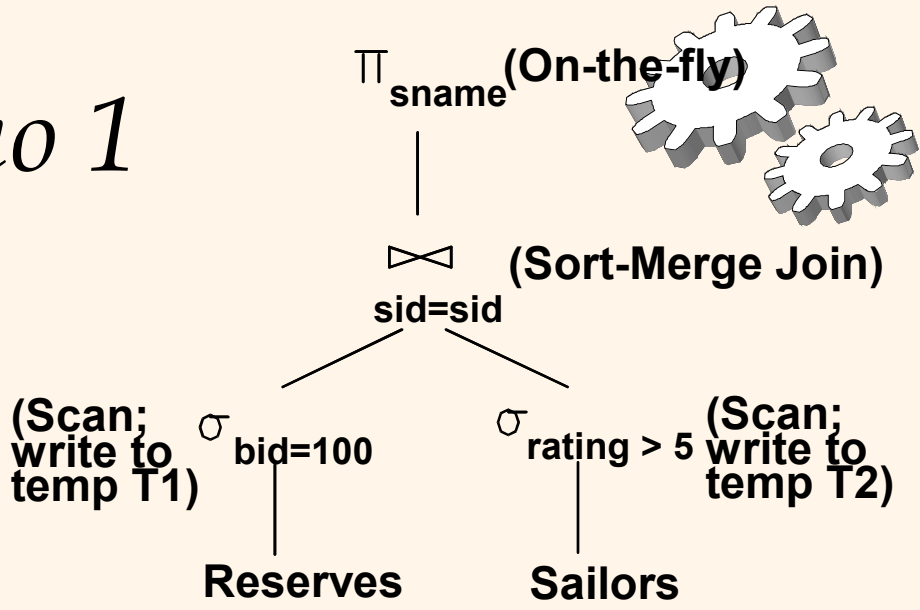



```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
    R.bid=100 AND S.rating>5
```

- Custo: 500+500*1000 E/S
- De forma alguma o pior plano. Plano:
- Perde otimizações: seleções podem ser feitas antes, não usa possíveis índices, etc.
- *Objetivo da otimização:* Achar planos mais eficientes que computam a mesma resposta.

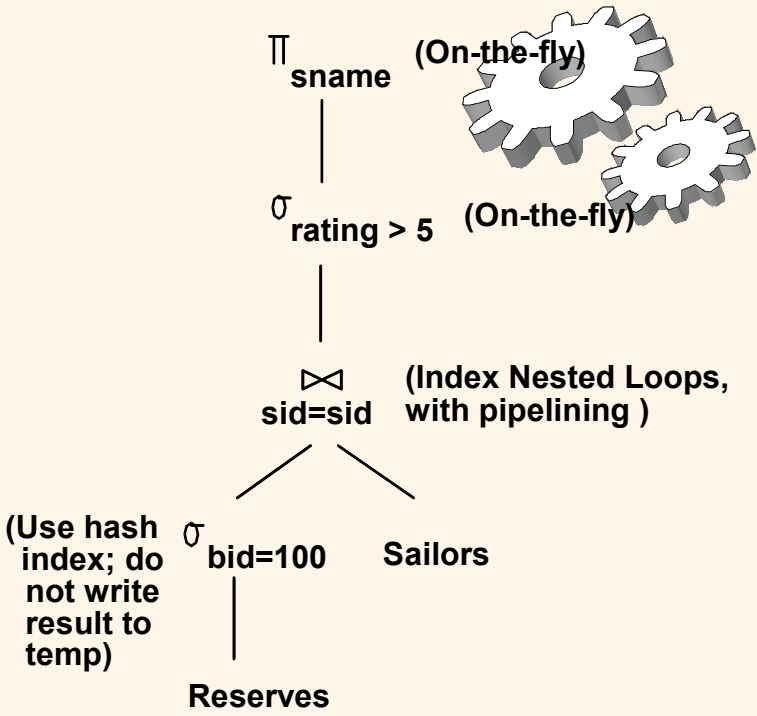


# *Alternativa de Plano 1 (Sem Índices)*

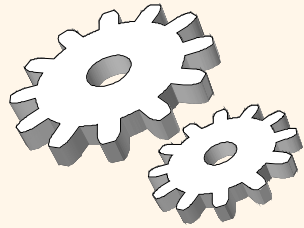



- ***Principal diferença:*** *adiantar seleções.*
- Com 5 buffers, custo do plano:
  - Percorrer Reserves (1000) + escrever temp T1 (10 págs, se temos 100 barcos, distribuição uniforme). = 1010 I/Os
  - Percorrer Sailors (500) + escrever temp T2 (250 págs, se temos 10 ratings). = 750
  - Ordenar T1 (2*2*10), Ordenar T2 (2*4*250), juntar (10+250) = 2300 I/Os
  - Total: 4060 I/Os de páginas.
- Se usarmos junção BNL (Block Nested Loop), custo da junção = 10+4*250. Custo total = 2770.
- Se adiantarmos projeções, T1 terá somente *sid*, T2 terá somente *sid* e *sname*:
  - T1 cabe em 3 págs, custo do BNL cai para menos de 250 págs, total < 2000.

# *Alternativa de Plano 2 Com Índices*




- Com índice em cluster em *bid* de Reserves, temos 100,000/100 = 1000 tuplas em 1000/100 = 10 págs.
- INL com ***pipelining*** (laço externo não materializado).
  - –Realizar projeção no dados não necessários do laço externo não ajuda.
- Coluna de junção *sid* é uma chave em Sailors.
  - –No máximo uma tupla que casa; sem índice agrupado em *sid* OK.
- Decisão de não adiantar *rating*>5 antes da junção está baseado na disponibilidade do índice em *sid* de Sailors.
- Custo: Seleção das tuplas em Reserves (10 I/Os); para cada, deve pegar tuplas que casam em Sailors (1000*1.2); total 1210 E/S.



# *Sumário*

- Há diversos algoritmos alternativos para avaliar cada operador relacional.
- Uma consulta é avaliada convertendo-a para uma árvore de operadores e avaliando cada um destes na árvore.
- Deve-se entender a otimização de consultas para compreensão do impacto de um design (relações, índices) na performance de uma carga de trabalho (conjunto de consultas).
- Dois passos para otimizar uma consulta:
  - Considerar um conjunto de planos alternativos.
    - Deve-se limitar o espaço considerado; tipicamente apenas planos com profundidade a esquerda.
  - Estimar o custo de cada plano considerado.
    - Estima-se o tamanho do resultado e o custo para cada nó do plano.
    - *Assuntos chave*: estatísticas, índices, implementação de operadores.