Otimização de Consultas

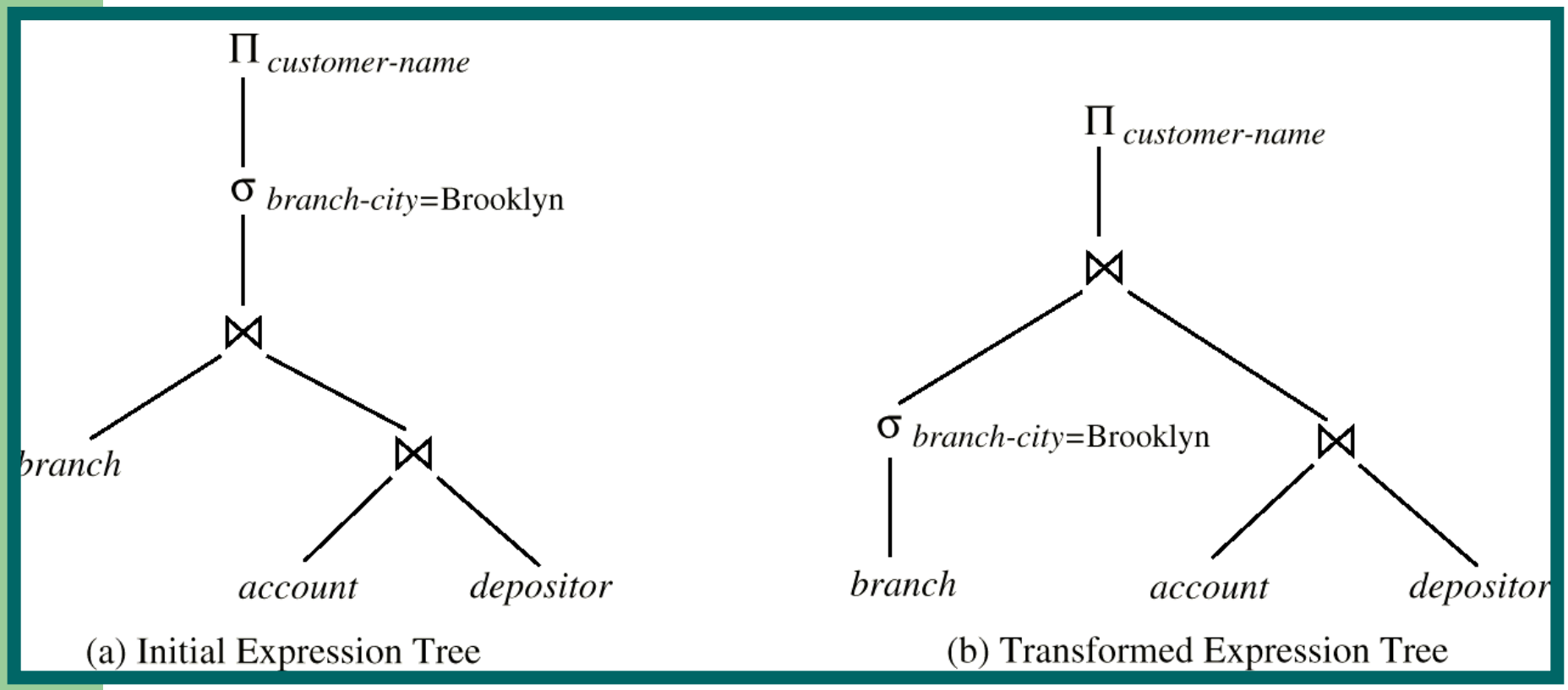
Laboratório de Bancos de Dados

Introdução

- Formas alternativas de avaliar uma dada consulta
  - Expressões equivalentes
  - Algoritmos diferentes para cada operação (Tema visto)
- Diferença de custo entre uma boa e uma má maneira de avaliar uma consulta pode ser grande
  - Exemplo: execução de *r* X *s* seguido por uma seleção *r.A* = *s.B* é bem mais lento que executar uma junção sobre a mesma condição
- Precisa estimar o custo das operações
  - Depende criticamente da informação estatística sobre as relações que o banco de dados deve manter
    - E.x. número de tuplas, número de valores diferentes para atributos de junção, etc.
  - Precisa estimar estatísticas para os resultados intermediários do custo de computação de expressões complexas

Introducão (Cont.)

As relações geradas por duas expressões equivalentes têm o mesmo conjunto de atributos e contem o mesmo conjunto de tuplas, entretanto os algoritmos para calcular as operações podem ser diferentes



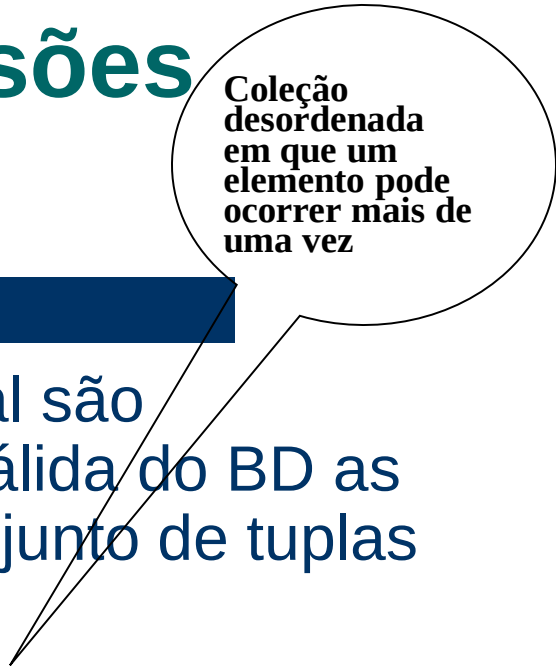
Introdução (Cont.)

- Geração de planos de avaliação de consultas para uma expressão considera vários passos:
  1. Geração de expressões logicamente equivalentes
     - Usar **regras de equivalência** para transformar uma expressão em equivalentes.
  2. Anotação de expressões resultantes para obter planos alternativos de avaliação de consultas
  3. Escolha o plano mais barato baseado no **custo estimado**
- A Estimação do custo do plano é baseada em:
  - Informação estatística sobre as relações
  - Estimação estatística dos resultados intermediários
  - Calculo da formula de custo dos algoritmos, usando estatísticas
- Todo o processo é chamado **otimização baseada no custo.**

Agenda

- Regras de equivalência
- Escolhas de planos de avaliação
- Algoritmo de otimização baseado no custo
- Estimação das estatísticas dos resultados das expressões
- Otimização de sub-consultas aninhadas

Transformação de Expressões Relacionais



- Duas expressões da álgebra relacional são equivalentes se para cada instância válida do BD as duas expressões geram o mesmo conjunto de tuplas
  - PS: a ordem das tuplas é irrelevante
- Em SQL, entradas e saídas são multiconjuntos de tuplas
  - Duas expressões na versão de multiconjunto da álgebra relacional são equivalentes se, em cada BD válido, as duas expressões geram o mesmo multiconjunto de tuplas
- Uma regra de equivalência diz que se expressões de duas formas são equivalentes
  - Podemos trocar expressão da primeira forma pela segunda, ou vice versa

Regras de Equivalência

1. As operações de uma seleção conjuntiva podem ser divididas numa seqüência de seleções individuais.

2. As operações de seleção são comutativas.

3. Somente a última de uma seqüência de operações de projeção é necessária, as outras podem ser omitidas.

1. As seleções podem ser combinadas com os produtos cartesianos e theta junções.
   - α. $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
   - β. $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

Regras de Equivalência (Cont.)

5. As operações de junção theta (e junções naturais) são comutativas.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

6. (a) As operações de junção natural são associativas:

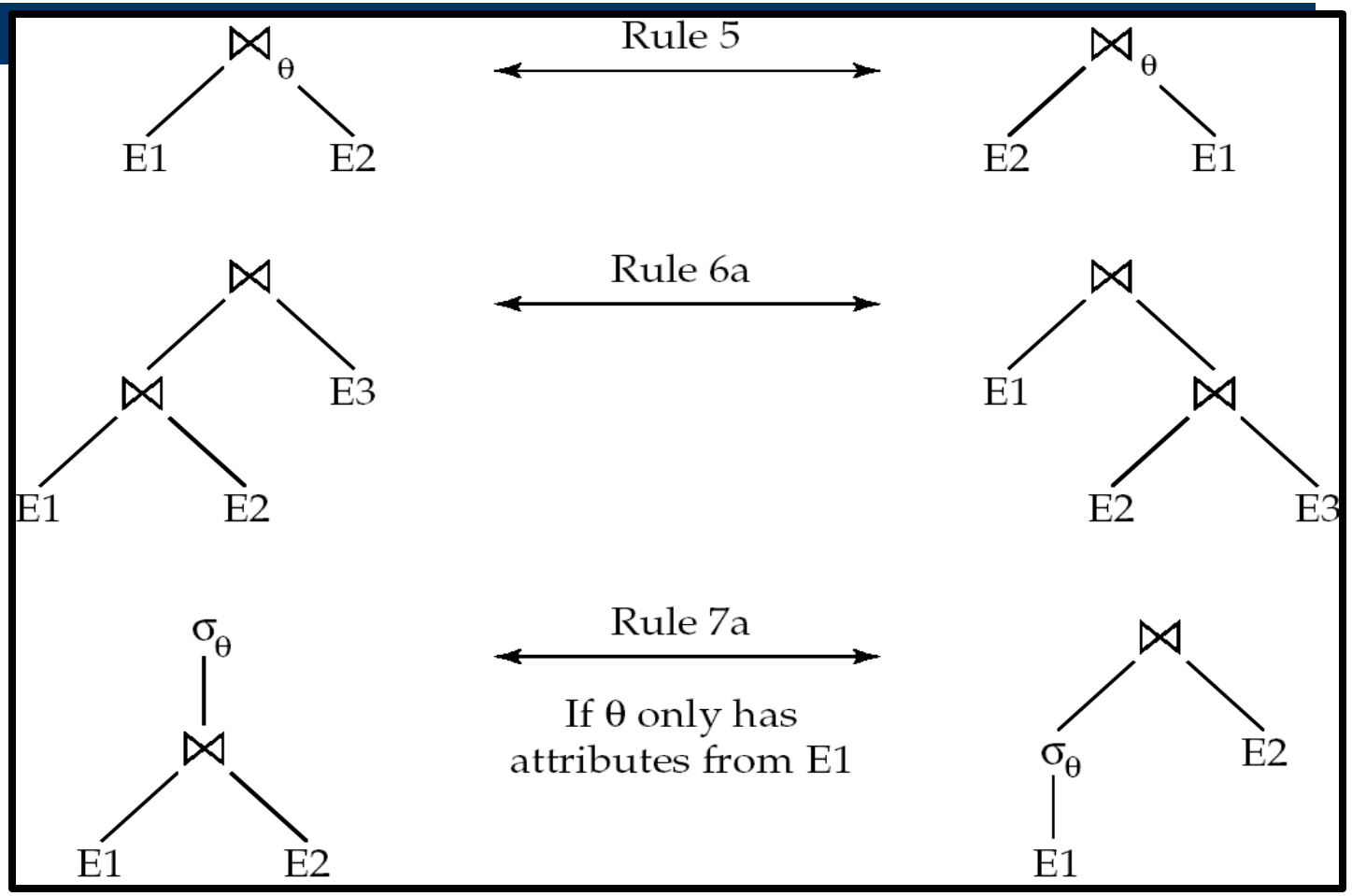
$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

(b) As junções theta são associativas da seguinte maneira:

$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

onde θ_2 contem atributos somente de E_2 e E_3.

Representação Gráfica das Equivalências



Regras de Equivalência (Cont.)

7. A operação de seleção se distribui pela operação de junção theta sob as seguintes duas condições:
   (a) Quando todos os atributos de θ_0 implicam unicamente os atributos de uma das expressões (E_1) que estão sendo juntadas.

$$\sigma_{\theta 0}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 0}(E_1)) \bowtie_{\theta} E_2$$

   (b) Quando θ_1 implica unicamente atributos de E_1 e θ_2 implica unicamente atributos de E_2.

$$\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_{\theta} (\sigma_{\theta 2}(E_2))$$

Regras de Equivalência (Cont.)

8. A operação de projeção se distribui pela operação de junção theta como segue:
   - Sejam L_1 e L_2 conjuntos de atributos de E_1 e E_2, respectivamente.

   (a) se θ envolve unicamente atributos de $L_1 \cup L_2$:

   (b) Considere uma junção $E_{1} \bowtie_{\theta} E_{2}$.
   - Seja L_3 os atributos de E_1 que são envolvidos na condição de junção θ, mas não estão em $L_1 \cup L_2$, e
   - Seja L_4 os atributos de E_2 que são envolvidos na condição de junção θ, mas não estão em $L_1 \cup L_2$.

Regras de Equivalência (Cont.)

9. O conjunto de operações união e interseção são comutativas

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

   - (diferença de conjuntos não é comutativa

10. União e interseção são associativas

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. A operação seleção distribui sobre $\cup$, $\cap$ e $-$.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

e similarmente para $\cup$ e $\cap$ no lugar de $-$
Também: $\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$ e similarmente para $\cap$ no lugar de $-$, mas não para $\cup$

12. A operação de projeção distribui sobre união

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$

Exemplo de Transformação

- Consulta: Achar os nomes de todos os clientes que tem uma conta em alguma agência localizada em Brooklyn.

$$\Pi_{\textit{nome-cliente}} (\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}} (\textit{agência} \bowtie (\textit{conta} \bowtie \textit{dono-conta})))$$

- Transformação usando regra 7a.

$$\Pi_{\textit{nome-cliente}} ((\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}} (\textit{agência})) \bowtie (\textit{conta} \bowtie \textit{dono-conta}))$$

- Executando a seleção o mais cedo possível reduz o tamanho da relação a ser reunida (joined).

Exemplo com várias Transformações

- Consulta: Achar os nomes de todos os clientes com uma conta numa agência de Brooklyn cujo saldo é maior de $1000.

  $\Pi_{\textit{nome-cliente}} (\sigma_{\textit{cidade-agência} = \text{“Brooklyn”} \wedge \textit{saldo} > 1000} (\textit{agência} \bowtie (\textit{conta} \bowtie \textit{dono-conta})))$

- Transformação usando junção associativamente (Regra 6a):

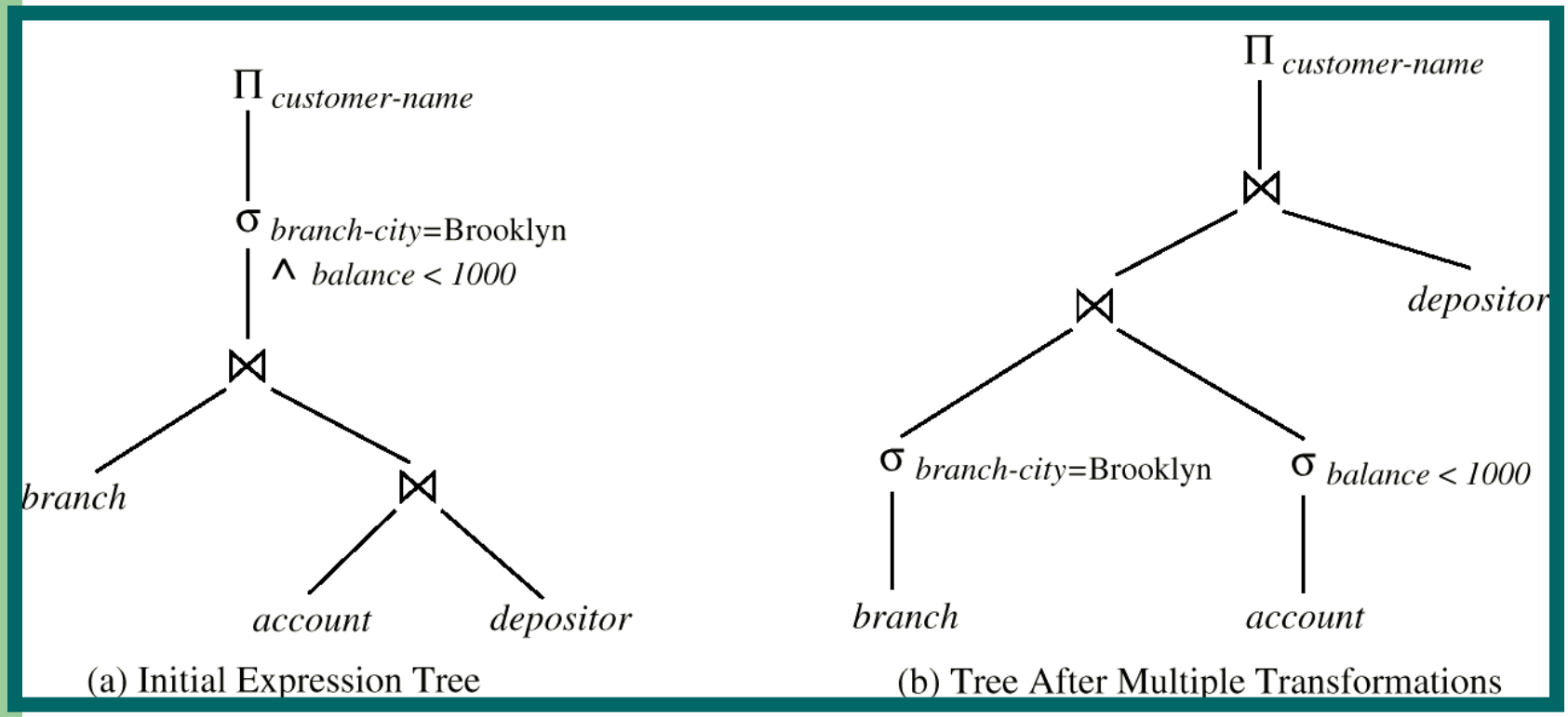
  $\Pi_{\textit{nome-cliente}} ((\sigma_{\textit{cidade-agência} = \text{“Brooklyn”} \wedge \textit{saldo} > 1000} (\textit{agência} \bowtie (\textit{conta})) \bowtie \textit{dono-conta})$

- Segunda forma fornece uma oportunidade para aplicar a regra “desenvolver as seleções cedo”, resultando na subexpressão

  $\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}} (\textit{agência}) \bowtie \sigma_{\textit{saldo} > 1000} (\textit{conta})$

- Assim uma seqüência de transformações pode ser útil

Múltiplas Transformações (Cont.)



Exemplo da Operação de Projeção

$\Pi_{\textit{nome-cliente}}((\sigma_{\textit{cidade-agência}} = \text{“Brooklyn”} (\textit{agência}) \bowtie \textit{conta}) \bowtie \textit{dono-conta})$

- Quando nós calculamos

  $(\sigma_{\textit{cidade-agência}} = \text{“Brooklyn”} (\textit{agência}) \bowtie \textit{conta})$

  nós obtemos uma relação cujo esquema é:
  (*nome-agência,cidade-agência, ativo, número-conta, saldo)*
- Forçar projeções usando a regra de equivalência 8b; elimina atributos não necessários de resultados intermediários:

  $\Pi_{\textit{nome-cliente}}(($
  $\Pi_{\textit{número-conta}}((\sigma_{\textit{cidade-agência}} = \text{“Brooklyn”} (\textit{agência}) \bowtie \textit{conta}))$
  $\bowtie \textit{dono-conta})$
- *Realizar a projeção o mais cedo possível reduz o tamanho da relação a ser reunida.*

Exemplo de Ordenação de Junções

- Para todas as relações r_1, r_2, e r_3,

  $(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$

- Se $r_2 \bowtie r_3$ é muito grande e $r_1 \bowtie r_2$ é pequena, nós selecionamos

  $(r_1 \bowtie r_2) \bowtie r_3$

  de forma que nós calculemos e armazenemos a menor relação temporária.

Exemplo de Ordenação de Junções (Cont.)

- Considere a expressão

  $\Pi_{\textit{nome-cliente}}((\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}}(\textit{agência})) \bowtie (\text{conta} \bowtie \textit{dono-conta}))$

- Poderíamos calcular conta $\bowtie$ *dono-conta* primeiro, e juntar o resultado com

  $\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}}(\textit{agência})$

  mas conta $\bowtie$ *dono-conta* é provavelmente uma relação grande.

- Já que o mais provável é que somente uma pequena fração dos clientes do banco têm contas em agências localizadas em Brooklyn, o melhor é calcular

  $\sigma_{\textit{cidade-agência} = \text{“Brooklyn”}}(\textit{agência}) \bowtie \text{conta}$

  primeiro.

Enumeração de Expressões Equivalentes

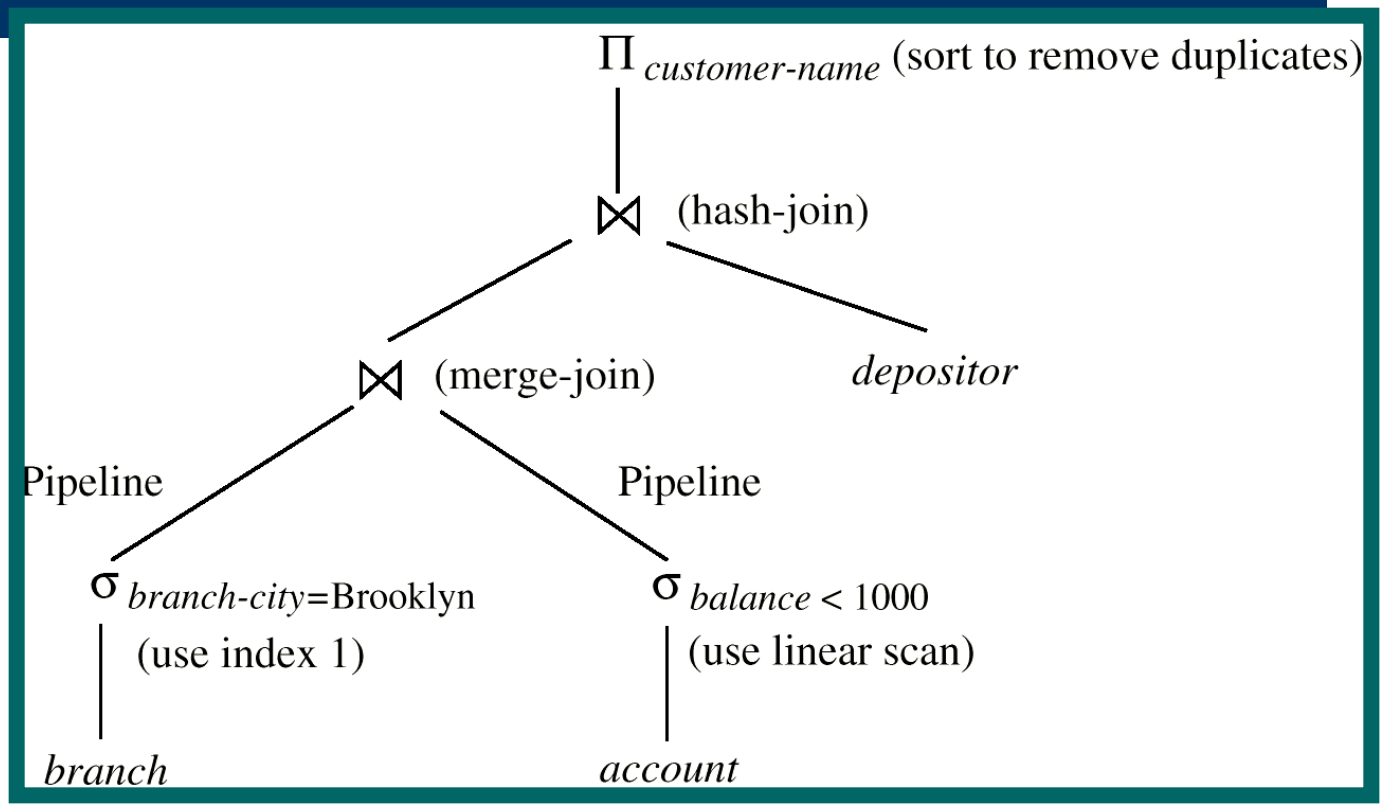
- Otimizadores de consulta usam regras de equivalência para gerar sistematicamente expressões equivalentes à expressão dada
- Pode gerar todas as expressões equivalentes como segue:
  - Repita
    - Aplicar todas as regras de equivalência possíveis sobre toda expressão achada até o momento
    - Adicionar expressões geradas ao conjunto de expressões equivalentes
  - Até que nenhuma nova expressão equivalente possa ser gerada
- A abordagem acima é muito custosa em espaço e tempo
  - Duas abordagens
    - Geração do plano otimizado baseado nas regras de transformação
    - Abordagem especial para consultas com somente seleções, projeções e junções

Estimação do custo

- O custo de cada operação descrito na aula de Processamento de consultas depende:
  - Das estatísticas das relações de entrada. Ex: número de tuplas, tamanho das tuplas
- Entradas podem ser o resultado de sub-expressões
  - Estimar as estatísticas do resultado de expressões
  - Estatísticas adicionais. Ex: número de valores diferentes para um atributo

Plano de Avaliação

- Um plano de avaliação define exatamente que algoritmo é usado para cada operação, e como a execução das operações é coordenada.



Escolha dos Planos de Avaliação

- Deve considerar a interação das técnicas de avaliação quando escolhemos planos de avaliação: escolher o algoritmo mais barato para cada operação independentemente pode não conduzir ao melhor plano total. Ex:
  - A junção merge pode ser mais custosa que a junção hash, mas pode fornecer uma saída ordenada que reduz o custo das operações seguintes.
  - A junção de laços aninhados pode fornecer facilidades para o pipelining
- Otimizadores de consultas práticos incorporam elementos das seguintes abordagens:
  1. Busca todos os planos e escolhe o melhor usando o custo.
  2. Usa heurísticas para escolher um plano.

Otimização Baseada no Custo

- Considere achar a melhor ordenação de junções para $r_1 \bowtie r_2 \bowtie \ldots r_n$.
- Existe $(2(n-1))!/(n-1)!$ ordenações de junção diferentes para a expressão acima. Com $n = 7$, o número é 665280, com $n = 10$, o número é maior que 176 bilhões!
- Não precisamos gerar todas as ordenações. Usando programação dinâmica, o custo mínimo da ordenação das junções para qualquer subconjunto de $\{r_1, r_2, \ldots r_n\}$ é calculado uma única vez e armazenado para seu uso futuro.

Programação Dinâmica na Otimização

- Para achar a melhor árvore de junções para um conjunto de *n* relações:
  - Para achar o melhor plano para um conjunto *S* of *n* relações, considere todos os planos possíveis da forma: $S_1 \bowtie (S - S_1)$ onde S_1 é qualquer subconjunto não vazio de *S*.
  - Recursivamente calculamos os custos para juntar subconjuntos de S para achar o custo de cada plano. Escolha o mais barato das $2^n - 1$ alternativas.
  - Quando o plano para qualquer subconjunto é calculado, armazene este e reuse-o quando ele for requerido de novo, no lugar de recalcular ele novamente
    - Programação Dinâmica

Algoritmo de Otimização da Ordenação de Junções

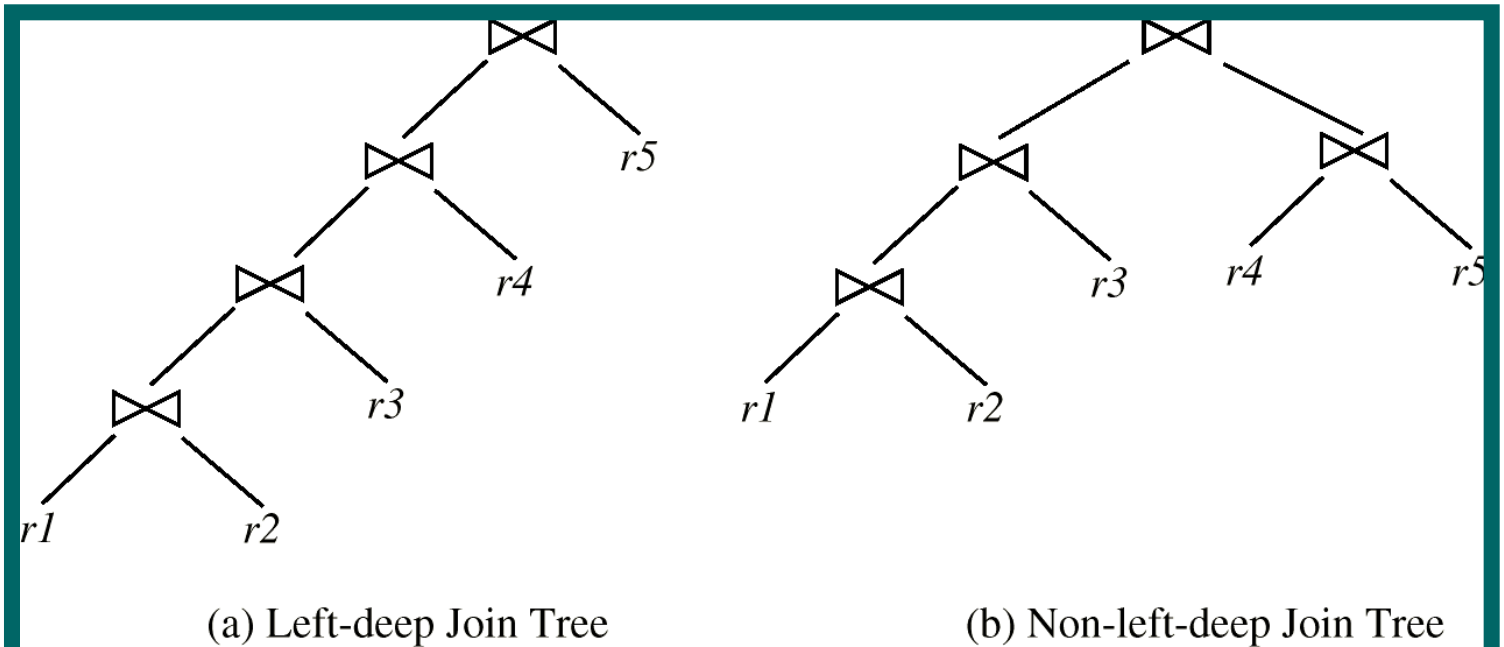
procedure achamelhorplano(*S*)
    if (*melhorplano*[*S*].*custo* ≠ ∞)
        return *melhorplano*[*S*]
    // else *melhorplano*[*S*] não tem sido calculado antes, calcule ele agora
    if (*S* contem somente 1 relação)
        faça *melhorplano*[*S*].*plano* e *melhorplano*[*S*].*custo* baseado na melhor forma de acessar *S* /* *Usiando seleções sobre S e indices sobre S* */
    else for each sub-conjunto não-vazio *S*1 de *S* tal que *S*1 ≠ *S*
        P1= achamelhorplano(*S*1)
        P2= achamelhorplano(*S* - *S*1)
        A = melhor algoritmo para juntar "joining" resultados de *P*1 e *P*2
        custo = *P*1.*custo* + *P*2.*custo* + custo de *A*
        if *custo* < *melhorplano*[*S*]. custo
            melhorplano[*S*]. custo = custo
            melhorplano[*S*]. *plano n* = "execute *P*1. *plano*; execute *P*2. *plano*; e faça join dos resultados de *P*1 e *P*2 usando *A*"
    return *melhorplano*[*S*]

Custo de Otimização

- Com programação dinâmica a complexidade em tempo da otimização com árvores é $O(3^n)$.
  - Com n = 10, este número é 59000 no lugar de 176 bilhões
- Complexidade em espaço é $O(2^n)$
- Para achar a melhor árvore de junções em profundidade pela esquerda para um conjunto de n relações:
  - Considere n alternativas com uma relação como entrada do lado direito e as outras relações como entradas do lado esquerdo
  - Modificar o algoritmo de otimização:
    - Substitua "**for each** sub-conjunto não-vazio $S1$ de S tal que $S1 \neq S$"
    - Por: **for each** relação r em S
seja S1 = S – r **.**
- Se somente árvores em profundidade pela esquerda são considerados, a complexidade em tempo para achar a melhor junção é da ordem $O(n\ 2^n)$
  - complexidade em espaço permanece em $O(2^n)$
- Otimização baseada no custo é custosa, mas útil para consultas sobre grandes conjuntos de dados (consultas típicas têm n pequeno, geralmente < 10)

Árvores de reunião em profundidade pela esquerda.

- Nos Árvores de reunião em profundidade pela esquerda**,** a entrada do lado direito para cada junção é uma relação, não o resultado de uma junção intermediária.



Ordenações Interessantes na Otimização Baseada no Custo

- Considere a expressão $(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$
- Uma ordenação **interessante é** uma ordenação particular de tuplas que pode ser útil para uma operação posterior.
  - A geração do resultado de $r_1 \bowtie r_2 \bowtie r_3$ classificado pelos atributos comuns com r_4 ou r_5 pode ser útil, mas a geração classificada pelos atributos comuns apenas a r_1 e r_2 não é.
  - Usar a junção merge para calcular $r_1 \bowtie r_2 \bowtie r_3$ pode ser mais custoso, mas pode oferecer uma saída classificada numa ordem de classificação interessante.
- Não é suficiente achar a melhor ordem de junção para cada subconjunto do conjunto de *n* relações dadas; deve achar a melhor ordem de junção para cada subconjunto, para cada ordem de classificação interessante
  - Extensão simples dos algoritmos de programação dinâmica
  - Usualmente, o número de ordens de classificação interessantes é muito pequeno e não afeta significantemente a complexidade tempo/espaço

Otimização Heurística

- Otimização baseada no custo é custosa, mesmo com programação dinâmica.
- Os sistemas podem usar *heurísticas* para reduzir o número de escolhas que devem ser feitas num método baseado em custo. Ideal combinação da programação dinâmica com heuríticas
- A otimização heurística transforma a árvore de consulta usando um conjunto de regras que tipicamente (mas não em todos os casos) melhoram o desempenho da execução:
  - Executar as seleções o mais cedo (reduz o número de tuplas)
  - Executar as projeções o mais cedo (reduz o número de atributos)
  - Executar as operações de seleção e junção mais restritivas antes de outras operações similares.
  - Alguns sistemas usam somente heurísticas, outros combinam heurísticas com otimização parcial baseada em custo.

Passos do algoritmo típico de Otimização Heurística

1. Decompor as seleções conjuntivas numa seqüência de operações de seleção simples (regra de equiv. 1.).
2. Mover as operações de seleção para a parte inferior da árvore de consultas para poder executa-as o antes possível (regras de equiv. 2, 7a, 7b, 10).
3. Executar primeiro essas operações de seleção e junção que produzirão as relações de menor tamanho (regra de equiv. 6).
4. Trocar as operações de produto cartesiano seguidas por uma condição de seleção por operações de junção (regra de equiv. 4a).
5. Decompor e mover para baixo da árvore o que seja possível as listas de atributos de projeção, criando novas projeções onde seja necessário (regras de equiv. 3, 8a, 8b, 11).
6. Identificar as sub-árvores cujas operações podem ser pipelined, e executa-as usando pipelining).

Estrutura dos Otimizadores de Consultas

- O otimizador do sistema R/Starburst considera somente ordens de junção em profundidade pela esquerda. Isto reduz a complexidade da otimização e gera planos convenientes para a avaliação pipelined. O Sistema R/Starburst também usa heurísticas para mover as seleções e projeções para baixo da árvore de consulta.
- Para buscas usando índices secundários, alguns otimizadores consideram a probabilidade que a página que contem a tupla esteja no buffer.
- Complexidade de SQL complica a otimização de consultas
  - E.g. subconsultas aninhadas

Estrutura dos Otimizadores de Consultas (Cont.)

- Alguns otimizadores de consulta integram seleção heurística e a geração de planos de acesso alternativos.
  - O Sistema R e Starburst usam um procedimento hierárquico baseado no conceito de bloco aninhado de SQL: heurística de reescrita seguida pela otimização baseada no custo da ordem de junção.
- Mesmo com o uso de heurísticas, a otimização baseada no custo da consulta impõe um overhead considerável.
- Este custo é mais do que compensado pela economia no tempo de execução da consulta, especificamente pela redução do número de acessos de disco lentos.

Informação estatística para estimação do custo

- n_r: número de tuplas na relação *r.*
- b_r: número de blocos contendo as tuplas de *r.*
- s_r: tamanho de uma tupla de *r.*
- f_r: fator de bloqueio de *r* → o número de tuplas de *r* que cabem num bloco.
- *V(A, r):* número de valores diferentes que aparecem em *r* para o atributo *A; o mesmo que o tamanho de* $\prod_A(r)$.
- Se as tuplas de *r* são armazenadas fisicamente juntas num arquivo, então:

$$b_r = \left\lceil \frac{n_r}{f_r} \right\rceil$$

Informação do Catalogo sobre os Índices

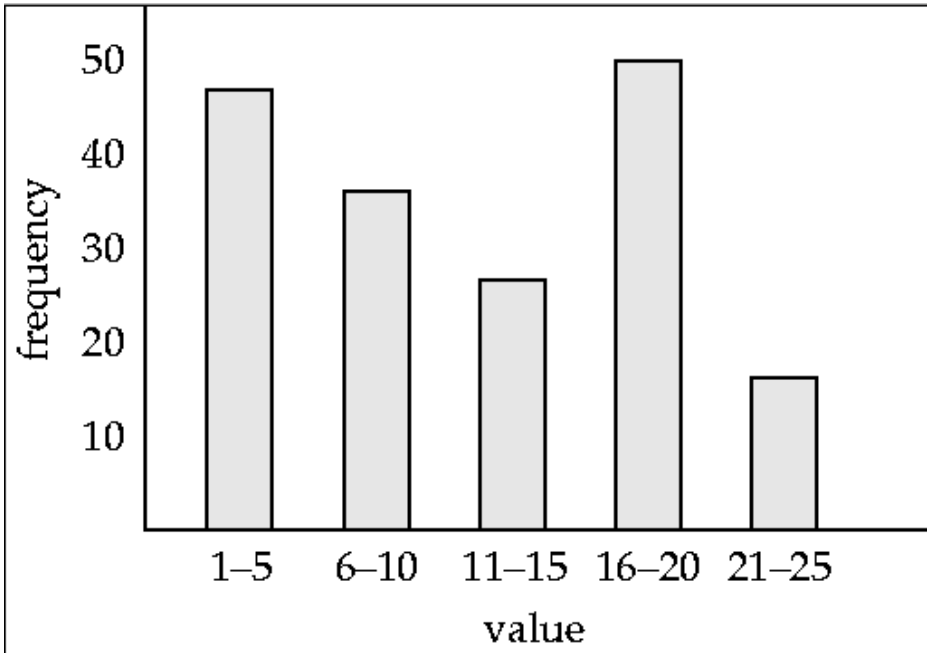
- f_i: número médio de nodos internos do índice *i*, para índices estruturados em árvores tais como B+-trees.
- HT_i: número de níveis no índice *i* — a altura de *i.*
  - Para um índice em árvore balanceado (tal como B+-tree) sobre o atributo *A* da relação *r,* $HT_i = \lceil \log_{fi}(V(A,r)) \rceil$.
  - Para um índice de hash, HT_i *é* 1.
  - LB_i: número de blocos do mais baixo nível do índice *i* — o número def blocos no nível das folhas do índice.

Informações do catálogo

- Manutenção das estatísticas: precisão vs. Overhead.
- Atualização durante períodos de pouca carga do sistema.
- Uma informação usada pelos SGBDs é a distribuição de valores para cada atributo como um histograma. Caso contrário se assume distribuição uniforme de valores.

Histogramas

- Histograma sobre o atributo *idade* da relação *pessoa*



- Histogramas de largura igual (veja figura)
- Histogramas de profundidade igual (intervalos com o mesmo número de valores)

Medidas do Custo de uma Consulta

- Lembre que
  - Tipicamente os acessos a disco é o custo predominante, e é também relativamente o mais fácil de estimar.
  - O *número de transferências de blocos de disco* é usado como uma medida do custo real da avaliação.
  - Se assume que todas as transferências de blocos têm o mesmo custo.
    - Otimizadores mais reais não supõe isso e diferenciam entre acessos seqüenciais e aleatórios
- Não incluímos o custo de escrita no disco.
- Nos referimos ao custo estimado do algoritmo *A* como E_A

Estimação do tamanho da Seleção

- **Seleção por igualdade** $\sigma_{A=V}(r)$
  - $n_r / V(A,r)$: número de registros que satisfarão a seleção
  - Condição de igualdade sobre um a atributo chave: $n_r / V(A,r) = 1$

Informação Estatística para Exemplos

- f_{conta}= 20 (20 tuplas de *conta* cabem em um bloco)
- *V(nome-agência, conta)* = 50 (50 agências)
- *V(saldo, conta)* = 500 (500 valores diferentes de *saldo*)
- π_{conta} = 10000 (*conta* tem 10,000 tuplas)
- Assuma que os seguintes índices existem para *conta:*
  - Um índice primário B^+-tree, para o atributo *nome-agência*
  - Um índice secundário, B^+-tree, para o atributo *saldo*

Seleções que envolvem Comparações

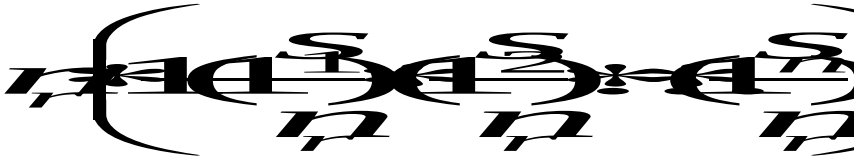
- Seleções da forma $\sigma_{A \leq V}(r)$ (caso de $\sigma_{A \geq V}(r)$ é simétrico)
- Seja c o número estimado de tuplas que satisfazem a condição.
  - Se min(A,r) e max(A,r) são disponíveis no catálogo
    - C = 0 se v < min(A,r) e se v>= max(A,r) ?
    - C = $n_r \times \frac{v - \min(A,r)}{\max(A,r) - \min(A,r)}$
  - Se histogramas são disponíveis, podem refinar o estimativo acima
  - Na ausência de informação estatística, a informação *de c* é assumido como $n_r / 2$.

Implementação de Seleções Complexas

- A **seletividade** de uma condição θ_i é a probabilidade que uma tupla na relação *r* satisfaça θ_i . Se s_i é o número de tuplas que satisfazem θ_i em *r*, a seletividade de θ_i é dado por s_i / n_r.
- **Conjunção:** $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \wedge \theta n}(r)$. O número estimado de tuplas no resultado é:

$$n_r * \frac{s_1 * s_2 * \ldots * s_n}{n_r^n}$$

- **Disjunção:** $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \vee \theta n}(r)$. O número estimado de tuplas:



- **Negação:** $\sigma_{\neg\theta}(r)$. O número estimado de tuplas: $n_r - size(\sigma_{\theta}(r))$

Operação de Junção : Exemplo de Execução

Exemplo de Execução:

dono-conta ⋈ *cliente*

Informação do Catálogo para exemplos de junção:

- $n_{cliente}$ = 10,000.
- $f_{cliente}$ = *25, o que implica* $b_{cliente}$ =10000/25 = 400.
- $n_{dono\text{-}conta}$ = 5000.
- $f_{dono\text{-}conta}$ = 50, o que implica $b_{dono\text{-}conta}$ = 5000/50 = 100.
- *V(nome-cliente, dono-conta)* = 2500, o que implica, na média, que cada cliente tem duas contas. Também assumimos que o *nome-cliente* em *dono-conta é uma chave estrangeira sobre cliente.*
  - *V(nome-cliente, cliente) = 10000 (chave principal!)*

Estimação do tamanho das junções

- O produto cartesiano r x s contem $n_r * n_s$ tuplas; cada tupla ocupa $s_r + s_s$ bytes.
- Se $R \cap S = \varnothing$, então $r \bowtie s$ é o mesmo que r x s.
- Se $R \cap S$ é uma chave para R, então uma tupla de s fará junção com no máximo uma tupla de r
  - Além disso, o número de tuplas em $r \bowtie s$ não é maior que o número de tuplas em s.
- *Se $R \cap S$ em* S é uma chave estrangeira em S referenciando R, *então* o número de tuplas em $r \bowtie s$ é exatamente o mesmo que o número de tuplas em s.
    - O caso de $R \cap S$ sendo uma chave estrangeira referenciando S é simétrico.
- No exemplo da consulta *dono-conta* $\bowtie$ *cliente, nome-cliente em dono-conta* é uma chave estrangeira de *cliente*
  - *portanto*, o resultado tem exatamente $n_{dono\text{-}conta}$ tuplas, a qual é 5000

Estimação do tamanho das junções (Cont.)

- Se $R \cap S = \{A\}$ não é uma chave de R ou S.
  Se nos assumimos que toda tupla t em R produz tuplas em $R \bowtie S$, o número de tuplas em $R \bowtie S$ é estimado para ser:

$$\frac{n_r * n_s}{V(A,s)}$$

  Se o contrário é verdadeiro, a estimação obtida será:

$$\frac{n_r * n_s}{V(A,r)}$$

  A menor destas duas estimações é provavelmente a mais correta.
- Pode melhorar os estimativos acima se histogramas são disponíveis

Estimação do tamanho das junções (Cont.)

- Calcular o tamanho estimado para *dono-conta* ⋈*cliente* sem o uso de informação sobre chaves estrangeiras:
  - *V(nome-cliente, dono-conta)* = 2500, e *V(nome-cliente, cliente)* = 10000
  - As duas estimações são 5000 * 10000/2500 = 20,000 e 5000 * 10000/10000 = 5000
  - Nós escolhemos a menor estimativa, a qual neste caso, é a mesma que nossa computação anterior que usa chaves estrangeiras.

Estimação do tamanho para Outras Operações

- Projeção: tamanho estimado de $\Pi_A(r)$ = $V(A,r)$
- Agregação : tamanho estimado de ${}_A\mathcal{g}_F(r)$ = $V(A,r)$
- Operações de conjuntos
  - Para uniões/interseções de seleções da mesma relação: reescreva e use o tamanho estimado para as seleções
    - E.x. $\sigma_{\theta 1}(r) \cup \sigma_{\theta 2}(r)$ pode ser reescrito como $\sigma_{\theta 1 \vee \theta 2}(r)$
  - Para operações sobre relações diferentes:
    - Tamanho estimado de $r \cup s$ = tamanho de r + tamanho de s.
    - Tamanho estimado de $r \cap s$ = tamanho mínimo de r e de s.
    - Tamanho estimado de $r - s$ = r.
    - Todas as três estimações pode ser muito inexatas, mas fornecem limites superiores para os tamanhos.

Estimação do tamanho (Cont.)

- Junção Externa (Outer join):
  - Tamanho estimado de $r \mathbin{⟕} s$ = *tamanho de* $r \bowtie s$ + *tamanho de r*
    - Caso de right outer join é simétrico
  - Tamanho estimado de $r \mathbin{⟗} s$ = *tamanho de r* $\bowtie s$ + tamanho de *r* + *tamanho de s*

Estimação do Número de Valores Diferentes

Seleções: $\sigma_\theta (r)$

- Se θ força A tomar um valor especificado: $V(A,\sigma_\theta (r)) = 1$.
  - ex., $A = 3$
- Se θ força A tomar um valor de um conjunto específico de valores:

  $V(A,\sigma_\theta (r))$ = número de valores especificados.
  - (ex., ($A = 1$ *ou* $A = 3$ *ou* $A = 4$)),
- Se a condição de seleção θ é da forma A *op* r estimado $V(A,\sigma_\theta (r)) = V(A.r) * s$
  - onde s *é* a seletividade da seleção.
- Em todos os outros casos: usa estimação aproximada de $\min(V(A,r), n_{\sigma\theta (r)})$
  - Estimação mais exata pode ser obtida usando teoria das probabilidades, mas esta trabalha bem geralmente

Estimação do Número de Valores Diferentes (Cont.)

Junções: $r \bowtie s$

- Se todos os atributos em A são de r
  estimativo $V(A, r \bowtie s) = \min(V(A,r), n_{r \bowtie s})$
- Se A contém atributos $A1$ de r e $A2$ de s, então estimado $V(A, r \bowtie s) =$

  $\min(V(A1,r)*V(A2 - A1,s), V(A1 - A2,r)*V(A2,s), n_{r \bowtie s})$
  - Estimativas mais precisas podem ser obtidas usando teoria da probabilidade, mas essas aproximações funcionam geralmente bem

Estimação de Valores Diferentes (Cont.)

- Estimação de valores diferentes é direta para projeções.
  - São iguais em $\Pi_{A\,(r)}$ que em *r*.
- O mesmo se cumpre para os atributos de agrupamento das agregações.
- Para valores agregados
  - Para min(*A*) e max(*A*), o número de valores diferentes pode ser estimado como min(V(*A,r*), *V*(*G,r*)) onde G descreve os atributos de agrupamento
  - Para outras agregações, assumimos que todos os valores são diferentes, e usamos *V*(*G,r*)

Otimização das subconsultas aninhadas**

- SQL conceitualmente trata sub-consultas aninhadas na cláusula where como funções que apanham parâmetros e retornam ou um único valor ou um conjunto de valores
  - Parâmetros são variáveis de consulta de nível externo que são usadas na sub-consulta aninhada; tais variáveis são chamadas de **variáveis de correlação**
- E.g.
  select *nome-cliente*
  from *credor*
  where exists (**select** *
  from *dono-conta*
  where *dono-conta.nome-cliente* =
  credor. nome-cliente)
- Conceitualmente, sub-consulta aninhada é executada uma vez para cada tupla no produto cartesiano gerado pela cláusula **from** externa
  - tal avaliação é chamada **avaliação correlacionada**
  - Nota: outras condições na cláusula where podem ser usadas para calcular uma junção (no lugar de um X) antes de executar a subconsulta aninhada

Otimização das subconsultas aninhadas**(Cont.)

- Avaliação correlacionada pode ser muito ineficiente já que
  - Um grande número de chamadas podem ser feitas à consulta aninhada
  - Pode resultar em E/S de disco aleatórias não necessárias
- Otimizadores SQL tentam transformar subconsultas aninhadas em junções onde for possível, facilitando o uso eficiente das técnicas de junção
- E.g.: a consulta aninhada anterior pode ser reescrita como
  select *nome-cliente*
  from *credor, dono-conta*
  where *dono-conta.nome-cliente = credor. nome-cliente*
  - Nota: a consulta acima não trata corretamente com duplicatas, pode ser modificada para fazer isso
- Em geral, pode não ser possível passar diretamente as relações da consulta aninhada para a cláusula from da consulta externa
  - Uma relação temporária é criada no lugar, e usada no corpo da consulta externa

Otimização das subconsultas aninhadas (Cont.)

Por exemplo, consultas SQL da forma abaixo podem ser reescritas como segue

- Reescrever: **select** …
  from L_1
  where P_1 **and exists** (**select** *
  from L_2
  where P_2)
- To: **create table** t_1 as
  select distinct V
  from L_2
  where P_2^1
  select …
  from L_1, t_1
  where P_1 **and** P_2^2
  - P_2^1 contem predicados em P_2 que não envolvem qualquer variável de correlação
  - P_2^2 re-introduz predicados que envolvem variáveis de correlação, com relações renomeadas apropriadamente
  - V contem todos os atributos usados nos predicados com variáveis de correlação

Optimização das subconsultas aninhadas (Cont.)

- No nosso exemplo, a consulta aninhada original seria transformada para
  create table t_1 **as**
  select distinct *nome-cliente*
  from *dono-conta*
  select *nome-cliente*
  from *credor*, t_1
  where t_1. *nome-cliente* = *credor. nome-cliente*
- O processo de trocar uma consulta aninhada por uma consulta com uma junção (possivelmente com uma relação temporária) é chamado **descorrelação**.
- Descorrelação é mais complicada quando
  - a sub-consulta aninhada usa agregação, ou
  - quando o resultado da sub-consulta aninhada é usada para teste de igualdade, ou
  - Quando a condição ligando a sub-consulta aninhada à outra consulta é **not exists**,

Programação Dinâmica

- **Programação dinâmica** é um método para a construção de algoritmos para a resolução de problemas computacionais, em especial os de otimização combinatória. Ela é aplicável à problemas no qual a solução ótima pode ser computada a partir da solução ótima previamente calculada e memorizada - de forma a evitar recálculo - de outros subproblemas que, sobrepostos, compõem o problema original.

Exemplo

- O exemplo clássico para o entendimento do método é o algoritmo para o cálculo da seqüência de Fibonacci:

```
declara mem <-- mapeamento(0 → 0, 1 → 1)
função fib(n)
{
    se n não está em índices(mem)
        mem[n] <-- fib(n − 1) + fib(n − 2)
    retorna mem[n]
}
```