

# *Processamento e otimização de Consultas*

Prof. Heloise Manica P. Teixeira

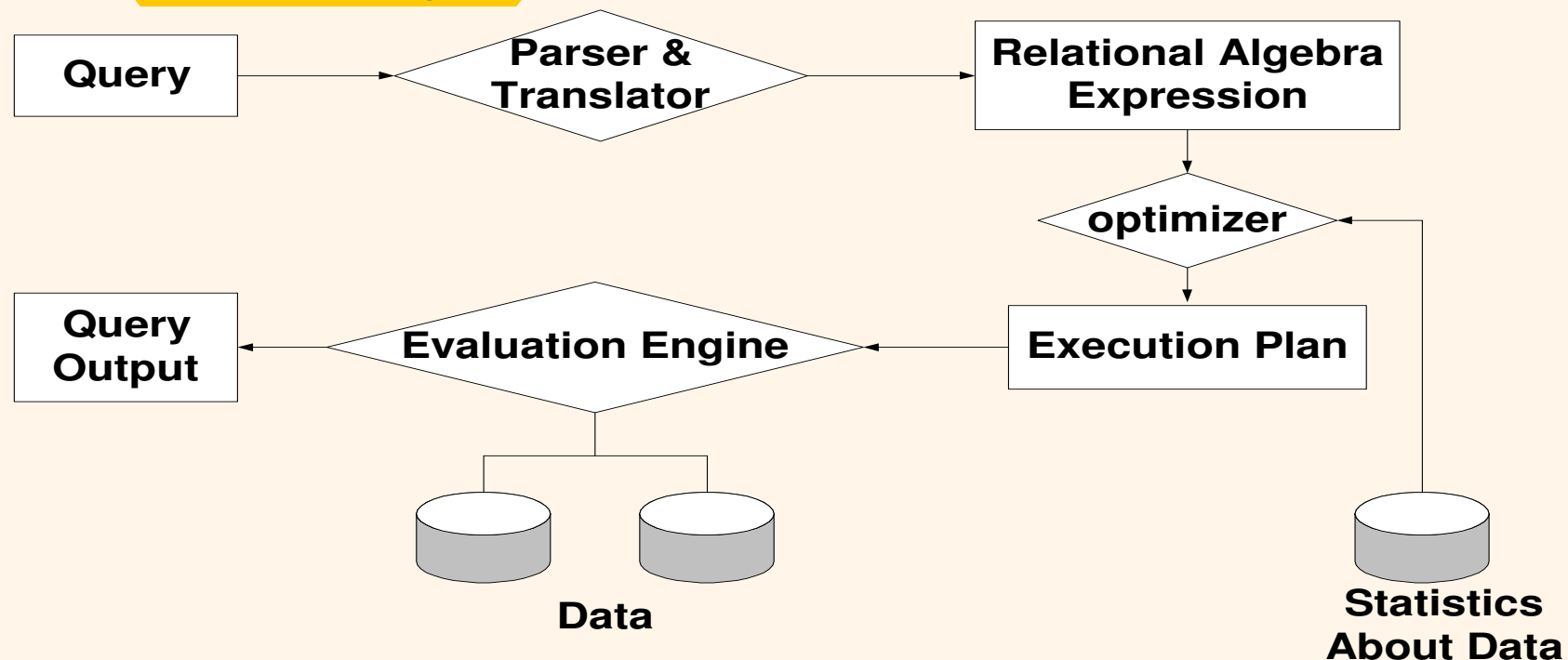


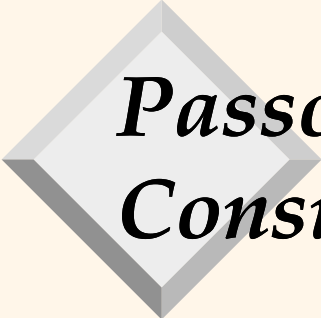
# *Processamento de Consultas*

- ❖ Atividade de extrair dados de um BD
- ❖ Custo do processamento de consulta
  - Acesso a disco
- ❖ Para consultas complexas, há estratégias que reduzem o custo

# *Passos Básicos no Processamento de Consultas*

1. Análise Sintática e tradução
2. Otimização
3. Avaliação





## *Passos Básicos no Processamento de Consultas (Cont.)*

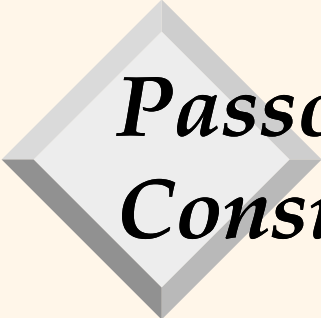
### 1. **Análise Sintática e tradução**

- ❖ SQL – linguagem para uso humano, para o BD álgebra relacional estendida (AR).
- ❖ Analisador sintático **confere a sintaxe**, verifica os nomes das relações, etc. **traduz a consulta** em sua forma interna **→ álgebra relacional**.

# *Passos Básicos no Processamento de Consultas (Cont.)*

## 2. Otimização

- ❖ Processo de selecionar o **plano de avaliação** de consultas **mais eficiente** para uma consulta.
- ❖ Cada consulta SQL pode ser traduzida para AR de vários modos, assim vários planos de avaliação da consulta pode ser definidos.
  - É de **responsabilidade do sistema** construir um plano que minimize os custos (acesso a disco)
  - Para fazer uma boa estimativa do custo, otimizadores usam **informações estatísticas** sobre as relações (tamanho da relação, , disponibilidade e profundidade dos índices, etc.)
- ❖ Exemplo:  $\sigma \text{ saldo} < 2500 (\pi \text{ saldo}(\text{conta}))$   
é equivalente a  $\pi \text{ saldo}(\sigma \text{ saldo} < 2500(\text{conta}))$



## *Passos Básicos no Processamento de Consultas (Cont.)*

### 3. Avaliação

- ❖ Uma vez disponível o plano de execução de consulta, a consulta é avaliada e é produzido o resultado da consulta.
- A sequencia de passos formam a base do processamento de consultas, mas nem todos os BD seguem esses passos exatamente (por ex., alguns usam árvore sintática anotada)



## *Catálogo de Informações para a Estimativa de Custo*

- ❖ A estratégia para avaliação da consulta depende do custo estimado
- ❖ Informações estatísticas relevantes são:
  - $n_r$ : número de tuplas na relação  $r$ .
  - $f_r$ : fator de bloco da relação  $r$  – ou seja - número de tuplas da relação  $r$  que cabe em um bloco.
  - $b_r$ : número de blocos que contêm tuplas da relação  $r$ .
  - $s_r$ : tamanho em bytes de uma tupla da relação  $r$ .



## *Catálogo de Informações para a Estimativa de Custo*

### ❖ Informações estatísticas (cont.)

- $V(A,r)$ : número de valores distintos que aparecem na relação  $r$  para o atributo  $A$ .
  - ◆ Se  $A$  é chave, este valor é igual ao tamanho de  $\pi_A(r)$ .
- $SC(A,r)$ : é a cardinalidade de seleção do atributo  $A$  da relação  $r$ .
  - ◆ número médio de registros que satisfazem uma condição de igualdade no atributo  $A$ .



# Catálogo de Informações para a Estimativa de Custo

- ❖ Se as tuplas da relação  $r$  estiverem armazenadas fisicamente juntas em um arquivo, a seguinte equação é válida:

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

Número de Blocos ←

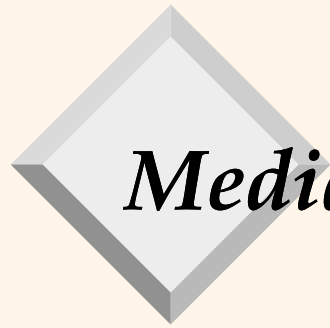
Número de tuplas

Número de tuplas que cabem em um bloco



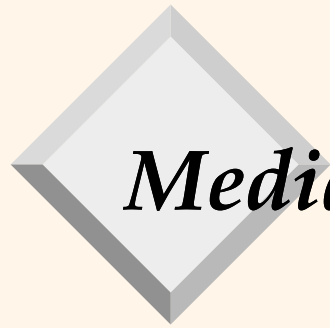
## *Catalogo de informações sobre índices*

- ❖ Além do catálogo de informações, o seguinte catálogo de **informações sobre índices** também é usado (entre outras).
  - $f_i$ : fan-out (número de entradas) médio dos nós internos do índice  $i$  para índices estruturados em árvore, como árvores-B+.
  - $HT_i$ : número de níveis no índice  $i$  – ou seja, a altura do índice  $i$ .
  - $LB_i$ : o número de blocos no nível de folha do índice.
- ❖ Se desejarmos manter estatísticas precisas, toda vez que uma relação for modificada, as estatísticas **devem ser atualizadas**, o que gera um **overhead**
  - Maioria dos SBD não atualiza as estatísticas em todas as alterações



## *Medidas do Custo de uma Consulta*

- ❖ O custo de avaliação de uma consulta pode ser medido por meio de vários recursos diferentes, incluindo:
  - acessos a disco, tempo de CPU e, em um SBD distribuído ou paralelo → custo de comunicação.
- ❖ Normalmente, os acessos a disco é o custo mais importante e fácil de estimar. Portanto, o **número de blocos** transferidos do disco é a medida utilizada para o custo de avaliação.



## *Medidas do Custo de uma Consulta*

- ❖ Assume-se que todas as transferências de blocos têm o mesmo custo
- ❖ Os custos dos algoritmos dependem do tamanho do buffer na memória principal.
- ❖ Quanto maior o tamanho da memória, menor a quantidade de acessos a disco.
- ❖ No pior caso, o buffer pode manter apenas alguns blocos de dados. Geralmente, faremos a suposição do pior caso.



## *Operação de Seleção*

- ❖ **Varredura de arquivos** – são algoritmos de procura que localizam e recuperam os registros que estão de acordo com uma condição de seleção.
- ❖ Nos referimos ao custo estimado de um algoritmo  $A$  como  $E_A$ .
  - Ignoramos o custo de escrever o resultado final de uma operação para o disco.
- ❖ Algoritmos básicos para a operação de seleção:
  - A1 (Busca Linear)
  - A2 (Busca binária)



## *Operação de Seleção (Cont.)*

### ❖ A1 (busca linear)

- cada bloco do arquivo é varrido e todos os registros são testados para verificar se satisfazem a condição de seleção.
- Custo estimado (número de blocos de disco varrido),  $E_{A1}=b_r$
- Para uma seleção em um atributo chave,  $E_{A1} = (b_r / 2)$
- Busca Linear pode ser aplicado indiferentemente a
  - ◆ Condição de seleção, ou
  - ◆ Ordem dos registros do arquivo, ou
  - ◆ Disponibilidade de índices.

## Operação de Seleção (Cont.)

### ❖ A2 (busca binária)

- Aplicada se o **arquivo é ordenado** e a seleção é uma **comparação** de igualdade no atributo.
- Baseia-se na hipótese de que os blocos de uma relação são armazenados no disco de forma **contígua**.
- Estimativa de **Custo** (número de blocos de disco que será varrido):

$$E_{A2} = \underbrace{\lceil \log_2 (b_r) \rceil}_{\text{custo para localizar a primeira tupla}} + \left\lceil \frac{SC(A,r)}{f_r} - 1 \right\rceil$$

$SC(A,r)$  → número de registros que satisfarão a seleção  
 $f_r$  → número de tuplas que cabem em um bloco.

- Se a condição de igualdade estiver em um atributo-chave:
  - ◆  $SC(A,r) = 1$ ; e a estimativa de custo se reduz a  $E_{A2} = \lceil \log_2 (b_r) \rceil$

## Informações estatísticas – Exemplo

- ❖ Suponha que temos a seguinte informação estatística sobre a relação conta:
  - $f_{\text{conta}} = 20$  (número de tuplas que cabem em um único bloco)
  - $V(\text{nome\_agencia}, \text{conta}) = 50$  (50 agências diferentes)
  - $V(\text{saldo}, \text{conta}) = 500$  (500 valores diferentes de saldo)
  - $n_{\text{conta}} = 10.000$  (a relação conta possui 10.000 tuplas)
- ❖ Considere que existem os seguintes índices em conta:
  - ◆ Um índice primário, árvore B + para o atributo nome\_agência
  - ◆ Um índice secundário, árvore B + para o atributo saldo

Qual o custo total da busca abaixo usando busca binária e busca linear?  
Considere que conta esteja ordenada por nome\_agência.

$$\sigma_{\text{nome\_agência}} = \text{"Perryridge"}(\text{conta}) \quad E_{A2} = \lceil \log_2 (b_r) \rceil + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil - 1$$



## Exemplo de Estimativa de custo

$$E_{A2} = \lceil \log_2 (b_r) \rceil + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil - 1$$

$\sigma_{\text{nome\_agência}=\text{"Perryridge"}(\text{conta})}$

- Busca binária para encontrar o primeiro registro =  $\lceil \log_2(500) \rceil = 9$  acessos
  - ◆ Número de blocos é  $b_{\text{conta}} = 500$  (10.000 tuplas; cada bloco armazena 20 tuplas)
- $V(\text{nome\_agência}, \text{conta}) = 50$ 
  - ◆ 50 agências diferentes
- $10.000 \text{ tuplas} / 50 = 200$ 
  - ◆ estima-se que 200 tuplas pertençam a agência Perryridge.
- $200 / 20 = 10$  blocos para estas tuplas.
  - ◆ 20 (número de tuplas que cabem em um único bloco)

O custo total da busca binária seria  $9 + 10 - 1 = 18$  acessos de bloco (versus 500 para busca linear)



## *Seleções Usando Índices*

- ❖ Algoritmos que usam índice são chamados de Varredura de índice
- ❖ Deve-se considerar o custo de **acesso** aos blocos que contém o **índice**
- ❖ O **predicado da seleção** é usado na escolha do índice para processamento da consulta.



## *Seleções Usando Índices*

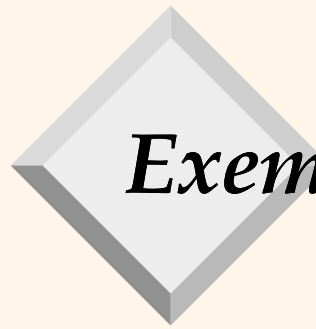
### ❖ A3 (índice primário, igualdade na chave)

– → para uma comparação de igualdade em um atributo chave com um índice primário.

◆ Recupera um único registro que satisfaz a condição

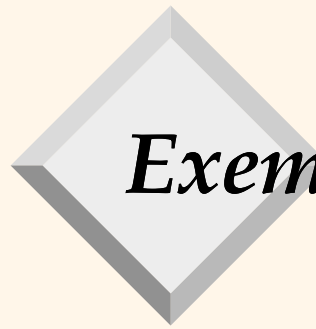
◆  $E_{A3} = HT_i + 1$

– HT: número de níveis do índice



## *Exemplo de Estimativa de Custo (Índices)*

- ❖ Considere a consulta  $\sigma_{\text{nome\_agência}=\text{"Perryridge"}}(\text{conta})$ , com um índice primário em nome\_agência.
- ❖ Se  $V(\text{nome\_agência}, \text{conta}) = 50$ , estima-se que  $10.000/50 = 200$  tuplas da relação conta pertençam à agência Perryridge.
- ❖ Uma vez que o índice é clustering,  $200/20 = 10$  leituras de blocos são necessárias para ler as tuplas de contas.



## *Exemplo de Estimativa de Custo (Índices)*

- ❖ Suponha que o índice árvore B+ armazene 20 ponteiros por nó, então o índice árvore B+ deve ter entre 3 e 5 nós folha e a árvore inteira tem uma profundidade de dois 2. Então, **2 blocos de índice** devem ser lidos.
- ❖ Assim, a estratégia precedente requer um total de **12 leituras** de blocos.

**E se nome\_agencia fosse uma chave-primária?**

# Seleções Usando Índices

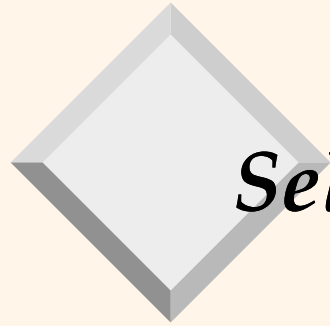
- ❖ **A4** (índice primário, igualdade em atributo que não é chave)
  - Retorna múltiplos registros.

$$E_{A4} = HT_i + \left\lceil \frac{SC(A,r)}{f_r} \right\rceil$$

Diagram illustrating the formula for  $E_{A4}$  (primary index, equality on non-key attribute):

- $SC(A,r)$  is labeled "Cardinalidade da seleção" (Cardinality of the selection).
- $f_r$  is labeled "Numero de tuplas em cada bloco" (Number of tuples in each block).

- ❖ **A5** (índice secundário, igualdade).
  - Recupera um único registro se o campo de indexação for uma chave  
 $E_{A5} = HT_i + 1$
  - Recupera registros múltiplos se o campo de indexação for um atributo não chave (cada registro pode estar em diferentes blocos).  
 $E_{A5} = HT_i + SC(A,r)$



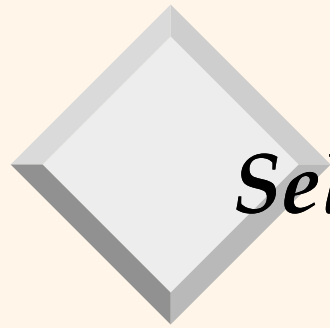
## *Seleções que envolvem Comparações (< >)*

- ❖ A6(índice primário, comparação). A estimativa de custo é:

$$E_{A6} = HT_i + \left\lceil \frac{C}{f_r} \right\rceil$$

onde c é o número de valores que satisfazem a condição e  
fr é o número de tuplas em cada bloco

- ❖ Na falta de informações estatísticas,  $c = n_r/2$ 
  - n = número de tuplas



## *Seleções que envolvem Comparações*

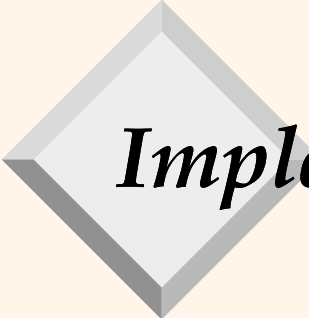
- ❖ **A7**(índice secundário, comparação). A estimativa de custo é :

$$E_{A7} = HT_i + \frac{LB_i}{2} + \frac{n_r}{2}$$

Número de blocos no  
nível folha

- ❖ Embora os algoritmos mostrem que os índices são úteis no processamento de seleções com comparações, eles nem sempre são úteis1
  - Ver exemplo cap. Processamento de Consultas (cap 12) Silberchatz.





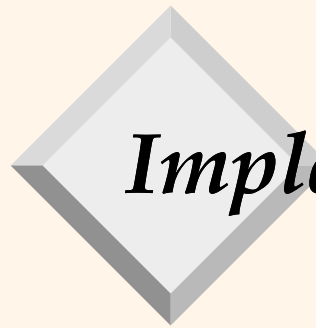
## Implementação de Seleções Complexas

- ❖ A seletividade de uma condição  $\theta_i$ , denotado por  $s_i$ , é a probabilidade que uma tupla na relação  $r$  satisfará  $\theta_i$ . Se  $s_i$  é o número de tuplas satisfatórias em  $r$ , a seletividade de  $\theta$  é  $s_i/n_r$ .
- ❖ **Conjunção:**  $\sigma_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n}(r)$ . A probabilidade de que uma tupla satisfaça todas as condições é dada por:

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

- ❖ **Disjunção:**  $\sigma_{\theta_1 \vee \theta_2 \vee \dots \vee \theta_n}(r)$ . A probabilidade de que uma tupla satisfaça a disjunção é dada por 1 menos a probabilidade de que ela não satisfaça nenhuma das condições:

$$1 - [1 - s_1/n_r] * [1 - s_2/n_r] * \dots * [1 - s_n/n_r]$$



## *Implementação de Seleções Complexas*

- ❖ **Negação:**  $\sigma_{\neg\theta}(r)$ . São as tuplas de  $r$  que não estão em  $\sigma_{\theta}(r)$ . O tamanho estimado de  $\sigma_{\neg\theta}(r)$  é:


$$\text{Tamanho}(r) - \text{tamanho}(\sigma_{\theta}(r))$$

- ❖ **A8** (*seleção de conjunção usando um índice*). Seleciona uma das condições  $\theta_i$  e utiliza um dos algoritmos A1 até A7 que resulta em um menor custo para  $\sigma_{\theta_i}(r)$ . Completamos operação testando, no buffer da memória, se cada registro recuperado satisfaz ou não as condições restantes.



## *Algoritmos para Seleções Complexas*

- ❖ **A9** (*seleção de conjunção usando índice composto*). Se disponível, utiliza um índice composto apropriado.
- ❖ **A10** (*seleção de conjunção por meio da **interseção** de identificadores*). Requer índices com registros de ponteiros nos campos envolvidos nas condições individuais.
  - Cada índice é varrido em busca de ponteiros para as tuplas que satisfaçam uma condição individual.
  - A interseção de todos os ponteiros recuperados é o conjunto de ponteiros para tuplas que satisfaçam a condição conjuntiva.
  - Então, usamos os ponteiros para recuperar os registros de fato. Se os índices não estão disponíveis em todas as condições individuais, então os registros recuperados são testados em relação às condições restantes. (ver exemplo slide 24)
- ❖ **A11** (*seleção de disjunção por meio da **união** de identificadores*). Aplicável em todas as condições que possuem índices disponíveis. Caso contrário, utiliza uma busca linear.



## *Exemplo de Estimativa de Custo para Seleções Complexas*

- ❖ Para ilustrar os algoritmos precedentes, suponha a seguinte consulta:

Select número\_conta

From conta

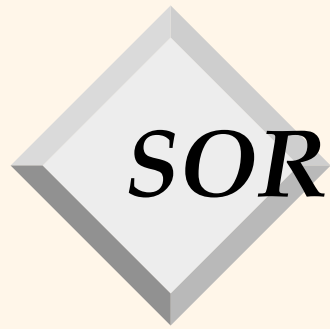
Where nome\_agência = "Perryridge" and saldo = 1200



## *Exemplo (cont.)*

❖ **Considerando o algoritmo A10:**

- Usamos o índice saldo para recuperar o conjunto  $S_1$  de ponteiros para registros com saldo = 1200.
- Usamos o índice nome\_agência para recuperar o conjunto  $S_2$  de ponteiros para registros com nome\_agência = “Perryridge”.
- $S_1 \cap S_2$  = conjunto de ponteiros para registro com nome\_agência = “Perryridge” e saldo = 1200.
- Esta técnica exige que ambos os índices sejam acessados. Cada índice têm uma altura de 2, **somando 4 acessos a blocos de índices.**
- Sabendo que  $V(\text{nome\_agência}, \text{conta})=50$  e  $V(\text{saldo}, \text{conta})=1.000$ , Estimamos que **uma tupla** em 50.000 ( $50 * 1.000$ ) tem simultaneamente nome\_agência = “Perryridge” e saldo = 1200.
- Já que  $n_{\text{conta}} = 10.000$ , estima-se que  $S_1 \cap S_2$  tenha apenas **um ponteiro.**
- O total da estimativa de custo desta estratégia **são cinco leituras de blocos.**



## ***SORTING (Classificação)***

- ❖ A classificação de dados é importante em sistemas de BD:
  - as consultas SQL podem especificar que o resultado seja apresentado ordenadamente;
  - diversas operações relacionais, como as junções, podem ser implementadas eficazmente se as relações de entrada forem primeiramente classificadas.



## ***SORTING** (Classificação)*

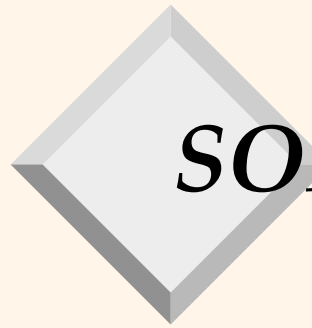
- ❖ Ordenar a relação logicamente (usando índice) pode conduzir a um acesso de disco para cada tupla. Por isso é desejável uma ordenação física.
- ❖ Se a relação cabe na memória, as técnicas padrão de classificação podem ser usadas (ex. quicksort)
- ❖ **Problema**: quando a relação a ser classificada não cabe na memória (classificação externa)
  - A técnica mais comum usada para a **classificação externa** é o algoritmo de **sort-merge externo**.



# ***SORT-MERGE EXTERNO***

- ❖ Seja  $M$  o tamanho de memória:
- ❖ primeiro estágio são executadas várias classificações temporárias:
  - $i = 0$ ;
  - repeat
    - leia  $M$  blocos da relação, ou o resto da relação, o que for menor;
    - ordene a parte da relação que está na memória;
    - escreva os dados ordenados no arquivo temporário  $R_i$ ;
    - $i = i+1$ ;
  - until o fim da relação





## ***SORT-MERGE EXTERNO (CONT.)***

- ❖ Na **segunda** etapa, faz-se o **merge** nos  $N$  arquivos temporários. Supomos que o número total de temporários  $N$ , seja menor que  $M$  (blocos da relação).

**repeat**

escolha a 1ª tupla (na ordem de classificação) entre todas as páginas do buffer;

escreva a tupla no resultado e apague-a da página de buffer;

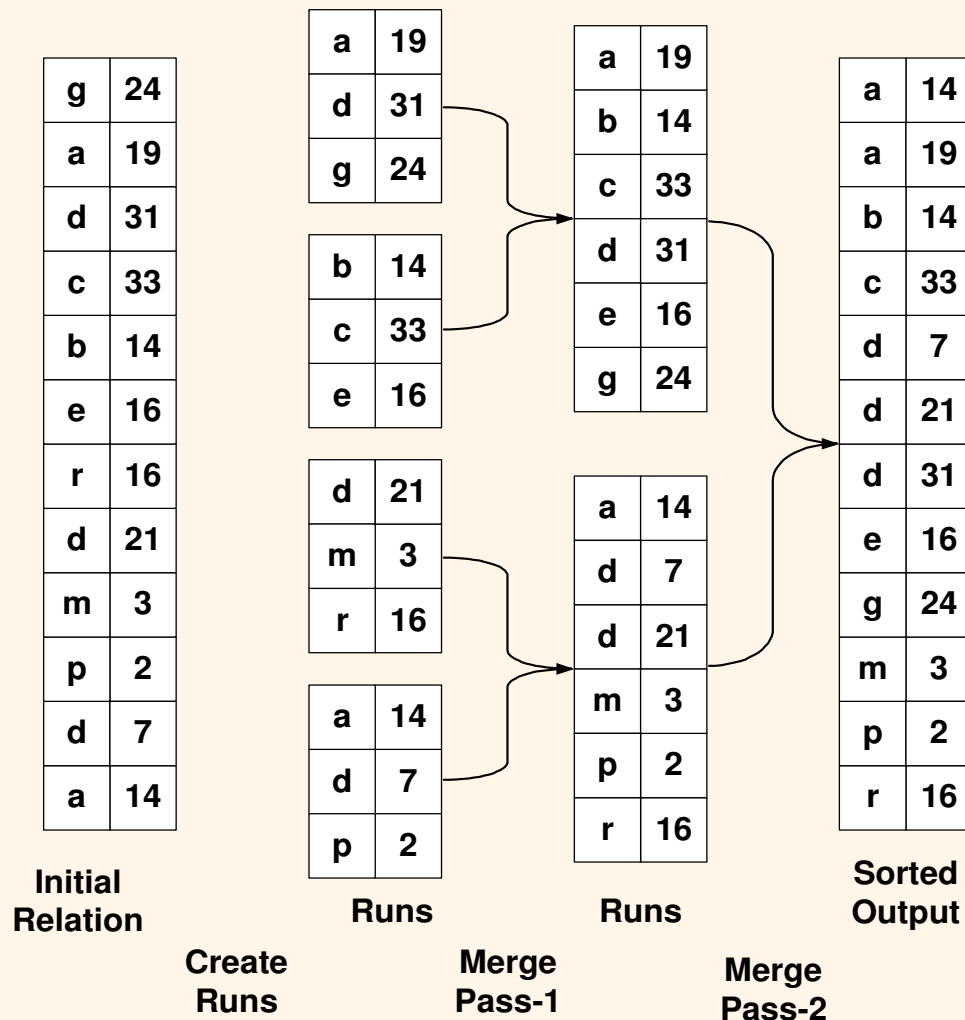
if página de buffer de qqer temporário  $R_i$  está vazia and not fim de arquivo( $R_i$ )

then leia o próximo bloco de  $R_i$  na página de buffer;

until todas as páginas de buffer que estiverem vazias

- ❖ O resultado da página do merge é a relação classificada

# EXEMPLO: CLASIFICAÇÃO EXTERNA USANDO SORT-MERGE



1º estágio: classificações temporárias

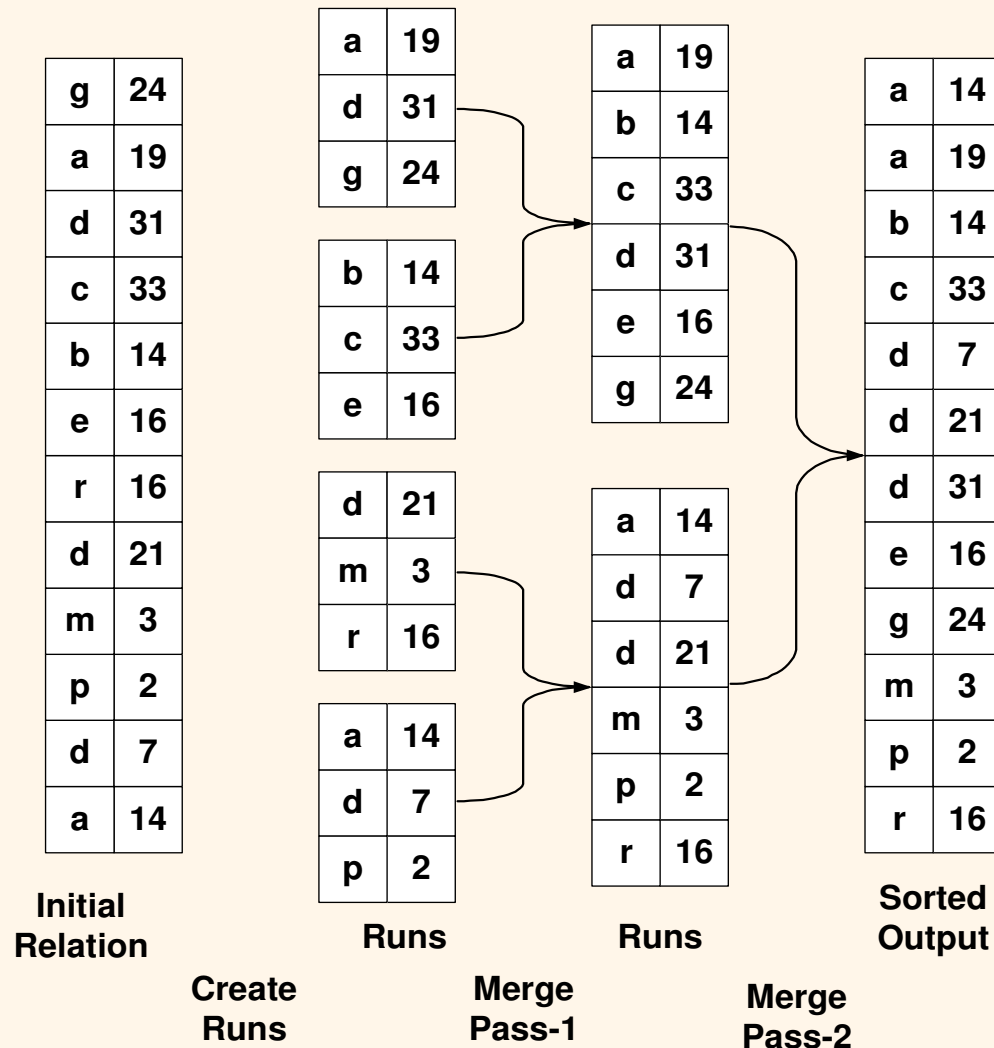
2º estágio: merge – se a relação é muito maior que a memória, pode ser necessário outras fases de merge.

Supomos que:

- apenas 1 tupla cabe em um bloco,
- a memória mantém no máximo 3 frames de página.

Durante o merge dois frames são usados para entrada e um para saída

## ESTIMATIVA DE CUSTO



Quantas transferências de bloco são necessárias para o sort-merge externo?

$$E = b_r(2 \lceil \log_{M-1} (b_r/M) \rceil + 1)$$

$$E = 12 * (2 \lceil \log_{3-1} (12/3) \rceil + 1)$$

$$E = 12 * (4 + 1) = 60$$

transferências de blocos.

$b$  = numero de blocos necessários para R  
 $M$  = temporários



# OPERAÇÃO DE JUNÇÃO

- ❖ Existem diferentes algoritmos para calcular a junção, com diferentes custos:
  - Junção de laço aninhado;
  - Junção de laço aninhado de blocos;
  - Junção de laço aninhado Indexada;
  - Merge-junção;
  - Hash-Junção.

R

A	B	C
2	a	e
1	c	a
4	d	b
1	c	m

S

D	E	F
c	d	q
q	f	e
a	g	e
b	a	b
e	d	g

$R \bowtie_{(R.C=S.F)} S$

A	B	C	D	E	F
2	a	e	q	f	e
2	a	e	a	g	e
4	d	b	b	a	b



# OPERAÇÃO DE JUNÇÃO: EXEMPLO

depositante [x] cliente

Cliente (nome\_cliente, rua, cidade)

Depositante (nome\_cliente, num\_conta)

Catálogo de informação sobre duas relações:

- ❖  $n_{\text{cliente}} = 10.000$ .
- ❖  $f_{\text{cliente}} = 25$ , o que implica  $b_{\text{cliente}} = 10.000/25 = 400$ .
- ❖  $n_{\text{depositante}} = 5.000$ .
- ❖  $f_{\text{depositante}} = 50$ , o que implica  $b_{\text{depositante}} = 5.000/50 = 100$ .
- ❖  $V(\text{nome\_cliente}, \text{depositante}) = 2.500$ , o que implica que, em média, cada cliente tem duas contas.
- ❖ Considere também que o nome\_cliente em depositante seja uma chave estrangeira.



## *ESTIMATIVA DO TAMANHO DAS JUNÇÕES:*

- ❖ O produto cartesiano  $r \times s$  contém  $n_r * n_s$  tuplas; cada tupla de  $r \times s$  ocupa  $s_r + s_s$  bytes.
- ❖ Se  $R \cap S = 0$ , ou seja, as relações não tem nenhum atributo em comum; então  $r \bowtie s$  é igual a  $r \times s$ , e podemos usar nossa técnica de estimativas para produtos cartesianos.



## ESTIMATIVA DO TAMANHO DAS JUNÇÕES:

- ❖ Se  $R \cap S$  é uma **chave estrangeira para R**, então o  $n^\circ$  de tuplas em  $r \bowtie s$  é exatamente **igual** ao  $n^\circ$  de tuplas em  $s$ .
- ❖ Em nosso ex. de **depositante**  $\bowtie$  **cliente**, nome-cliente em depositante é uma chave estrangeira de cliente;
  - o tamanho do resultado é exatamente  **$n_{\text{depositante}}$** , que é **5.000**.

## ESTIMATIVA DO TAMANHO DAS JUNÇÕES: (CONT.)

- ❖ Caso mais difícil: Se  $R \cap S = \{A\}$  não é uma chave para R ou S.
- ❖ Considerando todas as tuplas em r, estimamos:

$$\frac{n_r * n_s}{V(A,s)} \quad \text{tuplas em } r \bowtie s$$

- ❖ Observe que, se invertêssemos os papéis de r e s na estimativa precedente, obteríamos uma estimativa de:

$$\frac{n_r * n_s}{V(A,r)} \quad \text{tuplas em } r \bowtie s$$

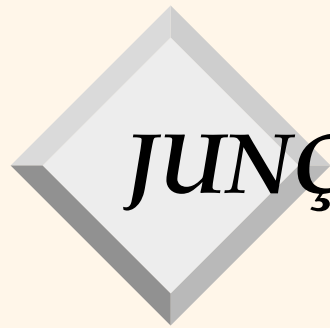
- ❖ Essas **duas estimativas** diferem se  $V(A,r) \neq V(A,s)$ 
  - A **mais baixa** das duas estimativas provavelmente é a **mais precisa**.





## *ESTIMATIVA DO TAMANHO DAS JUNÇÕES: (CONT.)*

- ❖ Calcular uma estimativa do tamanho para **depositante**  $\bowtie$  **cliente** sem utilizar as informações sobre chaves estrangeiras.
- ❖  $V(\text{nome\_cliente}, \text{depositante}) = 2.500$  e
- ❖  $V(\text{nome\_cliente}, \text{cliente}) = 10.000$ ,
- ❖ as duas estimativas que obtemos são
  - $5.000 * 10.000 / 2.500 = 20.000$  e
  - $5.000 * 10.000 / 10.000 = 5.000$ , **escolhemos a menor.**
- ❖ → A mais baixa dessas estimativas é igual àquela calculada anteriormente usando as informações sobre chaves estrangeiras.



# *JUNÇÃO DE LAÇO ANINHADO*

- ❖ O procedimento a seguir mostra um algoritmo simples para calcular a junção teta,  $r \bowtie s$ , de duas relações  $r$  e  $s$ .

**for** each tupla  $tr$ , in  $r$  **do** begin

**for** each tupla  $ts$  in  $s$  **do** begin

        teste o par  $(tr, ts)$  para ver se satisfazem a condição de junção teta

        se satisfazem, adicione (concatenando)  $tr.ts$  ao resultado.

**end**

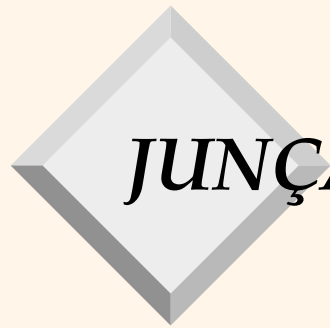
**end**

- ❖  $r$  é chamado de relação **externa** e  $s$  de relação **interna** da junção.
- ❖ Para cada registro em  $r$  temos que realizar uma varredura completa em  $s$ .



## *JUNÇÃO DE LAÇO ANINHADO (CONT.)*

- ❖ O algoritmo de junção de laço aninhado **não requer índices**, e pode ser usado seja qual for a condição de junção.
- ❖ O algoritmo é **caro**, já que examina todos os pares de tuplas nas duas relações.
- ❖ No **pior caso**, um total de  $n_r * b_s + b_r$  acessos de disco serão necessários;
- ❖ Se a relação menor ajustar completamente em memória, usa-se esta como relação interna.
- ❖ Isto reduz a estimativa de custo a acessos de disco de  $b_s + b_r$ 
  - o mesmo custo para o caso em que ambas relações cabem na memória. 43



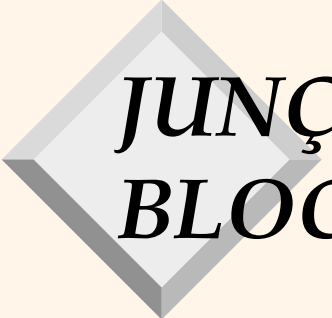
## *JUNÇÃO DE LAÇO ANINHADO (CONT.)*

- ❖ **Exemplo:** junção natural de depositante  $\bowtie$  cliente.
- ❖ Assumindo o pior caso, a estimativa de custo será
$$n_{\text{dep}} * b_{\text{cliente}} + b_{\text{dep}} =$$
$$5.000 * 400 + 100 = 2.000.100 \text{ acessos de disco.}$$
- ❖ Se a relação menor (depositante) ajusta completamente em memória, a estimativa de custo será
$$b_{\text{cliente}} + b_{\text{dep}} = 400 + 100 = 500 \text{ acessos de disco.}$$
- ❖ Algoritmo de laço aninhado de bloco (próximo slide) é preferível.

# *JUNÇÃO DE LAÇO ANINHADO DE BLOCOS*

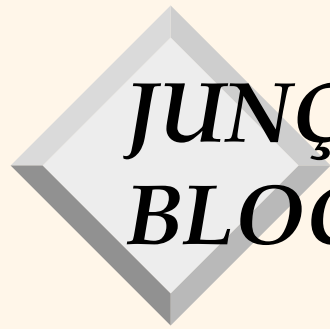
- ❖ Variante da junção de laço aninhado, em que cada bloco da relação interna é emparelhado com cada bloco da relação externa.

```
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 tupla  $t_s$  in  $B_s$  do begin
        teste o par  $(t_r, t_s)$  para ver se satisfazem a condição de
        junção
        se satisfizerem, adicione  $t_r . t_s$  ao resultado.
      end
    end
  end
end
end
```



## *JUNÇÃO DE LAÇO ANINHADO DE BLOCOS (CONT.)*

- ❖ Pior caso: o buffer pode manter apenas **um bloco** de cada relação.
- ❖ Estimativa pior caso:  $b_r * b_s + b_r$  acessos de blocos.
  - $(100*400)+100= 4.100$  acessos
- ❖ Melhor caso:  $b_r + b_s$  acessos de blocos.



## *JUNÇÃO DE LAÇO ANINHADO DE BLOCOS (CONT.)*

- ❖ Melhorias que podem ser implementadas:
  - Varrer o laço interno de maneira alternada para a frente e para trás;
  - Usar índices, na junção de laço interno, se disponível.

## JUNÇÃO DE LAÇO ANINHADO INDEXADA

- ❖ Se um índice estiver disponível no atributo de junção do laço interno, procuras por meio de índice podem substituir varreduras de arquivos.
- ❖ Para cada tupla  $tr$  na relação externa  $r$ , o índice é usado para procurar as tuplas em  $s$  que satisfaçam a condição de junção.
- ❖ No pior caso, o custo da junção é :  $br + nr * c$ ,
  - sendo que  $c$  é o custo de seleção única em  $s$  usando a condição de junção.
- ❖ Se os índices estiverem disponíveis nas duas relações, geralmente é mais eficiente usar aquela que tem menos tuplas como a relação externa.



## EXEMPLO DE JUNÇÃO DE LAÇO ANINHADO INDEXADA

- ❖ Considere uma junção de laço aninhado indexada de **depositante** ⋈ **cliente**, com **depositante** como relação externa.
- ❖ Considere também que o cliente possui um índice árvore-B+ primário no atributo de junção **nome\_cliente**, sendo a altura da árvore 4.

$$E = br + nr * c \quad c \text{ (custo da seleção em cliente)} = 4+1$$

$$E = 100 + 5.000 * 5 = 25.100 \text{ acessos de disco.}$$

–  $n_{\text{depositante}}$  é 5.000, e  $b_{\text{dep}}=100$ ,

- ❖ Esse custo é menor que os **40.100 acessos** necessários para a **junção de laço aninhado de bloco**.

# MERGE-JUNÇÃO

- ❖ O algoritmo de Merge-junção pode ser usado para calcular junções natural e equi-join.

**Junção Natural:** Conjunto das tuplas concatenando as tuplas de R e S, e eliminando as colunas duplicadas sempre que os valores correspondentes de todos os atributos em comum de R e S sejam iguais.

$$R \bowtie S$$

Exemplo:

R		
A	B	C
c	1	b
d	3	e
f	3	b
d	2	g

S		
A	B	D
f	3	e
a	2	i
d	1	n
c	1	e
f	3	b

$R \bowtie S$			
A	B	C	D
c	1	b	e
f	3	b	e
f	3	b	b

# MERGE-JUNÇÃO

- ❖ A figura abaixo mostra duas relações que estão classificadas no atributo de junção a1. Como as relações estão na ordem de classificação, as tuplas com o mesmo valor nos atributos de junção estão em ordem consecutiva.

*pr* →

<i>a1</i>	<i>a2</i>
a	3
b	1
d	8
d	13
f	7
m	5
q	6

*r*

*ps* →

<i>a1</i>	<i>a3</i>
a	A
b	G
c	L
d	N
m	B

*s*

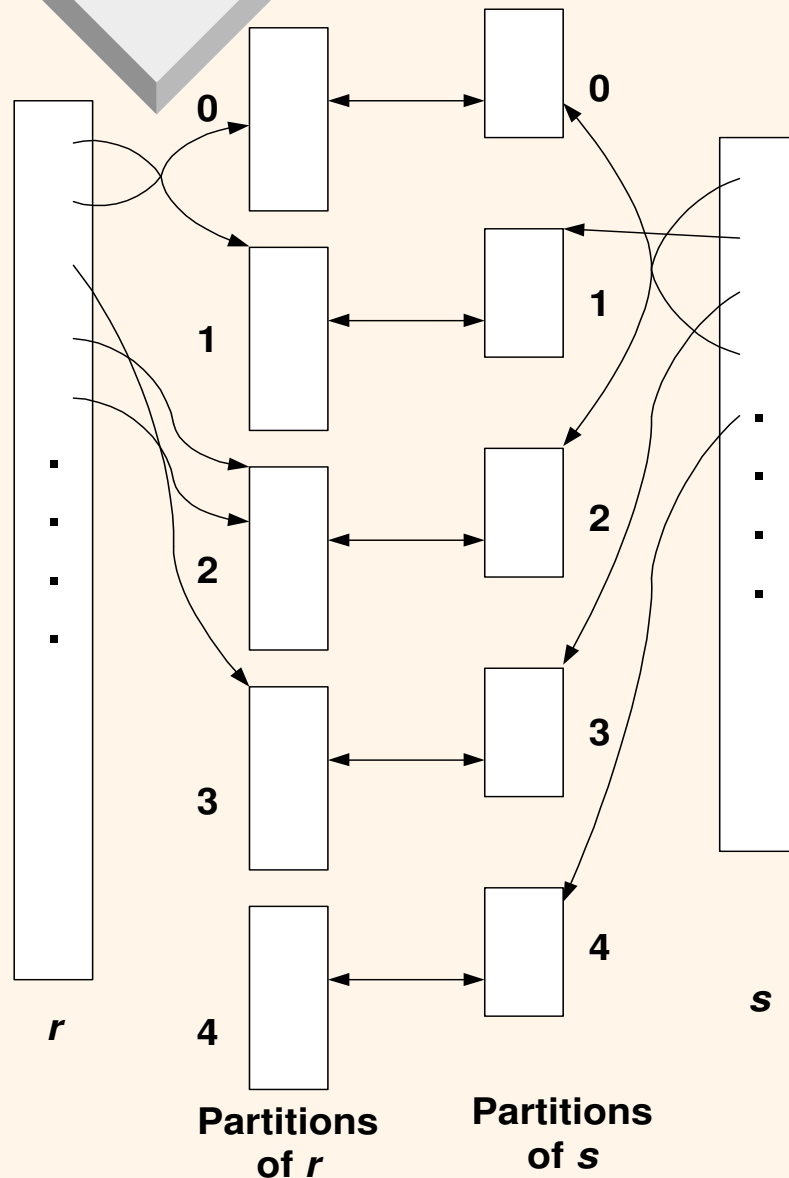
- ❖ Cada tupla na ordem de classificação precisa ser lida somente uma vez, e, como resultado, cada bloco também é lido somente uma vez.
- ❖ O n° de acessos a bloco é igual à soma do n° de blocos em ambos os arquivos, **br + bs**



## Hash-Junção

- ❖ Tal como o algoritmo merge-junção, o algoritmo **hash-junção** é usado para implementar as junções naturais e as equi-joins;
- ❖ Uma função hash  $h$  é usada para particionar as tuplas de ambas as relações em conjuntos que têm o mesmo valor hash nos atributos de junção.
- ❖ As tuplas  $r$  in  $H_{ri}$  precisam apenas ser comparadas com as tuplas de  $s$  em  $H_{si}$ ;
  - elas não precisam ser comparadas com as tuplas de  $s$  em qualquer outra partição.

## Hash-Junção (Cont.)



❖ **Particionamento recursivo** é necessário quando o número *max* de partições é **maior** que o número de páginas  $M$  da memória.



## *Custo do Hash-Junção*

- ❖ **Sem** necessidade de particionamento recursivo :

$$3(b_r + b_s) + 2 * \max$$

- ❖ Caso contrário, o custo total estimado é:

$$2(b_r + b_s) \lceil \log_{M-1}(b_s) - 1 \rceil + b_r + b_s$$

# *Junções Complexas*

## ❖ **Junção como uma condição conjuntiva:**

$$\mathbf{r} \bowtie_{\theta_1 \wedge \theta_2 \wedge \dots \wedge \theta_n} \mathbf{S}$$

- Uma ou mais técnicas de junção descritas anteriormente podem ser aplicadas às junções nas condições individuais

$$\mathbf{r} \bowtie_{\theta_1} \mathbf{S}, \quad \mathbf{r} \bowtie_{\theta_2} \mathbf{S}, \quad \mathbf{r} \bowtie_{\theta_3} \mathbf{S} \quad \text{e assim por diante.}$$

- O resultado da junção completa consiste nas tuplas do **resultado intermediário** que satisfazem as condições restantes.

$$\theta_1 \wedge \dots \wedge \theta_{i-1} \wedge \theta_{i+1} \wedge \dots \wedge \theta_n$$

## *Junções Complexas*

❖ **Junção como uma condição disjuntiva:**

$$\mathbf{r} \bowtie \theta_1 \vee \theta_2 \vee \dots \vee \theta_n \mathbf{s}$$

- Pode ser calculada como a união dos registros nas junções individuais  $\mathbf{r} \bowtie \theta_i \mathbf{s}$ :

$$(\mathbf{r} \bowtie \theta_1 \mathbf{s}) \cup (\mathbf{r} \bowtie \theta_2 \mathbf{s}) \cup \dots \cup (\mathbf{r} \bowtie \theta_n \mathbf{s})$$





## *Outras Operações*

- ❖ Podem ser implementadas outras operações relacionais e operações relacionais estendidas:
  - eliminação de duplicidade,
  - projeção,
  - operações de conjuntos,
  - junção externa e agregação.



## *Outras Operações: Eliminação de duplicidade*

- ❖ Podemos facilmente implementar a **eliminação de duplicidade** usando a classificação ou via hashing.
- ❖ Usando a **classificação**, as tuplas idênticas aparecerão adjacentes uma das outras, e todas exceto uma cópia, podem ser removidas.
- ❖ Assim, a **estimativa de custo** para eliminação de duplicidade é igual a estimativa de custo de **classificação** da relação.



## *Outras Operações: Eliminação de duplicidade*

- ❖ Eliminar duplicidade utilizando **Hashing**:
  - ❖ é similar ao algoritmo hash-junção:  
duplicações virão no mesmo bucket.
- ❖ Devido ao custo relativamente alto da eliminação de duplicidade, as linguagens de consulta comerciais exigem um pedido explícito do usuário para remover duplicatas; caso contrário, as duplicatas são mantidas.



## *Outras Operações: Projeção*

- ❖ Projeção pode ser implementada por meio da execução da projeção em cada tupla, o que resulta em uma relação que poderia ter registros **duplicados**, e então remover esses registros.
- ❖ A eliminação da duplicidade pode ser feita conforme descrito anteriormente.



## *Outras Operações: Projeção*

- ❖ Se os atributos na lista de projeção incluem uma chave da relação, nenhuma duplicata existirá.
- ❖ O tamanho de uma projeção da forma  $\Pi_A(r)$  é calculado como  $V(A,r)$ , uma vez que a projeção elimina as duplicatas.



## *Outras Operações: Operações de Conjunto*

- ❖ Podemos implementar operações de conjuntos ( $\cup$ ,  $\cap$  e  $-$ ) utilizando a classificação ou o hash-junção.
- ❖ Operações de cj. usando classificação:
  - **Primeiro ambas as relações são classificadas.**
  - $r \cup s$ : quando uma varredura concorrente em ambas as relações revela a mesma tupla em ambos os arquivos, uma única tupla é mantida.
  - $r \cap s$ : o resultado da varredura concorrente conterá somente as tuplas que aparecem em ambas relações.
  - $r - s$ : mantém as tuplas em  $r$  apenas se elas não existirem em  $s$ .



## *Outras Operações: Operações de Conjunto*

- ❖ Para todas essas operações, apenas **uma varredura** das duas relações de entrada é necessária, assim o custo é

**$br + bs$**  (fora o custo de classificação)



## *Outras Operações: Junção externa*

### ❖ Relação empréstimo

nome_agência	número_emprestimo	total
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700

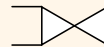
### ❖ Relação devedor

nome_cliente	numero_empréstimo
Jones	L-170
Smith	L-230
Hayes	L-155



# Exemplo de Junção Externa

nome_agência	número_emprestimo	total
Downtown	L-170	3000
Redwood	L-230	4000
Perryridge	L-260	1700



nome_cliente	numero_empréstimo
Jones	L-170
Smith	L-230
Hayes	L-155

❖ empréstimo  devedor

nome_agência	número_empréstimo	total	nome_cliente	número_empréstimo
Downtown	L-170	3000	Jones	L-170
Redwood	L-230	4000	Smith	L-230
Perryridge	L-260	1700	nulo	nulo



## *Outras Operações: Junção externa*

- ❖ **Junções externas podem ser implementadas de seguinte forma:**
  - Adicionar tuplas ao resultado da junção para obter o resultado da junção externa.
  - Pela modificação dos algoritmos de junção (merge-sort e hash-junção).



## *Outras Operações: Agregação*

❖ Relação conta agrupada pelo nome\_agência:

nome_agência	número_conta	saldo
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

❖ nome\_agência  $\Sigma$  saldo (conta)

nome_agência	soma_saldo
Perryridge	1300
Brighton	750
Redwood	700



## *Outras Operações: Agregação*

- ❖ Agregação pode ser implementada de forma semelhante à eliminação de duplicidade.
  - **Classificação ou o hash:** formar grupos com as tuplas de mesmo valor, e então a função de agregação pode ser aplicada em cada grupo.
  - Para otimizar o processo, podemos implementar as operações de agregação enquanto os grupos estão sendo construídos.



## *Avaliação de Expressão*

- ❖ Vimos como expressões relacionais individuais são realizadas.
- ❖ Agora, consideraremos como avaliar uma **expressão** que contém múltiplas operações.
- ❖ Exemplo:

Ex.:  $\Pi_{customer-name} ( \sigma_{balance < 2500} (conta) \bowtie cliente )$



## *Avaliação de Expressão*

❖ Duas abordagens:

❖ **Materialização**: Avalia uma operação por vez, o resultado de cada operação intermediária são criados (materializados) e, então, usados na avaliação das operações do próximo nível.

→ a desvantagem nessa abordagem é a necessidade de construir relações temporárias, as quais (a menos que sejam pequenas) devem ser escritas no disco.

❖ **Pipeline**: avaliam várias operações simultaneamente.

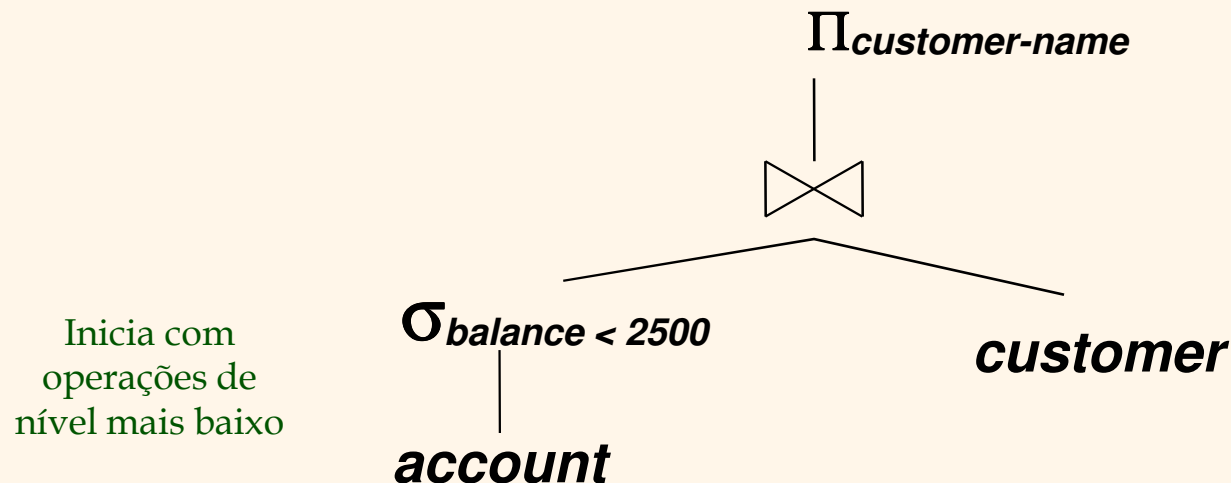


## Avaliação de Expressão: Materialização

❖ Exemplo (Materialização): Considere a expressão:

$\Pi_{customer-name} ( \sigma_{balance < 2500} (conta) \bowtie cliente )$

- Primeiro é executada a seleção  $\sigma_{saldo < 2500}(conta)$  e o resultado é armazenado;
- Então é executado e armazenado a junção com *cliente* , e
- Finalmente executado a projeção em *nome-cliente*.





## *Avaliação de Expressão: Pipeline*

- ❖ Podemos melhorar a eficiência da avaliação de consultas reduzindo o número de arquivos temporários produzidos e, conseqüentemente, é reduzido o custo de leitura e escrita.
- ❖ **Pipeline**: avaliamos muitas operações simultaneamente, passando os resultados de uma operação para a próxima.





## *Avaliação de Expressão: Pipeline*

- ❖ **Exemplo:** na expressão do exemplo anterior, ao invés de armazenar o resultado de  $\sigma_{\text{saldo} < 2500}(\text{conta})$  – passamos as tuplas imediatamente para a junção.
- ❖ Similarmente não armazenamos o resultado da junção, passamos as tuplas diretamente para a projeção.
- ❖ Assim, o pipelining possui um custo menor que a materialização (pois não precisa armazenar uma relação temporária em disco).
- ❖ Porém, o Pipelining nem sempre é possível - exemplo: na classificação e hash-join.

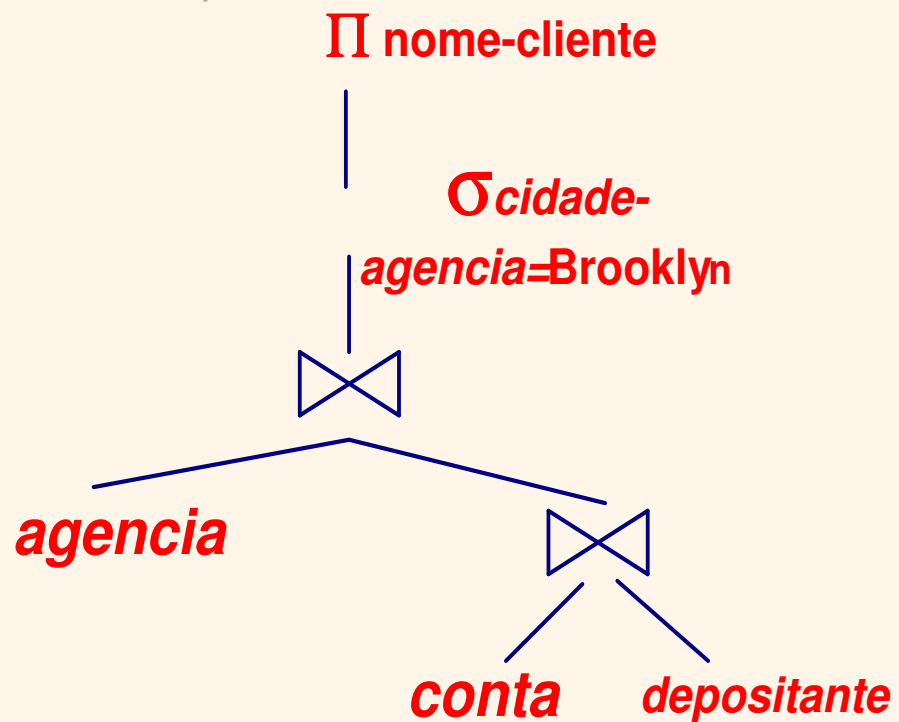
# *Transformação de Expressões Relacionais*

- ❖ Dada uma expressão da álgebra relacional, é função do otimizador de consulta propor um plano de avaliação da consulta que gere o mesmo resultado da expressão fornecida e que seja de uma maneira **menos onerosa** de gerar o resultado.
- ❖ Dois passos:
  1. Geração de expressões que são logicamente equivalentes à expressão dada;
  2. Escrever as expressões resultantes de maneiras alternativas para gerar planos de avaliação alternativos;

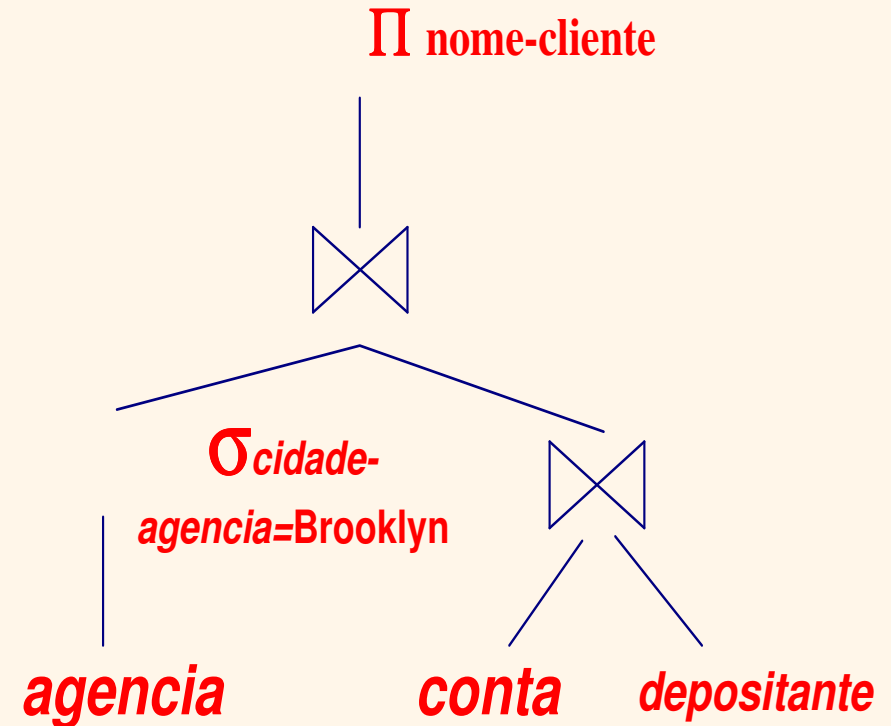
# *Equivalência de Expressões*

- ❖ Preservar a equivalência significa que as relações geradas pelas duas expressões têm o mesmo conjunto de atributos e contêm o mesmo conjunto de tuplas, embora seus atributos possam estar ordenados de forma diferente.
- ❖ O otimizador utiliza um conjunto de regras, denominadas **regras de equivalência** para transformar uma expressão em outra logicamente equivalente.

# Equivalência de Expressões *(cont.)*



(a) Árvore de Expressão Inicial



(b) Árvore de Expressao Transformada



# *Regras de Equivalência*

1. Operações de seleções conjuntivas podem ser quebradas em uma sequência de seleções individuais.

Essa transformação é chamada de **cascata  $\sigma$** .

$$\sigma_{\theta_1 \wedge \theta_2} (E) = \sigma_{\theta_1} (\sigma_{\theta_2} (E))$$



# *Regras de Equivalência*

2. Operações de Seleção são **Comutativas**

$$\sigma_{\theta_1} (\sigma_{\theta_2} (E)) = \sigma_{\theta_2} (\sigma_{\theta_1} (E))$$

3. Apenas as operações finais em uma sequência de operações de projeção são necessárias, as outras podem ser omitidas.

É chamada de **Cascata de  $\Pi$** .

$$\Pi_{L_1} (\Pi_{L_2} (...(\Pi_{L_n}(E))...)) = \Pi_{L_1} (E)$$



## *Regras de Equivalência*

4. Seleções podem ser combinadas com produtos cartesianos e junções teta.

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

Essa expressão é exatamente a definição da junção teta.

$$(b) \sigma_{\theta_1} (E_1 \bowtie_{\theta_1} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$$

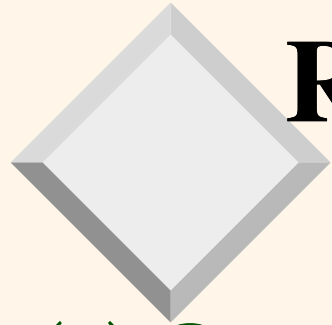


# Regras de Equivalência

5. Operações junção teta são comutativas:

$$E_1 \bowtie_{\theta} E_2 = E_2 \bowtie_{\theta} E_1$$





# Regras de Equivalência

6. (a) Operações de junção natural são associativas:

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

(b) A junção teta é associativa da seguinte maneira:

$$(E_1 \bowtie_{\theta_1} E_2) \bowtie_{\theta_2 \wedge \theta_3} E_3 = E_1 \bowtie_{\theta_1 \wedge \theta_3} (E_2 \bowtie_{\theta_2} E_3)$$

## Regras de Equivalência (Cont.)

7. A operação de seleção pode ser distribuída por meio da operação de junção teta, com as seguintes condições:

É distribuída quando todos os atributos na condição de seleção  $\theta_0$  envolvem apenas os atributos de uma das expressões (E1) que estão participando da junção.

$$\sigma_{\theta_0} (E_1 \bowtie_{\theta} E_2) = (\sigma_{\theta_0} (E_1)) \bowtie_{\theta} E_2$$

(b) É distribuída quando a condição de seleção  $\theta_1$  envolve apenas os atributos de  $E_1$  e  $\theta_2$  envolve apenas os 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 é distribuída por meio da operação de junção teta:

(a) Se envolve apenas atributos de  $L_1 \cup L_2$ :

$$\Pi_{L_1 \cup L_2} (E_1 \bowtie_{\theta} E_2) = (\Pi_{L_1} (E_1)) \bowtie_{\theta} (\Pi_{L_2} (E_2))$$

(b) Considere uma junção  $E_1 \bowtie_{\theta} E_2$ . Sejam  $L_1$  e  $L_2$  conjuntos de atributos de  $E_1$  e  $E_2$ , respectivamente. Seja  $L_3$  atributos de  $E_1$  que estão envolvidos na condição de junção  $\theta$ , mas que não estejam em  $L_1 \cup L_2$ , e seja  $L_4$  atributos de  $E_2$  que estão envolvidos na condição de junção, mas que não estão em  $L_1 \cup L_2$ . Então:

$$\Pi_{L_1 \cup L_2} (E_1 \bowtie_{\theta} E_2) = \Pi_{L_1 \cup L_2} ((\Pi_{L_1 \cup L_3} (E_1)) \bowtie_{\theta} (\Pi_{L_2 \cup L_4} (E_2)))$$

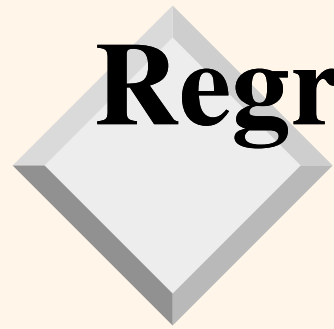
# Regras de Equivalência (Cont.)

9. As operações de conjunto união e intersecção são comutativas.

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

10. A união e a intersecção de conjuntos são associativas.

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



# Regras de Equivalência (Cont.)

11. A operação de seleção é distribuída por meio das operações de união, intersecção e diferença de conjuntos.

$$\sigma_P(E_1 - E_2) = \sigma_P(E_1) - \sigma_P(E_2)$$

12. A operação de projeção é distribuída por meio da operação de união.

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

# *Exemplo da Operação de Seleção*

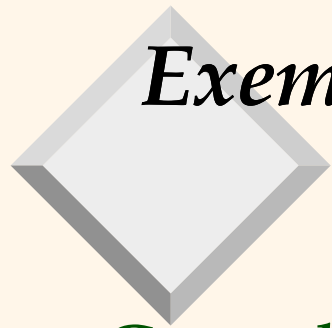
- ❖ Consulta: Encontrar os nomes de todos os clientes que tem uma conta em alguma agência localizada em Brooklyn.

$$\Pi_{\text{nome-cliente}} (\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (\text{agencia} \bowtie (\text{conta} \bowtie \text{depositante})))$$

- ❖ Transformação usando a regra 7a.

$$\Pi_{\text{nome-cliente}} ((\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (\text{agencia})) \bowtie (\text{conta} \bowtie \text{depositante}))$$

- ❖ Executando a seleção antes é possível reduzir o tamanho da relação para realizar a junção.



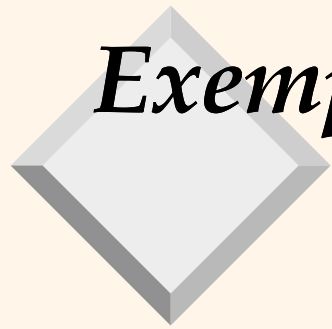
## *Exemplo da Operação de Seleção(Cont.)*

- ❖ Consulta: Encontrar os nomes de todos os clientes que tem uma conta em alguma agência localizada em Brooklyn e saldo maior que \$1000.

$\Pi_{\text{nome-cliente}} (\sigma_{\text{cidade-agencia} = \text{"Brooklyn"} \wedge \text{saldo} > 1000}$   
 $(\text{agencia} \bowtie (\text{conta} \bowtie \text{depositante}))$

- ❖ Transformação usando a associatividade (Regra 6a):

$\Pi_{\text{nome-cliente}} (\sigma_{\text{cidade-agencia} = \text{"Brooklyn"} \wedge \text{saldo} > 1000}$   
 $(\text{agencia} \bowtie \text{conta}) \bowtie \text{depositante})$



# Exemplo da Operação de Projeção

$\Pi_{\text{nome-cliente}} ((\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (agencia) \bowtie conta) \bowtie depositante)$

Com o resultado da subexpressão:

$(\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (agencia) \bowtie conta)$

Nós obtemos uma relação cujo esquema é:

(nome-agencia, cidade-agencia, fundos, numero-conta, saldo)

- ❖ Usar projeções usando as regras 8a e 8b; eliminar atributos desnecessários de resultados intermediários para obter:

$\Pi_{\text{nome-cliente}} ((\Pi_{\text{numero-conta}} ((\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (agencia \bowtie (conta \bowtie depositante))))$



## Exemplo da Operação de Junção (Cont.)

❖ Considere a expressão

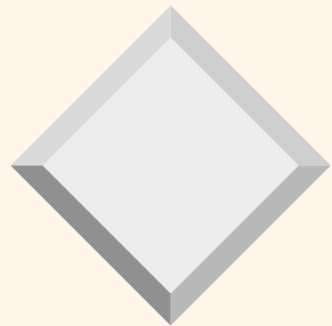
$\Pi_{\text{nome-cliente}} ((\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (\text{agencia})) \bowtie$   
 $\text{conta}) \bowtie \text{depositante})$

Poderíamos executar ~~conta~~ depositante primeiro e, então fazer a junção do resultado com:

$\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (\text{agencia})$

mas,  $\text{conta} \bowtie \text{depositante}$  é provavelmente uma relação grande, já que contém uma tupla para cada conta. Como o banco tem um grande número de agências amplamente distribuídas, é provável que apenas uma fração pequena dos clientes do banco tenha contas em agencias localizadas no Brooklyn, assim é melhor executar

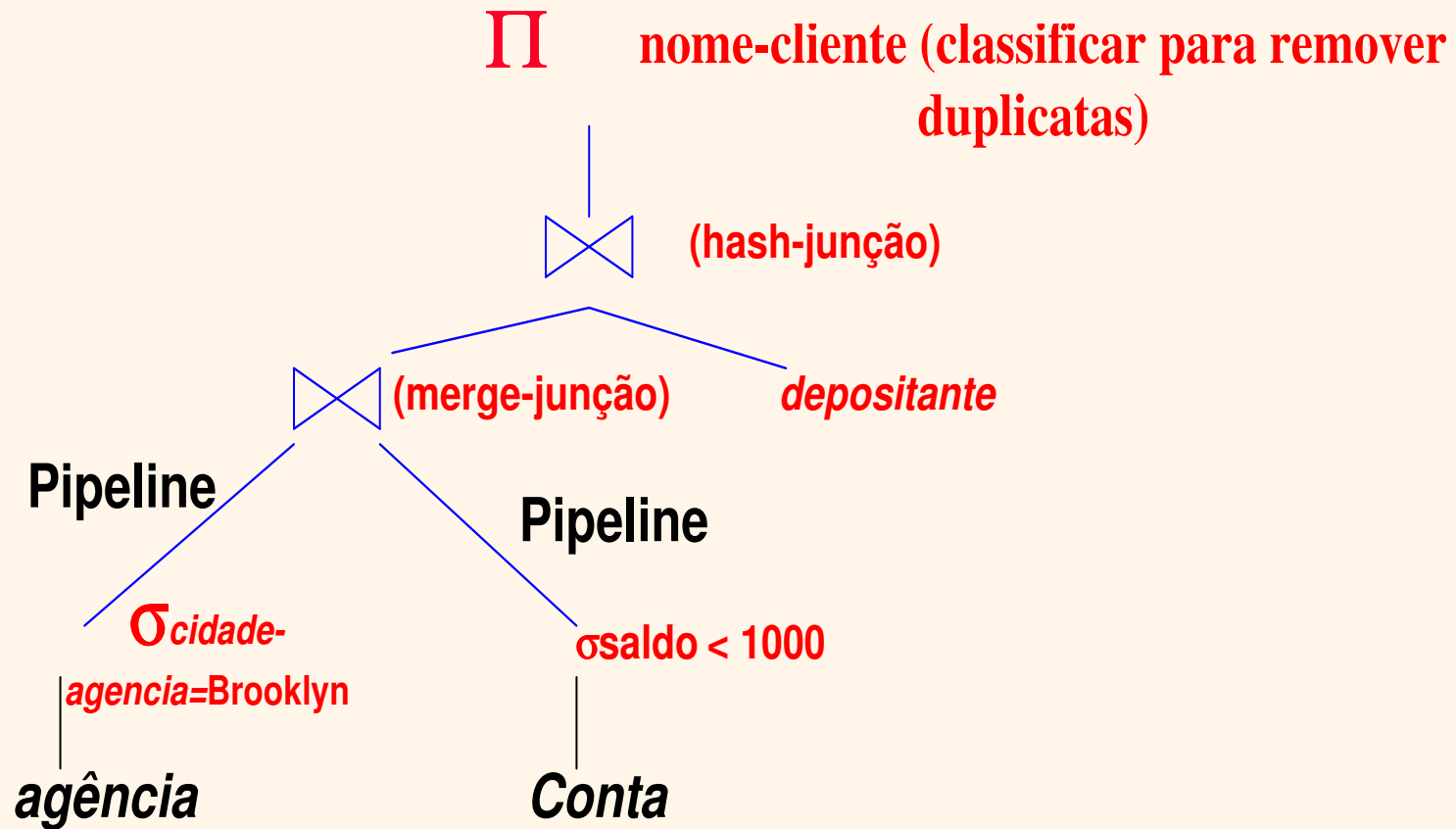
$\sigma_{\text{cidade-agencia} = \text{"Brooklyn"}} (\text{agencia}) \bowtie \text{conta}$ , primeiro



# *Planos de Avaliação*

# Planos de Avaliação

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





# *A Escolha de Planos de Avaliação*

- ❖ A escolha do algoritmo mais barato para cada operação de forma independente não necessariamente é uma boa idéia. E.g.
  - Uma função merge-junção em certo nível pode ser mais cara que uma hash-junção, contudo pode conseguir um **resultado** que torna **mais barata** a avaliação de uma operação posterior.
  - Um laço aninhado com indexação pode proporcionar oportunidades para colocar os resultados em um pipeline.



## *A Escolha de Planos de Avaliação*

- ❖ Na prática, os otimizadores de consultas incorporam elementos com duas abordagens:
  1. Procurar todos os planos e escolher o melhor com base no custo;
  2. Usar a heurística para escolher o plano.

# Otimização baseada no Custo

- ❖ Considere a expressão:

$$r_1 \bowtie r_2 \bowtie \dots r_n$$

Em que as junções estão expressas sem qualquer ordem.

- ❖ Com  $n$  relações, há  $(2(n - 1))! / (n - 1)!$  Ordens de junções diferentes.

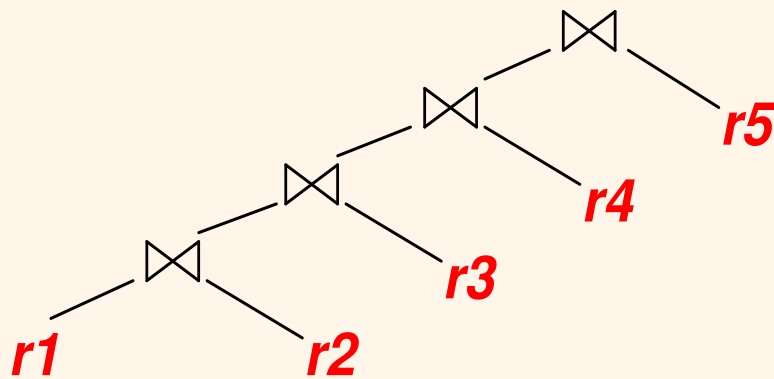
Com  $n = 7$ , o número é 665.280,

Com  $n = 10$ , o número é maior que 176 bilhões.

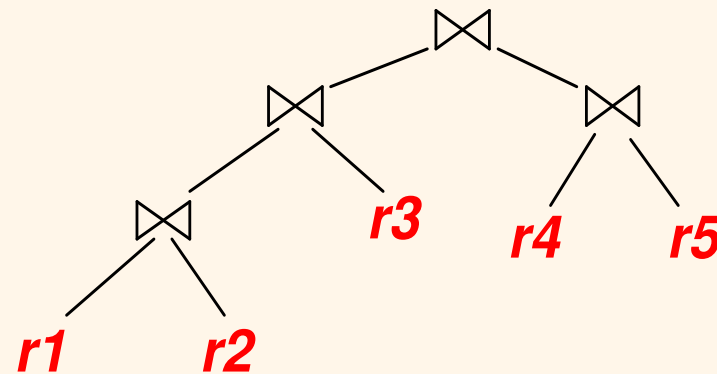
- ❖ Para um consulta complexa o número de diferentes planos de avaliação de consultas que são equivalentes pode ser grande!

## Otimização baseada em Custo (Cont.)

- ❖ Em árvores de profundidade à esquerda, são convenientes para avaliação em pipeline, já que o operando a direita é uma relação armazenada.



**(a) Left-deep Join Tree**



**(b) Non-left-deep Join Tree**

# *Ordem de Classificação Interessante Baseada no Custo de Otimização*

- ↳ Considere a expressão  $(r_1 \bowtie r_2 \bowtie r_3) \bowtie r_4 \bowtie r_5$
- ↳ Uma determinada ordem de classificação das tuplas é dita **ordem de classificação interessante** se ela puder ser útil para operação futura.
- ↳ Dado um resultado de  $r_1 \bowtie r_2 \bowtie r_3$ , sua classificação segundo atributos comuns com  $r_4$  ou  $r_5$ , poderia ser útil, mas sua classificação segundo os atributos comuns com apenas  $r_1$  e  $r_2$  não.
- ↳ Usando uma merge-junção para computar  $r_1 \bowtie r_2 \bowtie r_3$  pode ser mais cara e que a utilização de alguma outra técnica de junção, mas pode prover um resultado classificado em uma ordem interessante.



## *Ordem de Classificação Interessante baseada no Custo de Otimização* (cont.)

- ❖ Consequentemente, não é suficiente encontrar a **melhor ordem de junção** para cada subconjunto do conjunto das  $n$  relações dadas. Em vez disso, temos que encontrar a melhor ordem de junção para cada subconjunto, para cada ordem de classificação interessante do resultado da junção para aquele subconjunto.



# *Otimização Heurística*

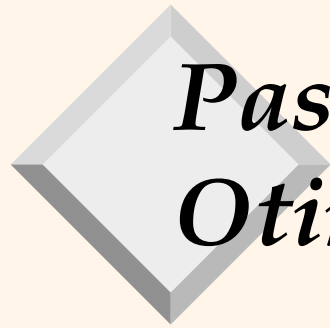
- ↪ Uma desvantagem da otimização baseada no custo é o custo da própria otimização.
- ↪ Embora o custo do processamento da consulta possa ser reduzido por meio de otimizações inteligentes, a otimização baseada em custo ainda é cara.
- ↪ Sistemas usam a heurística para **reduzir o número de alternativas** que podem ser escolhidas em uma abordagem baseada no custo.
- ↪ Alguns sistemas usam apenas heurísticas, outros combinam heurísticas com otimizações baseadas em custos parciais.



## *Otimização Heurística (cont.)*


❖ A otimização heurística transforma a árvore de consulta usando um conjunto de regras que melhoram (não em todos os casos) o desempenho. São elas:

- ↳ Execute as operações de seleção assim que possível (reduz o número de tuplas)
- ↳ Execute as projeções antes (reduz o número de atributos)
- ↳ Evite os produtos cartesianos



## *Passos em uma Típica Otimização Heurística*

1. **Separe as seleções** conjuntivas em uma seqüência de operações de seleção isoladas. (Equiv. regra 1).
2. **Mova as operações** de seleção para baixo na árvore de consulta para que sua execução ocorra o mais cedo possível. (Equiv. regras 2, 7a, 7b, 11).
3. **Determine quais operações** de seleção e de junção produzirão as menores relações. (Equiv. regra 6).



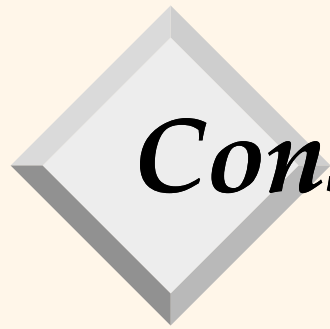
## *Passos em uma Típica Otimização Heurística*

4. **Substitua as operações** de produto cartesiano que são seguidas por uma condição de seleção por operações de junção. (Equiv. regra 4a).
5. **Separe e mova o mais para baixo** possível na árvore as listas de atributos de projeção, criando projeções novas onde forem necessárias. (Equiv. regras 3, 8a, 8b, 12).
6. **Identifique** aquelas **subárvores** cujas operações podem ser colocadas em pipeline e execute-as usando pipelining.



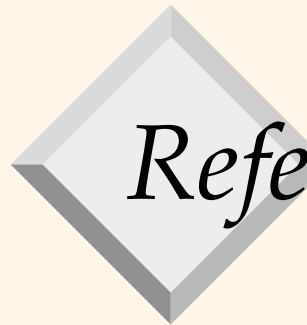
## *Passos em uma Típica Otimização Heurística*

- ❖ Em resumo, as heurísticas listadas reordenam uma representação inicial de uma árvore de consulta, de tal forma que as operações que reduzem o tamanho do resultados intermediários são aplicados primeiro.
- ❖ Executar seleções mais cedo reduz o número de tuplas e executar projeções mais cedo reduz o número de atributos.



## *Considerações Finais*

- ↪ Até mesmo com o uso da heurística, a otimização baseada no custo impõe um overhead substancial ao processo da consulta.
- ↪ Porém o custo adicional da otimização de consulta baseada em custo é normalmente mais compensador que a economia de tempo na execução da consulta, que é dominada por acessos lentos a disco.
- ↪ A diferença no tempo de execução entre um plano bom e ruim pode ser enorme, tornando a otimização da consulta essencial.



## *Referência*

- ❖ KORTH, H. F.; SILBERSCHATZ, A.; SUDARSHAN, S. Sistema de Banco de Dados, Tradução da 5ª Edição, Campus, 2006.