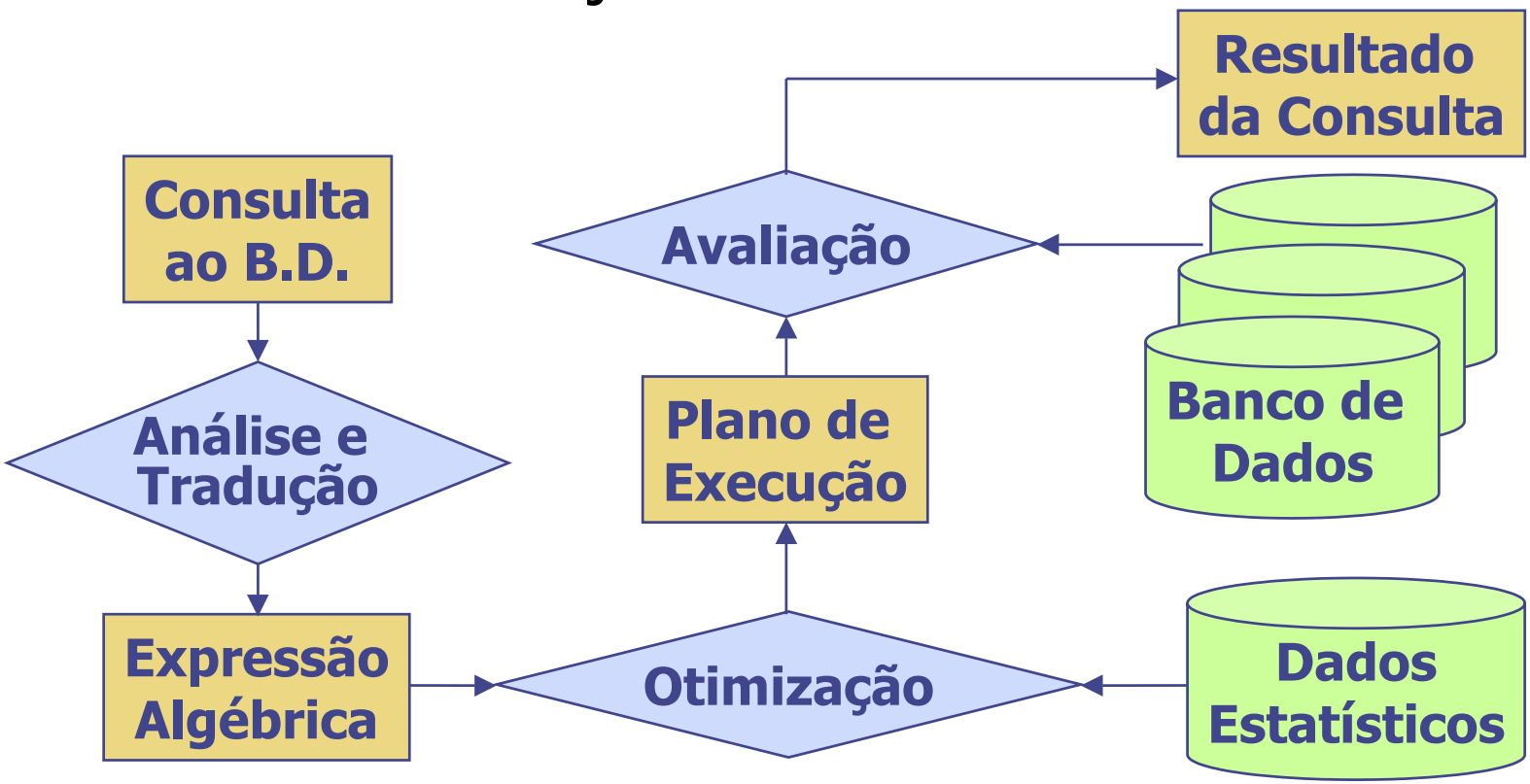
Unidade III

Processamento de Consultas

- Processamento de Operações
- Classificação dos Dados
- Avaliação de Expressões
- Otimização de Consultas

Processamento de Operações

- Passos na execução de consultas



Processamento de Operações

- Passos na execução de consultas
  - Análise e Tradução
    - Verifica a sintaxe da consulta em SQL
    - Confere se as tabelas, colunas e visões existem
    - Traduz para expressão em álgebra relacional
  - Otimização
    - Verifica os planos de execução possíveis
    - Estima o custo de cada plano
    - Seleciona o plano mais eficiente
  - Avaliação
    - Executa o plano selecionado
    - Obtém o resultado da consulta

Processamento de Operações

- Estimativa de Custo
  - Em geral, o custo é considerado como o tempo total gasto para processar a consulta
  - Principais fatores que compõem o custo
    - Custo de acesso ao disco
      - Posicionamento: duração de operações seek
      - Leitura de blocos: duração de operações read
      - Escrita de blocos: duração de operações write
    - Custo de processamento: tempo de uso da CPU
    - Custo de transmissão: tempo gasto para transmitir os dados pela rede, caso seja necessário

Processamento de Operações

- Estimativa de Custo
  - Como simplificação, vamos considerar apenas o número de blocos transferidos para calcular o custo
  - Vamos desprezar:
    - A diferença entre custo de acesso seqüencial e aleatório no disco
    - A disponibilidade de buffers, que podem reduzir o número de acessos ao disco
    - O custo de processamento e de transmissão
    - O custo de escrita de dados no disco
  - Em geral, os sistemas mais robustos levam em conta todos estes fatores

Processamento de Operações

- Catálogo de informações: contém dados sobre as relações do BD usados para estimar o custo

n_r	Número de tuplas da relação r
b_r	Número de blocos que contém tuplas de r
s_r	Tamanho em bytes de uma tupla de r
f_r	Fator de bloco da relação r, ou seja, o número de tuplas que cabem em um bloco do disco
E_{An}	Custo estimado do algoritmo A_n
V(A,r)	Número de valores distintos do atributo A na relação r - se A for uma chave, V(A,r) = n_r
SC(A,r)	Cardinalidade do atributo A de r - se a distribução de A for uniforme, SC(A,r) = n_r/V(A,r) - se A for uma chave, SC(A,r) = 1

Processamento de Operações

- Operação de Seleção
  - Comando mais usado → otimizar ao máximo!
  - Vários algoritmos, com custos diferentes, podem ser usados para implementar seleções
  - Algoritmos para seleção de igualdade por varredura
    - Busca Linear
    - Busca Binária
  - Algoritmos para seleção de igualdade usando índices
    - Índice primário
    - Índice secundário

Processamento de Operações

- Operação de Seleção
  - Algoritmos para seleção de comparação
    - Índice primário
    - Índice secundário
  - Algoritmos para seleções complexas
    - Conjunção
      - Índice simples (uma chave)
      - Índice composto (múltiplas chaves)
      - Intersecção de identificadores
    - Disjunção

Processamento de Operações

- Algoritmo de Busca Linear (A1)
  - Percorre linearmente toda a tabela
  - Custo estimado: $E_{A1} = b_r$
    - Percorre todos os blocos e verifica se cada uma das tuplas satisfaz a condição
  - Se a condição de seleção for de igualdade de um atributo-chave: $E_{A1} = b_r/2$
    - Procura até achar uma tupla que satisfaça a condição, o que na média ocorre no meio da tabela
  - Pode ser aplicado independentemente de:
    - Condição de seleção
    - Ordenação das tuplas
    - Disponibilidade de índices

Processamento de Operações

- Algoritmo de Busca Binária (A2)
  - Usado para achar tuplas com um determinado valor do atributo usado para ordenar a tabela
  - Custo para achar o bloco com a primeira tupla que satisfaz a condição de igualdade: $\log_2(b_r)$
  - Custo para encontrar os demais blocos com tuplas que satisfazem a condição: $SC(A,r)/f_r - 1$
    - Supondo que os blocos são contíguos
    - Supondo uma distribuição uniforme dos dados
    - Supondo que a primeira tupla com um determinado valor aparece no início de um bloco
  - Custo total estimado: $E_{A2} = \log_2(b_r) + SC(A,r)/f_r - 1$

Processamento de Operações

- Algoritmos para seleção de igualdade usando índice primário
  - Igualdade de atributo chave (A3)
    - Procurar no índice e acessar o bloco do disco
    - Custo estimado: $E_{A3} = HT_i + 1$
    - HT_i = Número de níveis de uma árvore B+
  - Igualdade de atributo não-chave (A4)
    - Procurar no índice, e em seguida localizar as demais tuplas que satisfazem a condição
    - Custo para encontrar o primeiro bloco: $HT_i + 1$
    - Custo para achar os demais blocos: $SC(A,r)/f_r - 1$
    - Custo estimado: $E_{A4} = HT_i + SC(A,r)/f_r$

Processamento de Operações

- Algoritmo para seleção de igualdade usando índice secundário (A5)
  - Se a chave de procura é uma chave candidata
    - Procurar no índice e acessar o bloco do disco
    - Custo estimado: $E_{A5} = HT_i + 1$
  - Se a chave de procura não é uma chave candidata
    - Procurar no índice, e em seguida acessar os blocos que contém tuplas que satisfazem a condição
    - Custo estimado: $E_{A5} = HT_i +$ número de blocos
    - Custo no pior caso: $E_{A5} = HT_i + SC(A,r)$ (cada tupla reside em um bloco de disco)

Processamento de Operações

- Algoritmo para seleção de comparação usando índice primário (A6)
  - Na estimativa de custo, vamos supor que metade das tuplas satisfaz a comparação
  - Para seleções do tipo A > V
    - Usar o índice para localizar a primeira tupla que satisfaz a condição e continuar seqüencialmente
    - Custo estimado: $E_{A6} = HT_i + b_r/2$
  - Para seleções do tipo A < V
    - Fazer busca linear até a primeira tupla onde $A \geq V$
    - Custo estimado: $E_{A6} = b_r/2$
    - O índice não é usado!

Processamento de Operações

- Algoritmo para seleção de comparação usando índice secundário (A7)
  - Para seleções do tipo A > V
    - Localizar no índice a primeira tupla que satisfaz a condição e obter ponteiros para tuplas seguintes
  - Para seleções do tipo A < V
    - Obter ponteiros através do índice até a primeira tupla onde A ≥ V
  - Custo estimado: $E_{A7} = HT_i + LB_i/2 + n_r/2$
  - LB_i = número de blocos no último nível do índice i
  - Será mais barato fazer a busca sequencial caso muitas tuplas satifaçam a condição de comparação

Processamento de Operações

- Algoritmos para seleção de conjunção
  - Usando índice simples (A8)
    - Usando o algoritmo que tiver o menor custo (A1...7), obter as tuplas que satisfazem a condição mais restritiva (que retorna menos tuplas)
    - Verificar as outras condições nas tuplas obtidas
    - Desprezando o custo para processar as demais condições, o custo é o mesmo do algoritmo usado
  - Usando índice composto (A9)
    - Usando o algoritmo para busca no índice (A3...5), localizar as tuplas que satisfazem a condição
    - Custo estimado: o mesmo do algoritmo usado

Processamento de Operações

- Algoritmos para seleção de conjunção
  - Fazendo intersecção de identificadores (A10)
    - Para cada condição, buscar em um índice ponteiros para tuplas que a satisfazem
    - Verificar os ponteiros retornados por todas as condições (intersecção), e acessar os dados
    - Se uma condição for em atributo não-indexado, devemos verificá-la nas tuplas retornadas pelas demais condições
    - O custo estimado é a soma do custo para ler os índices com o custo para acessar os blocos com tuplas que satisfazem a todas as condições

Processamento de Operações

- Algoritmo para seleção de disjunção (A11)
  - Requer índices para todos os atributos usados nas condições
    - Se uma das condições for em um atributo não indexado, é necessário fazer a busca linear
  - Para cada condição, buscar em um índice ponteiros para tuplas que a satisfazem
  - Verificar os ponteiros retornados por pelo menos uma condição (união), e acessar os dados
  - O custo estimado é a soma do custo para ler os índices com o custo para acessar os blocos com tuplas que satisfazem a pelo menos uma condição

Processamento de Operações

- Operação de Junção
  - Fazer produto cartesiano de duas tabelas → caro!
  - Aparece com freqüência em seleções → otimizar!
- Algoritmos para implementar junções
  - Junção de laço aninhado
  - Junção de laço aninhado de bloco
  - Junção de laço aninhado indexada
  - Junção merge
  - Junção hash
- Primeiro vamos estimar o tamanho da junção

Processamento de Operações

- Tamanho da junção
  - Produto cartesiano: r x s
    - $n_{r\ x\ t} = n_r * n_t$ tuplas
    - $s_{r\ x\ s} = s_r + s_s$ bytes
  - Junção natural: $r \bowtie s$
    - Se as relações não têm nenhum atributo em comum, $r \bowtie s$ é o produto cartesiano entre r e s
    - Se o atributo é uma chave de r, $n_{r \bowtie s} \leq n_s$
    - Se o atributo em s é uma chave estrangeira de r, $n_{r \bowtie s} = n_s$
    - Se o atributo não for uma chave, estima-se que $n_{r \bowtie s} = (n_r * n_s) / MAX(V(A,r), V(A,s))$

Processamento de Operações

- Algoritmo de junção de laço aninhado
  - Usa dois laços aninhados – um para cada relação
  - A condição de junção é testada em cada passo

```
for each tupla t_r in r do begin
  for each tupla t_s in s do begin
    if condição é válida para (t_r , t_s) then
      junção := junção + (t_r , t_s)
  end
end
```

  - Aceita qualquer tipo de condição; não requer índices
  - Custo no pior caso: $E_{A12} = n_r * b_s + b_r$ blocos
  - Custo cai para $E_{A12} = b_s + b_r$ se s couber na memória

Processamento de Operações

- Algoritmo de junção de laço aninhado de blocos
  - Usa quatro laços aninhados – percorre blocos e tuplas

for each bloco B_r **of** r **do begin**
  for each bloco B_s **of** s **do begin**
    for each tupla t_r **in** B_r **do begin**
      for each tuple t_s **in** B_s **do begin**
        if condição é válida para (t_r , t_s) **then**
          junção := junção + (t_r , t_s).
      end
    end
  end
end

Processamento de Operações

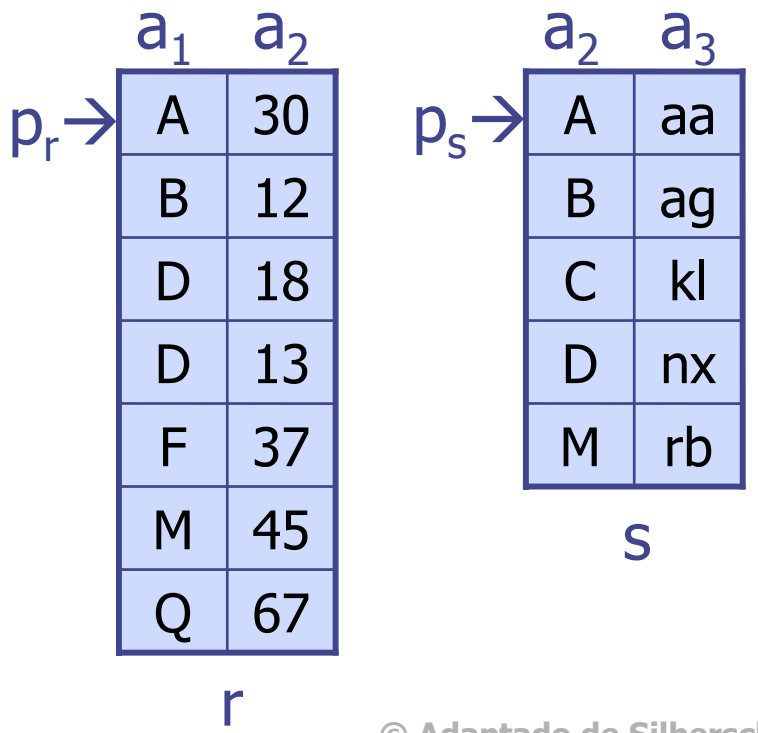
- Algoritmo de junção de laço aninhado de blocos
  - Assim como o algoritmo anterior, este algoritmo aceita qualquer tipo de condição e não requer índices
  - Custo no pior caso: $E_{A13} = b_r * b_s + b_r$ blocos
  - Possível otimização:
    - Se temos M blocos de memória livres, podemos usar M-2 blocos para ler r
    - Os dois outros blocos são usados para ler s e para guardar o resultado da operação
    - Custo: $E_{A13} = [b_r/(M-2)] * b_s + b_r$
  - Custo no melhor caso: $E_{A13} = b_s + b_r$ (caso $M-2 \geq b_r$)

Processamento de Operações

- Algoritmo de junção de laço aninhado indexada
  - Para cada tupla t_r, usa-se o índice para procurar tuplas t_s que satisfaçam a condição com t_r
  - Custo estimado: $E_{A14} = b_r + n_r * c$
    onde c é o custo estimado de fazer uma seleção com a mesma condição usada na junção (A3...5)
  - Inverter r e s se $n_r < n_s$ e se houver um índice em t_r
  - Pode valer a pena criar o índice para fazer a junção

Processamento de Operações

- Algoritmo de junção merge
  - Usado se as duas relações estão ordenadas pelo atributo usado na junção
  - Cada relação é lida com um ponteiro
  - Os ponteiros avançam de acordo com o valor do atributo
  - Bastam dois blocos de memória
  - Custo: $E_{A15} = b_r + b_s$

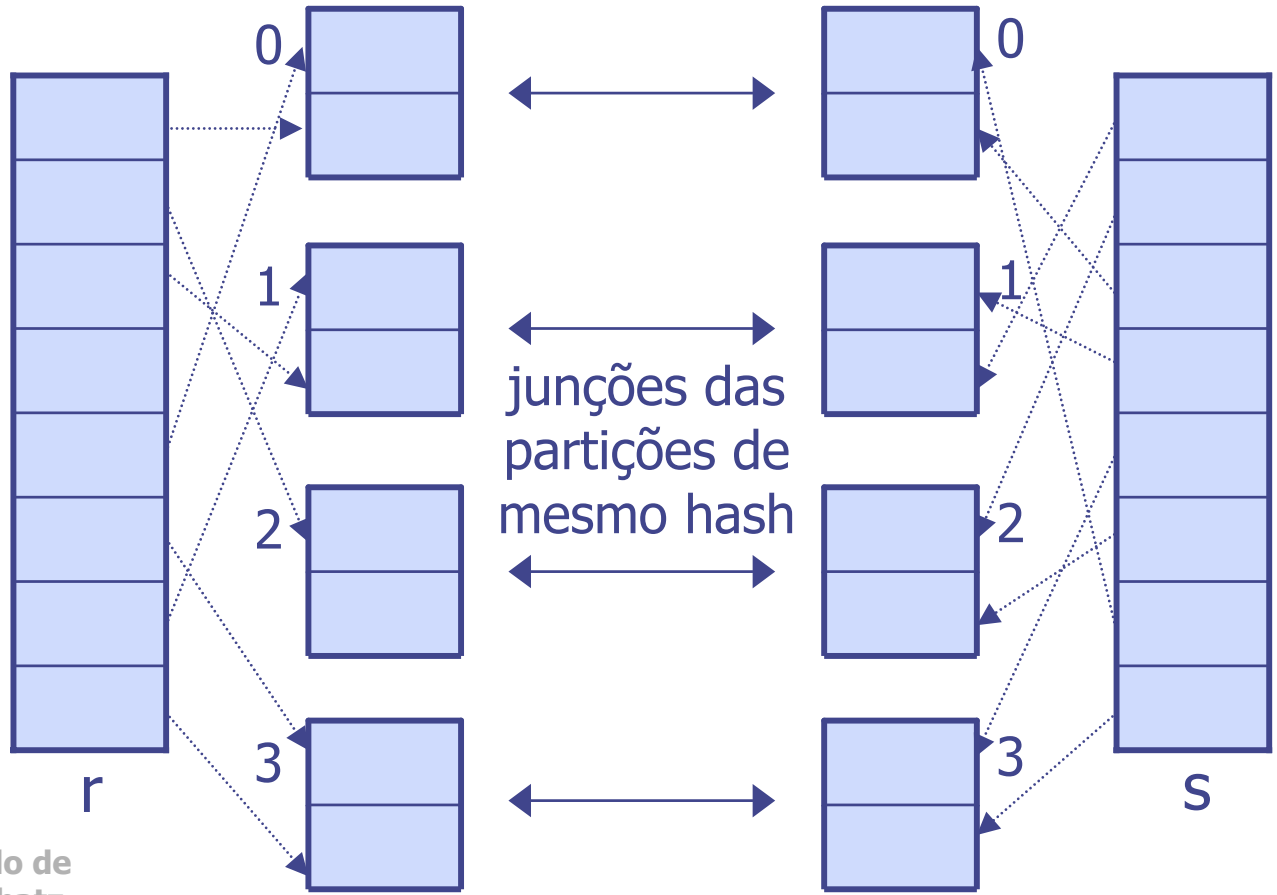


Processamento de Operações

- Algoritmo de junção hash
  - Usa uma função de hash do atributo de junção
  - Divide as duas relações em n blocos
    - r_0, r_1 ... r_n para a relação r
    - s_0 , s_1 ... s_n para a relação s
    - Custo: $2(b_r + b_s)$
  - Os blocos de mesmo hash são comparados
    - r_0 com s_0, r_1 com s_1, ...
    - Custo: $b_r + b_s$
  - Pode ocorrer um overhead de até 2n no caso de haver blocos parcialmente cheios
  - Custo estimado: $E_{A16} = 3(b_r + b_s) + 2n$

Processamento de Operações

- Algoritmo de junção hash



Adaptado de
Silberschatz,
Korth & Sudarshan

Processamento de Operações

- Junção externa
  - Pode ser calculada modificando algoritmos de merge-junção e hash-junção para adicionar ao resultado as tuplas sem correspondentes preenchidas com nulos
  - Junção externa à esquerda: adicionar ao resultado somente as tuplas da relação à esquerda da junção que não tiverem correspondente na relação à direita
  - Junção externa à direita: adicionar ao resultado somente as tuplas da relação à direita da junção que não tiverem correspondente na relação à esquerda
  - Junção externa completa: adicionar todas as tuplas que não tiverem correspondente na relação oposta

Processamento de Operações

- Junção de três ou mais relações
  - Pode-se calcular a junção de duas relações, depois a junção do resultado com a relação seguinte, e continuar até fazer a junção de todas as relações
    - $q \bowtie r \bowtie s = (q \bowtie r) \bowtie s = q \bowtie (r \bowtie s)$
  - Uma alternativa mais eficiente consiste em fazer as junções ao mesmo tempo usando dois índices
    - Calcular $q \bowtie r \bowtie s$ pegando cada tupla t_r e buscando as tuplas correspondentes t_q e t_s nos índices de q e s
    - Assim r é lida somente uma vez → menor custo!

Processamento de Operações

- Junção com condição conjuntiva
  - Opção 1: usar loops aninhados (em bloco ou não)
  - Opção 2:
    - Obter as tuplas que atendem a primeira condição aplicando uma das técnicas vistas anteriormente
    - Verificar a condição seguinte no resultado da condição anterior
    - Continuar até aplicar a última condição
    - No final, restarão apenas as tuplas para as quais todas as condições são válidas

Processamento de Operações

- Junção com condição disjuntiva
  - Opção 1: usar loops aninhados (em bloco ou não)
  - Opção 2:
    - Obter as tuplas que atendem a cada condição aplicando uma das técnicas vistas anteriormente
    - Fazer a união dos resultados de cada uma das condições, eliminando as tuplas duplicadas

Processamento de Operações

- Operações com conjuntos
  - Particionar as relações usando uma função de hash
    - r_0, r_1 ... r_n para a relação r
    - s_0 , s_1 ... s_n para a relação s
  - Processar cada partição usando um índice de hash criado em memória para r_i
  - União: adicionar tuplas de $s_i \notin r_i$ ao índice hash, e depois pôr no resultado todas as tuplas do índice
  - Intersecção: pôr no resultado as tuplas de s_i que estiverem no índice hash de r_i
  - Diferença: remover as tuplas de $s_i \in r_i$ do índice hash, e depois pôr no resultado todas as tuplas do índice

Processamento de Operações

- Agrupamento
  - Fazer hash ou classificação para colocar as tuplas com mesmo valor do atributo de agrupamento no mesmo hash ou em sequência
  - Calcular a função (sum, avg, ...) sobre as tuplas com mesmo valor do atributo de agrupamento
- Eliminação de duplicatas
  - Fazer hash ou classificação para que tuplas duplicadas apareçam no mesmo hash ou em sequência
  - Ler hashs ou a tabela e remover as tuplas duplicadas
- Projeção
  - Selecionar os atributos de interesse
  - Fazer a eliminação das tuplas duplicadas

Classificação dos Dados

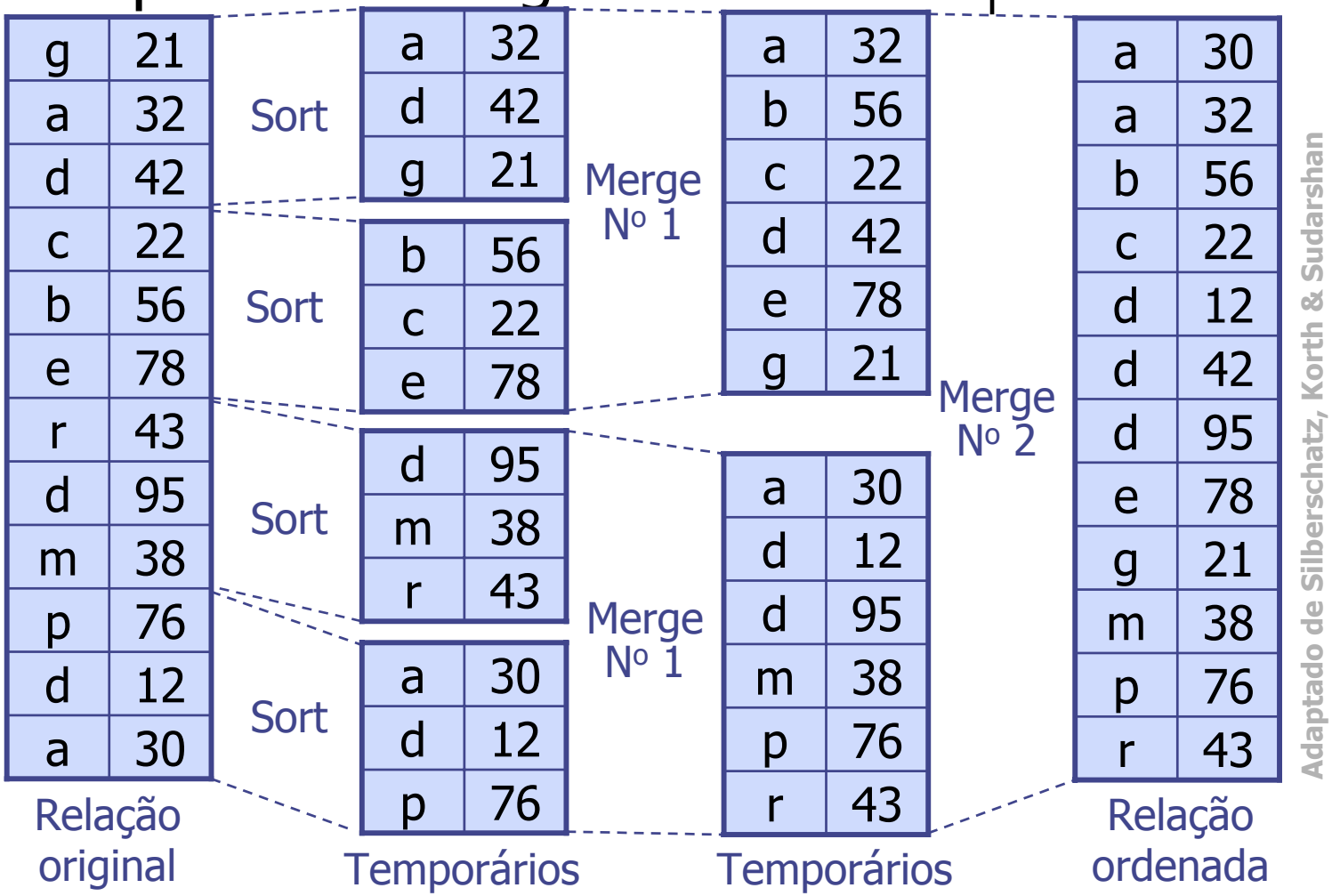
- Se a relação inteira cabe na memória, pode ser usado um algoritmo tradicional de ordenação
  - Exemplo: algoritmo Quicksort
- Algoritmo Sort-Merge
  - Usado se não houver memória suficiente para manter toda a relação (considere M a memória disponível)
  - Fase de Sort:
    - Lê M blocos da relação de cada vez, ordena as tuplas e as coloca num arquivo temporário
    - Repete o passo anterior até ler todos os blocos
    - Ao final, são criados b_r/M arquivos temporários
    - $E_{Sort} = 2b_r$ (para cada bloco lido, um é escrito)

Classificação dos Dados

- Algoritmo Sort-Merge (cont.)
  - Fase de Merge
    - Une M-1 temporários por vez, lendo um bloco de cada e ordenando as tuplas em um novo arquivo
    - Continua até que acabem os temporários
    - Repete passos anteriores até unir todas as tuplas
    - São necessários $\log_{M-1}(b_r/M)$ passos de merge
    - Cada passo tem custo de $2b_r$
    - $E_{Merge} = \log_{M-1}(b_r/M) * 2b_r$
  - Custo total: $E_{Sort\text{-}Merge} = [1 + \log_{M-1}(b_r/M)] * 2b_r$
  - Se o resultado não for escrito no disco, o custo cai para $E_{Sort\text{-}Merge} = [1 + 2\log_{M-1}(b_r/M)] * b_r$

Classificação dos Dados

- Exemplo: Sort-Merge com M=3 e f_r=1



Avaliação de Expressões

- Algoritmos vistos anteriormente nos permitem executar operações individualmente
- Exitem técnicas que permitem avaliar expressões completas de uma só vez
  - Materialização
    - Calcula resultados parciais e grava no disco
    - Segue calculando até completar a expressão
  - Pipeline
    - Tuplas calculadas são passadas imediatamente para outras operações que necessitam delas
    - Pode ser dirigido pelo produtor ou por demanda

Avaliação de Expressões

- Materialização
  - Inicia executando as operações mais simples, para as quais as entradas são as tuplas de relações, e grava (materializa) o resultado no disco
  - Em seguida, executadas operações cujas entradas dependem do resultado de operações já calculadas, e grava o resultado no disco
  - Segue executando as operações para as quais a entrada está disponível, até avaliar toda a expressão
- Análise
  - Materialização sempre pode de ser aplicada
  - O custo de leitura/escrita no disco é muito alto

Avaliação de Expressões

- Pipeline
  - Várias operações são executadas simultaneamente
  - As tuplas calculadas são passadas imediatamente para outras operações que necessitem delas
- Análise
  - Pipeline é usado para algoritmos que geram tuplas durante sua execução, e não somente no final
  - Não pode ser usado em junções merge e hash, nem para operações de classificação
  - Tem custo muito menor, pois não grava resultados intermediários no disco

Avaliação de Expressões

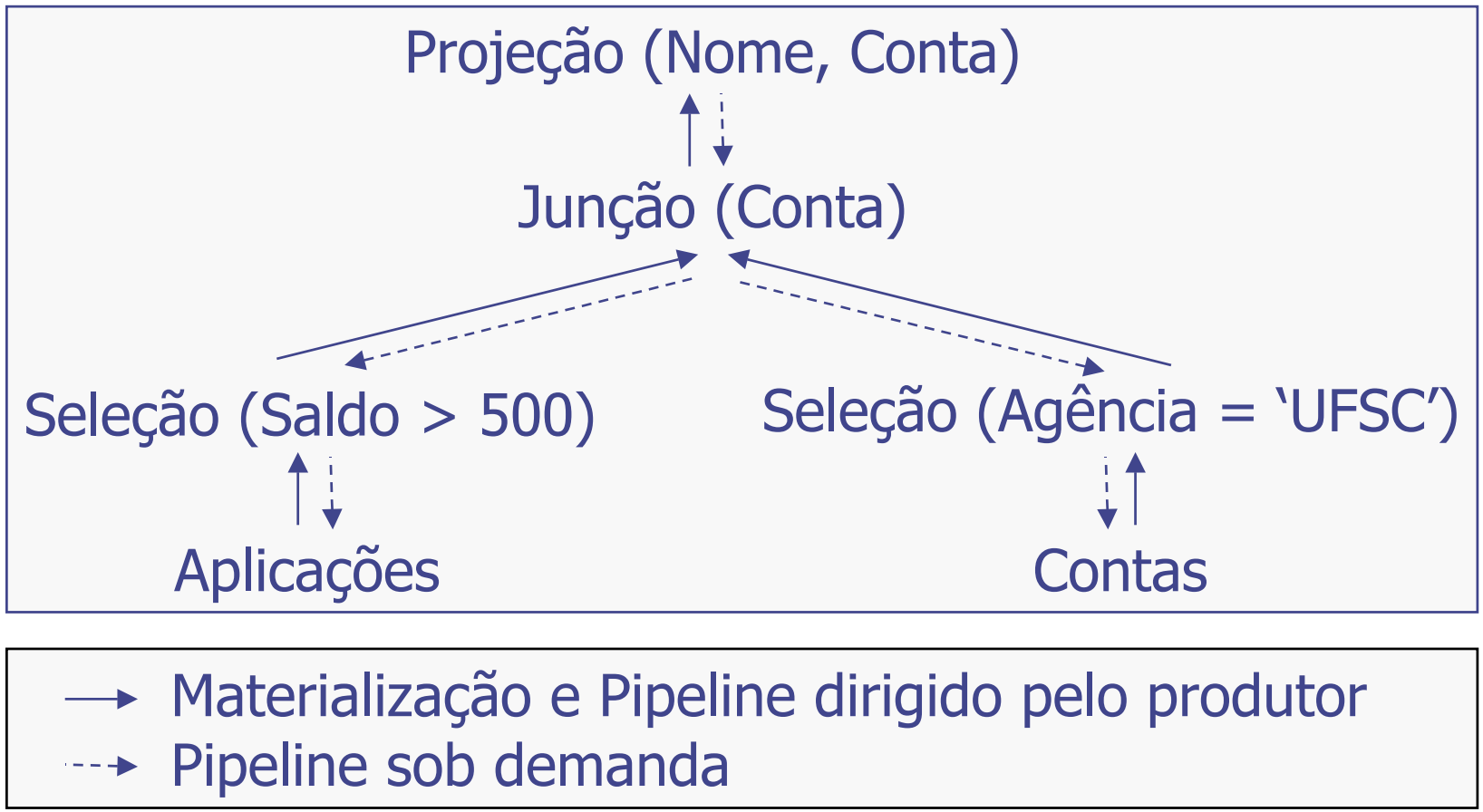
- Pipeline dirigido pelo produtor
  - As operações mais simples vão sendo executadas
  - As tuplas calculadas são armazenadas em buffers
    - Operação fica bloqueada se o buffer estiver cheio
    - O sistema dá prioridade de execução a operações com espaço livre no buffer
  - As tuplas são lidas dos buffers por operação de mais alto nível que usam essas tuplas como entrada
  - O processo segue até a operação de mais alto nível, que obtém o resultado da expressão completa

Avaliação de Expressões

- Pipeline dirigido por demanda
  - As operações geram as tuplas sob demanda
  - Tuplas são solicitadas pelo sistema para a operação de mais alto nível
  - A operação de mais alto nível solicita as tuplas que precisa de operações das quais depende do resultado
  - Operações seguem solicitando tuplas umas às outras, até chegar ao nível mais baixo, na qual seja possível obter as tuplas a partir das relações
  - Operações devem manter estado, para saber o que devem retornar a outras operações

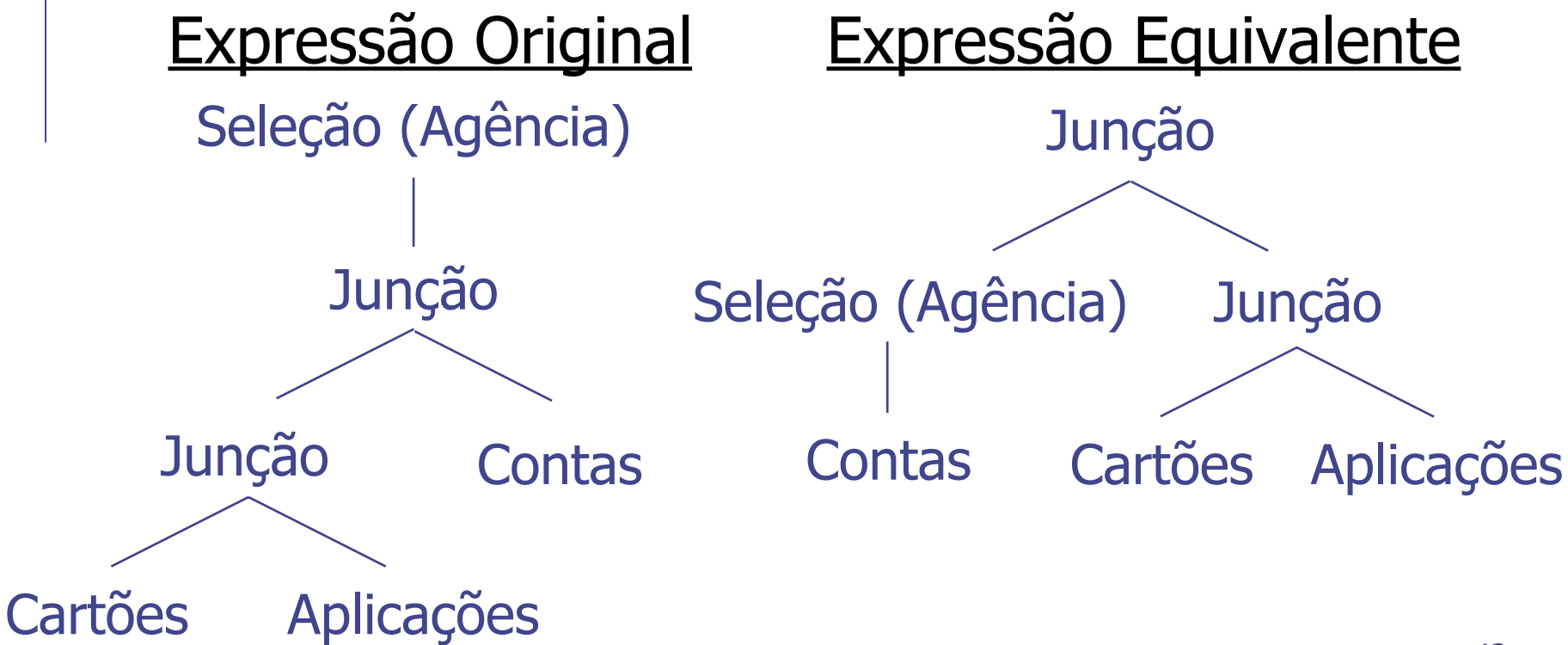
Avaliação de Expressões

- Exemplo



Otimização de Consultas

- Expressões podem ser otimizadas fazendo a troca por expressões equivalentes
- Exemplo:



Otimização de Consultas

- Regras de Equivalência

σ = seleção	θ = predicado
Π = projeção	$\bowtie$ = junção
E = expressão algébrica	L = lista de atributos

  - Seleções conjuntivas podem ser quebradas em seqüências de seleções

    $\sigma_{\theta 1 \wedge} \sigma_{\theta 2}(E) = \sigma_{\theta 1}(\sigma_{\theta 2}(E))$

  - Seleções são comutativas

    $\sigma_{\theta 1}(\sigma_{\theta 2}(E)) = \sigma_{\theta 2}(\sigma_{\theta 1}(E))$

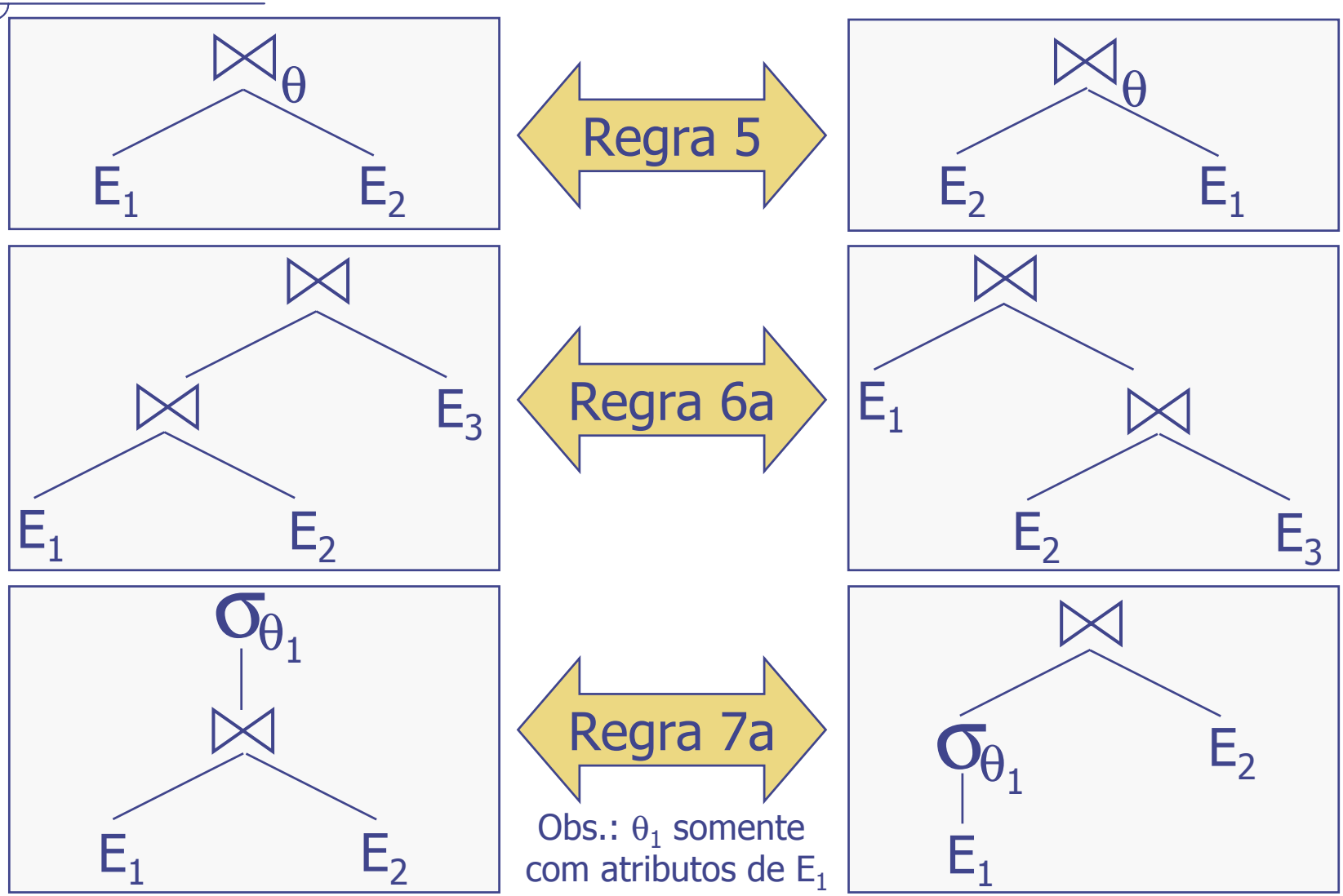
  - Projeções em seqüência são desnecessárias

    $\Pi_{L1}(\Pi_{L2}(...(\Pi_{Ln}(E))...)) = \Pi_{L1}(E)$

Otimização de Consultas

- Regras de Equivalência (cont.)
  - Seleções podem ser combinadas com produtos cartesianos e junções
    a) $\sigma_{\theta}(E_1 x E_2) = E_1 \bowtie_{\theta} E_2$
    b) $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$
  - Junções são comutativas
    $E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$
  - Junções são associativas nos seguintes casos:
    a) $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
    b) $(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$
    onde θ_2 envolve apenas os atributos de E_2 e E_3

Otimização de Consultas



Otimização de Consultas

- Regras de Equivalência (cont.)
  - Seleções podem ser distribuídas na forma de junções, observando as seguintes condições:
    a) $\sigma_{\theta 1}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_{\theta} E_2$
    b) $\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta 1}(E_1)) \bowtie_{\theta} (\sigma_{\theta 2}(E_2))$
    onde θ_1 envolve apenas atributos de E_1 e θ_2 de E_2
  - Projeções podem ser distribuídas por junções
    a) $\Pi_{L1 \cup L2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L1}(E_1) \bowtie_{\theta} \Pi_{L2}(E_2)$
    onde θ envolve apenas os atributos de $L_1 \cup L_2$
    b) $\Pi_{L1 \cup L2}(E_1 \bowtie_{\theta} E_2) = \Pi_{L1 \cup L3}(E_1) \bowtie_{\theta} \Pi_{L2 \cup L4}(E_2)$
    onde os atributos L_3 de E_1 e L_4 de E_2 estão envolvidos em θ mas não pertencem a $L_1 \cup L_2$

Otimização de Consultas

- Regras de Equivalência (cont.)
  - Operações de união e intersecção são comutativas
    a) $E_1 \cup E_2 = E_2 \cup E_1$
    b) $E_1 \cap E_2 = E_2 \cap E_1$
  - Operações de união e intersecção são associativas
    a) $(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$
    b) $(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$
  - Seleções de operações de conjunto são distributivas
    a) $\sigma_\theta (E_1 \cup E_2) = \sigma_\theta (E_1) \cup \sigma_\theta (E_2)$
    b) $\sigma_\theta (E_1 \cap E_2) = \sigma_\theta (E_1) \cap E_2 = \sigma_\theta (E_1) \cap \sigma_\theta (E_2)$
    c) $\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - E_2 = \sigma_\theta (E_1) - \sigma_\theta (E_2)$
  - Projeções podem ser distribuídas por junções
    $\Pi_L(E_1 \cup E_2) = \Pi_L(E_1) \cup \Pi_L(E_2)$

Otimização de Consultas

- O processo de otimização deve:
  - Buscar expressões equivalentes à expressão original com menor custo de execução
  - Montar o plano de avaliação da expressão
- Técnicas de otimização
  - Otimização pelo custo
  - Otimização heurística
- A maioria dos otimizadores utiliza as duas técnicas de otimização cobinadas

Otimização de Consultas

- Otimização pelo custo
  - Verificar o custo de execução da expressão original
  - Procurar expressões equivalentes à original aplicando as regras de equivalência (Obs.: as regras não dizem que forma tem o menor custo)
  - Calcular o custo de avaliação das expressões usando os algoritmos aplicáveis para processar cada operação
  - Selecionar a expressão de menor custo de execução

Otimização de Consultas

- Verificação do custo de expressões equivalentes
  - Com n relações, são (2(n-1))!/(n-1)! expressões equivalentes (n = 5 → 1680; n = 7 → 665.280; n = 9 → 518 milhões de equivalências)
  - Nem todas as expressões equivalentes precisam ter seu custo calculado
  - Estratégia: dividir a expressão em partes e calcular o custo das expressões equivalentes de cada parte
  - Exemplo: dividir uma expressão com 9 relações em 3 partes com 3 relações cada, e calcular o custo das 12 expressões equivalentes de cada parte (36 no total)

Otimização de Consultas

- Otimização Heurística
  - Usa regras gerais para transformar consultas de modo a reduzir seu custo
  - Separe seleções conjuntivas em seleções isoladas
  - Mova as seleções para baixo da ávore de consulta para executá-las primeiro (regras 2, 7a, 7b, 11)
  - Dê preferência às seleções e junções que geram relações menores (regras 6 e 10)
  - Substitua produtos cartesianos seguidos de seleções por junções (regra 4a)
  - Mova as projeções para baixo da árvore de consulta (regras 3, 8a, 8b, 12) e crie outras se for possível
  - Dê preferência a árvores executáveis com pipeline

Otimização de Consultas

- Plano de avaliação
  - Define como cada operação será executada e como toda a expressão será avaliada
  - É a saída do otimizador e a entrada do avaliador
  - Exemplo de plano de avaliação: