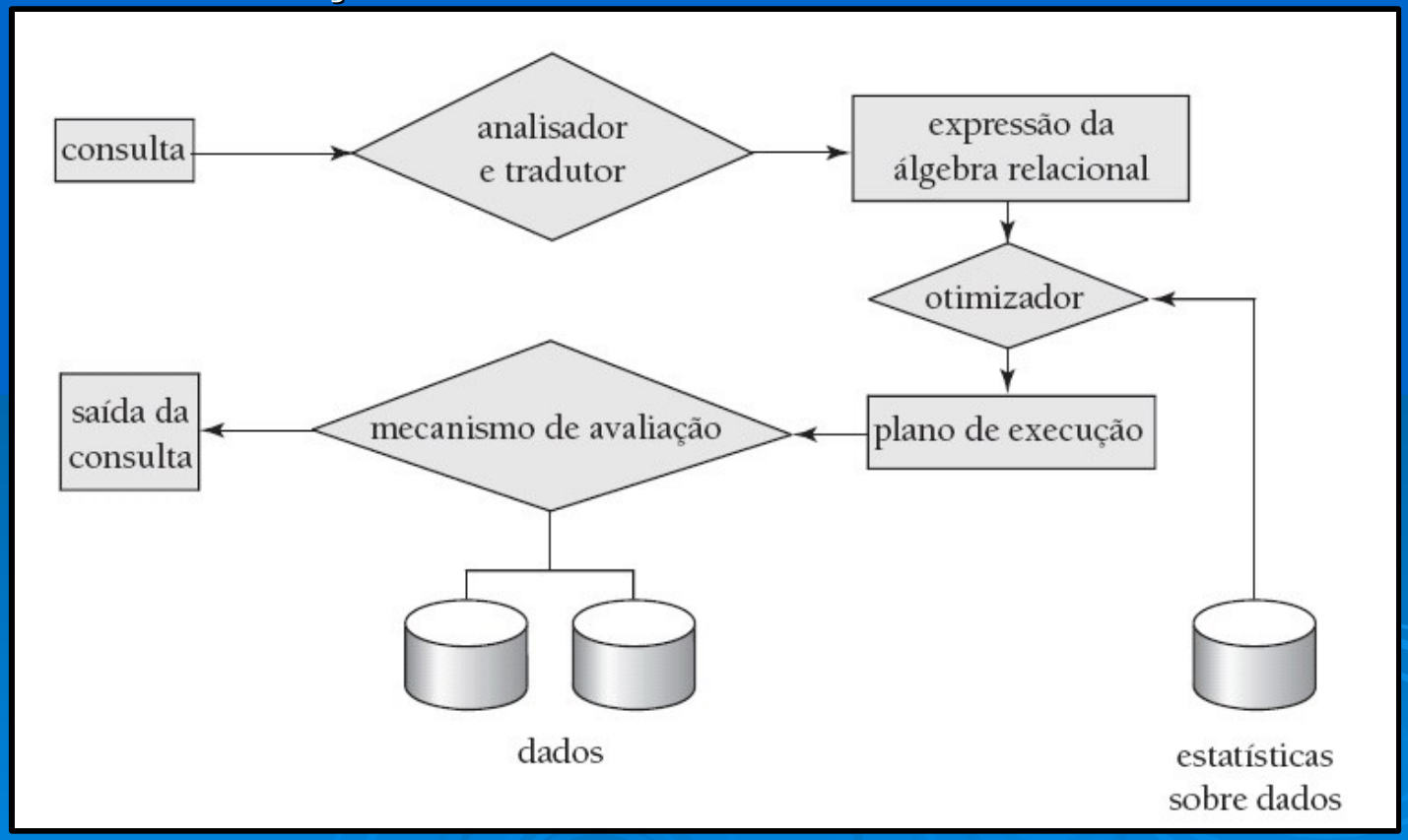
Processamento de Consultas

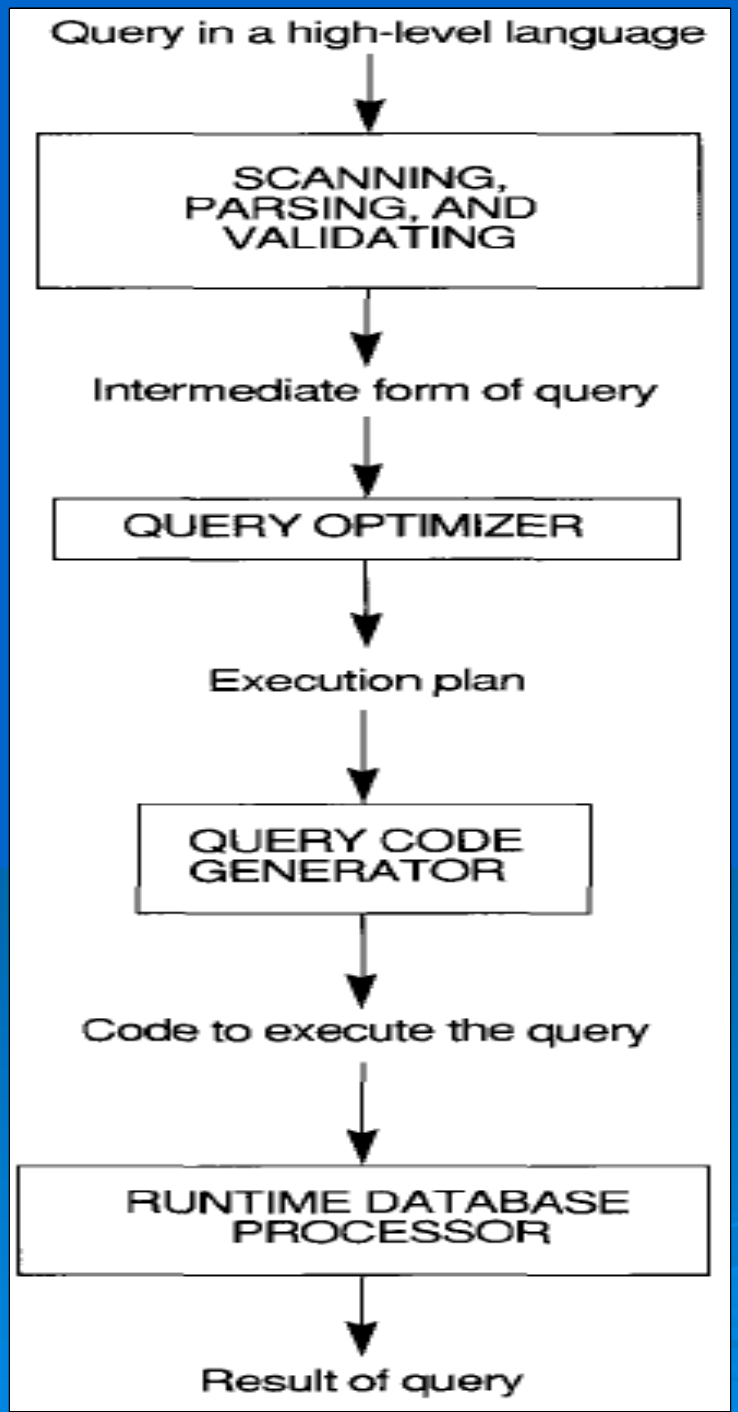
Pablo Vieira Florentino

Etapas básicas no processamento da consulta

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



Etapas básicas no processamento da consulta



Etapas básicas no processamento da consulta (cont.)

- Análise e tradução
  - traduz a consulta para o seu formato interno. Este é então traduzido para a álgebra relacional.
  - O analisador verifica a sintaxe e as relações
- Avaliação
  - O mecanismo de execução de consulta escolhe um plano de execução, executa o mesmo e retorna as respostas à consulta.

Revisão de Álgebra Relacional em 10 segundos !!!

- **Seleção σ**
- **Projeção ¶**
- **Junção ►◄**
- **União U**
- **Interseção ∩**
- **Produto Cartesiano X**
- **Diferença -**

Etapas básicas no processamento da consulta (Otimização)

- Uma expressão da álgebra relacional pode ter muitas expressões equivalentes
  - $\sigma_{saldo<2500}(\Pi_{saldo}(conta))$

    é equivalente a

    $$\Pi_{saldo}(\sigma_{saldo<2500}(conta))$$
- Cada operação da álgebra relacional pode ser avaliada usando um de vários algoritmos diferentes
  - Da mesma forma, uma expressão da álgebra relacional pode ser avaliada de muitas maneiras.

Etapas básicas no processamento da consulta (Otimização)

- Expressão anotada especificando estratégia de avaliação detalhada é chamada de plano de avaliação.
  - pode usar um índice sobre *saldo* para encontrar contas com saldo < 2500,
  - ou pode realizar varredura completa da relação e descartar contas com saldo ≥ 2500

Etapas básicas no processamento da consulta (Otimização)

- Otimização da consulta: Entre todos os planos de avaliação equivalentes, escolha aquele com o menor custo (ou utilize heurísticas de otimização de consultas).
  - O custo é estimado usando informações estatísticas do catálogo de banco de dados
  - Exemplo: número de tuplas em cada relação, tamanho das tuplas etc.
- Tentar entender como medir ALGUNS custos da consulta
  - Algoritmos para avaliação de operações da álgebra relacional
  - Como combinar algoritmos para operações individuais a fim de avaliar uma expressão completa

Medidas de custo da consulta

- Custo geralmente é medido como tempo total gasto para responder a consulta
  - Principais fatores influenciadores para o custo de tempo:
    - *acessos ao disco,*
    - *CPU*,
    - *comunicação da rede*

Medidas de custo da consulta

- Normalmente, o acesso ao disco é o custo predominante, levando-se em conta
  - Número de buscas * custo-médio-de-busca
  - Número de blocos lidos * custo-médio-de-leitura-de-bloco
  - Número de blocos escritos * custo-médio-de-escrita-de-bloco
  - Custo para escrever um bloco: *os dados são lidos de volta depois de serem escritos, para garantir que a escrita teve sucesso*

Estrutura

- Bloco: unidade de transferência de dados entre disco e memória

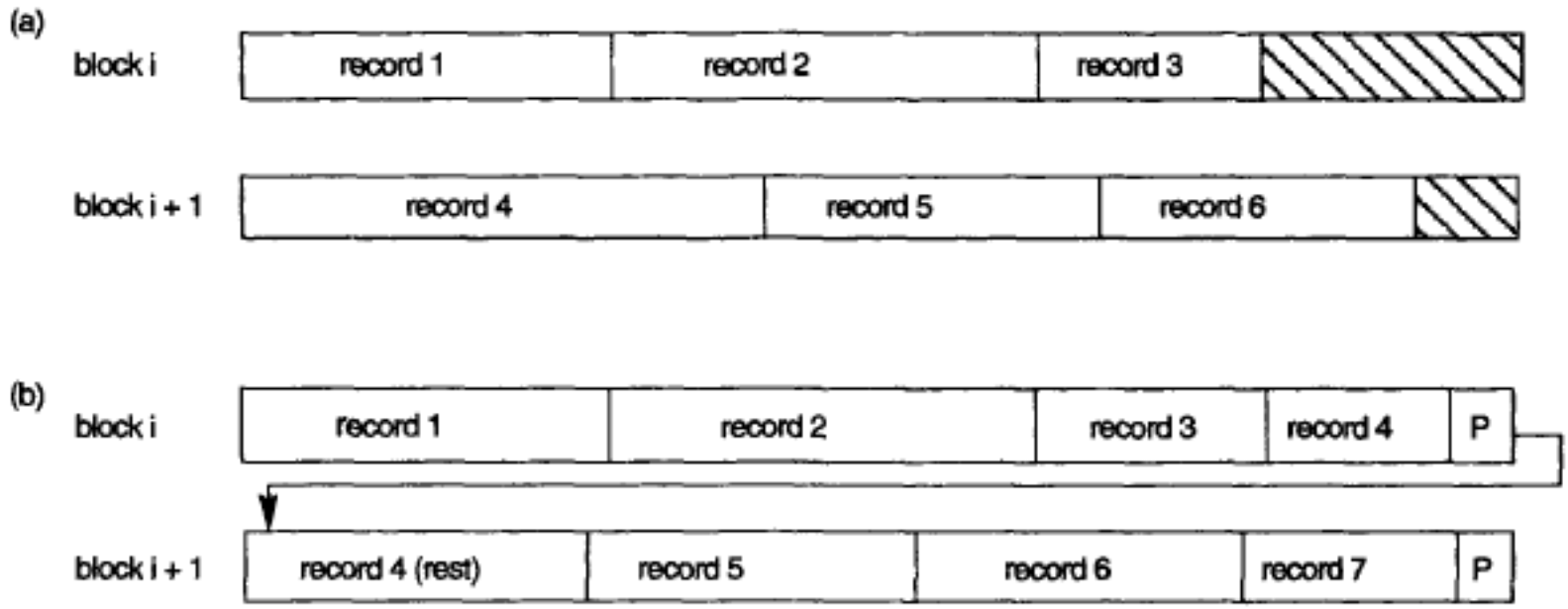


FIGURE 13.6 Types of record organization. (a) Unspanned. (b) Spanned.

Estrutura e Métricas do Católogo

- Registros de um arquivo de dados precisam ser alocados em blocos
  - Bloco > Registro → Um bloco com vários registros
    - Mais usual
    - Fator de Blocagem (bf_r) = B / R
  - onde B é o tamanho do bloco em bytes e R o tamanho do registro em bytes
  - *blocking factor* para *r* – Ex: Número de tuplas de *r que cabem em um bloco*.
    - Espaço não usado em cada bloco → B – (*bfr* * R)
    - *b = (x / fr)* → N° de blocos *b* necessários para x registros de tamanho variável (b_r = número de blocos de *r);*

Estrutura e Métricas do Católogo

- *n_r = número de tuplas em r*
- *s_r = tamanho (size) de uma tupla de r em bytes*
- *HT_i:* número de níveis em um índice *i*
  - Ex: a altura de *i.*
- Para um índice de árvore balanceada (como uma B+tree) sobre um atributo *A* de uma relação *r*,
- Para um índice hash, HT_i é 1.

Se as tuplas de r são armazenadas fisicamente juntas em um arquivo:

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

Estrutura e Métricas do Católogo

- *Fator de Seletividade*
  - *Percentual de tuplas que satisfazem um predicado*
  - *Utilizado para:*
    - *Estimar tamanho de resultados*
    - *Decidir sobre uso de índices e outras estratégias de acesso*
  - *Cálculo leva em conta estatísticas existentes*

Estrutura e Métricas do Católogo

- *Fator de Seletividade*
  - *Valor que varia entre*

    0 <= FSel() <= 1
  - *Fsel = 0 → NADA*
  - *Fsel = 1 → TUDO*

Custos - Premissas

- Para simplificar
  - usamos apenas *número de transferências de bloco do disco* como medida de custo
  - ignoramos a diferença no custo entre E/S seqüencial e aleatória
  - ignoramos os custos de CPU.
- Os custos dependerão fortemente do tamanho do buffer na memória principal
  - Ter mais memória reduz a necessidade de acesso ao disco
- Memória real disponível para o buffer depende de outros processos concorrente do SO
  - Difícil de determinar antes da execução real

Custos - Premissas

- Normalmente usam-se estimativas do pior caso, supondo que apenas a quantidade mínima de memória necessária para a operação esteja disponível
- Sistemas reais levam em conta o custo da CPU, diferenciam entre E/S seqüencial e aleatória e levam em conta o tamanho do buffer
- Nas análises apresentadas não são incluídos os custos para escrever a saída em disco
- Revisão sobre Índices

Operação de seleção (possibilidades de execução)

- 1) Varredura de Arquivo X 2) Varredura de Índice

- 1) Varredura de Arquivo
  - Algoritmo A1 (*busca linear*).
  - Leitura de cada bloco de arquivo e teste de todos os registros para verificar a condição de seleção.
  - Estimativa de custo (n° de blocos de disco varridos) = b_r, *sendo* b_r *o* número de blocos contendo registros da relação *r*
  - Se a seleção for sobre um atributo de chave, custo médio = (b_r/2)

Operação de seleção (possibilidades de execução)

- 1) Varredura de Arquivo
  - A2 *(busca binária).* Aplicável se a seleção for uma comparação de igualdade sobre o atributo em que o arquivo é ordenado.
  - Considerando os blocos de uma relação armazenados contiguamente.
  - Estimativa de custo (número de blocos de disco a serem varridos):

$\lceil \log_2(b_r) \rceil$ — custo de localização da primeira tupla por uma busca binária nos blocos

Operação de seleção

- 2) Varredura de Índice - algoritmos de busca que usam um índice (condição de seleção precisa ser sobre chave de busca do índice)
  - A3 (*índice primário sobre chave candidata, igualdade*). Recupera um único registro que satisfaz a condição de igualdade correspondente - *Custo* = HT_i + 1
  - A4 (*índice sobre chave secundária, igualdade)* Recupera vários registros - *Custo* = HT_i + N° **de blocos contendo registros recuperados**
  - A5 (igualdade sobre chave de busca do índice secundário*)*. Recupera um único registro se a chave de busca for uma chave candidata (*Custo* = HT_i + 1) ou Recupera vários registros se a chave de busca não for uma chave candidata

Custo = HT_i + *número de registros apanhados*

Operação de seleção

- 2) Varredura de Índice - algoritmos de busca que usam um índice (condição de seleção precisa ser sobre chave de busca do índice)
  - A6 (*índice primário, comparação* < , > , >= , <=) (Relação classificada sobre A, sendo $\sigma_{A \geq v}(r)$)
    - Para $\sigma_{A \geq V}(r)$, índice para encontrar primeira tupla $\geq v$ e varrer a relação sequencialmente a partir de lá
    - Para $\sigma_{A \leq V}(r)$, varredura da relação sequencialmente até a primeira tupla > *v* (não usar índice)

Operação de seleção

- 2) Varredura de Índice - algoritmos de busca que usam um índice (condição de seleção precisa ser sobre chave de busca do índice)
  - A7 (*índice secundário, comparação* < > >= <=)
    - Para $\sigma_{A \geq V}(r)$, índice para encontrar primeira entrada de índice $\geq v$, varrer o índice sequencialmente a partir de lá, para encontrar ponteiros para registros.
    - Para $\sigma_{A \leq V}(r)$, basta varrer páginas de folha de índice encontrando ponteiros para registros, até primeira entrada > *v*

Varredura de arquivo linear pode ser mais barata se muitos registros tiverem que ser apanhados!

Operação de seleção

- 2.1) Implementação de seleções complexas

  Conjunção: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$

  - A8 (*seleção conjuntiva usando um índice).*
    - *combinação de θ_i e algoritmos de A1 a A7 com menor custo para $\sigma_{\theta i}$ (r)*
  - *A9 (seleção conjuntiva usando índice de chaves múltiplas).*
    - *índice composto apropriado (chave múltipla) se estiver disponível.*

Operação de seleção

- 2.1) Implementação de seleções complexas

  Disjunção: $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.

  - A10 (*seleção disjuntiva pela união de identificadores*).

  - Negação: $\sigma_{\neg\theta}(r)$
  - Use varredura linear no arquivo
  - Se muito poucos registros satisfizerem $\neg\theta$, e um índice for aplicável a θ, utilizar índice

Classificação

- Índice sobre a relação, e depois usar o índice para ler a relação na ordem classificada. Pode levar a um acesso ao bloco de disco para cada tupla.

- Para relações que cabem na memória, opção de técnicas como quick-sort.
- Para relações que não cabem na memória, o sort-merge externo é uma boa opção.
  - número total de acessos ao disco para a classificação externa: $b_r (2\lceil \log_{M-1}(b_r / M) \rceil + 1)$, onde b_r *é o n° de blocos para r e M o n° de blocos de r que cabem na memória*
  - Grande dependência do tamanho do buffer

Operação de Junção

- Vários algoritmos diferentes para implementar junções
  - Junção de loop aninhado
  - Junção de loop aninhado indexado
  - Junção merge
  - Junção de hash
- Escolha baseada em estimativa de custo
  - Notação:
    - b_r = número de blocos de *r;*
    - n_r = número de tuplas em *r*
    - s_r = *tamanho de uma tupla de r em bytes.*
    - f_r = *blocking factor* para *r* – Ex: Número de tuplas de *r que cabem em um bloco*.

Operação de Junção

- Exemplos usam a seguinte informação, numa junção R ►◄ S, onde R é a relação externa:
  - Cliente ►◄ Depositante
  - Número de registros de *cliente*: 10.000 *depositante*: 5000
  - Número de blocos de *cliente*: 400 *depositante*: 100
  - Ou seja
    - $b_c = 400$ $n_c = 10000$
    - $b_d = 100$ $n_d = 5000$

Junção de loop aninhado

- Contexto A) Memória suficiente somente para manter um bloco de cada relação:
  o custo estimado é $n_r * b_s + b_r$ acessos ao disco.
  - _____ * ___ + ___ = _____ acessos ao disco
    com *depositante* como relação externa
  - _____ * ___ + ___ = _____ acessos ao disco
    com *cliente* como relação externa.

Junção de loop aninhado

- Contexto B) Relação menor cabe inteiramente na memória:
  - Relação menor como relação interna.
  - custo passa para $b_r + b_s$ acessos ao disco.
  - Se a relação menor (*depositante*) couber inteiramente na memória, a estimativa de custo será

    $b_c + b_d$ = ___ + ___ = ___ acessos ao disco

Junção de loop aninhado em bloco

- Estimativa no pior caso: $b_r * b_s + b_r$ acessos ao bloco.
  - Cada bloco na relação interna *s* é lido uma vez para cada *bloco* na relação externa (em vez de uma vez para cada tupla na relação externa)

    $b_c * b_d + b_c$
- Melhor caso: $b_r + b_s$ acessos ao disco.

Junção de loop aninhado indexado

- Pior caso: buffer tem espaço para somente uma página de *r* e, para cada tupla em *r*, realizamos uma pesquisa de índice sobre *s*.
  - Custo da junção: $b_r + n_r * c$

  onde *c* é o custo de atravessar o índice e recuperar todas as tuplas *s* que combinam para uma tupla em *r*
  - *c* pode ser estimado como custo de uma única seleção sobre a outra relação participante da junção (no caso, *s*) usando a condição de junção.

Exemplo de sala: custos de junção

- Ex.: Calcule *depositante* ►◄*cliente,* com *depositante* como relação externa.

Considere que *cliente* tem um índice de chave primária B^+ sobre o atributo de junção *nome-cliente*, que contém 20 entradas em cada nó de índice.

Como *cliente* tem 10.000 tuplas, a altura da árvore é 4, e mais um acesso é necessário para encontrar os dados reais → 5

- Loops aninhados em bloco

$b_c * b_d + b_c$ = ___ * ___ + ___ = acessos ao disco

supondo a memória de pior caso

- Loops aninhados indexados

$b_r + n_r * c$ = ___ + ___ * __ = acessos ao disco.

Junção Merge

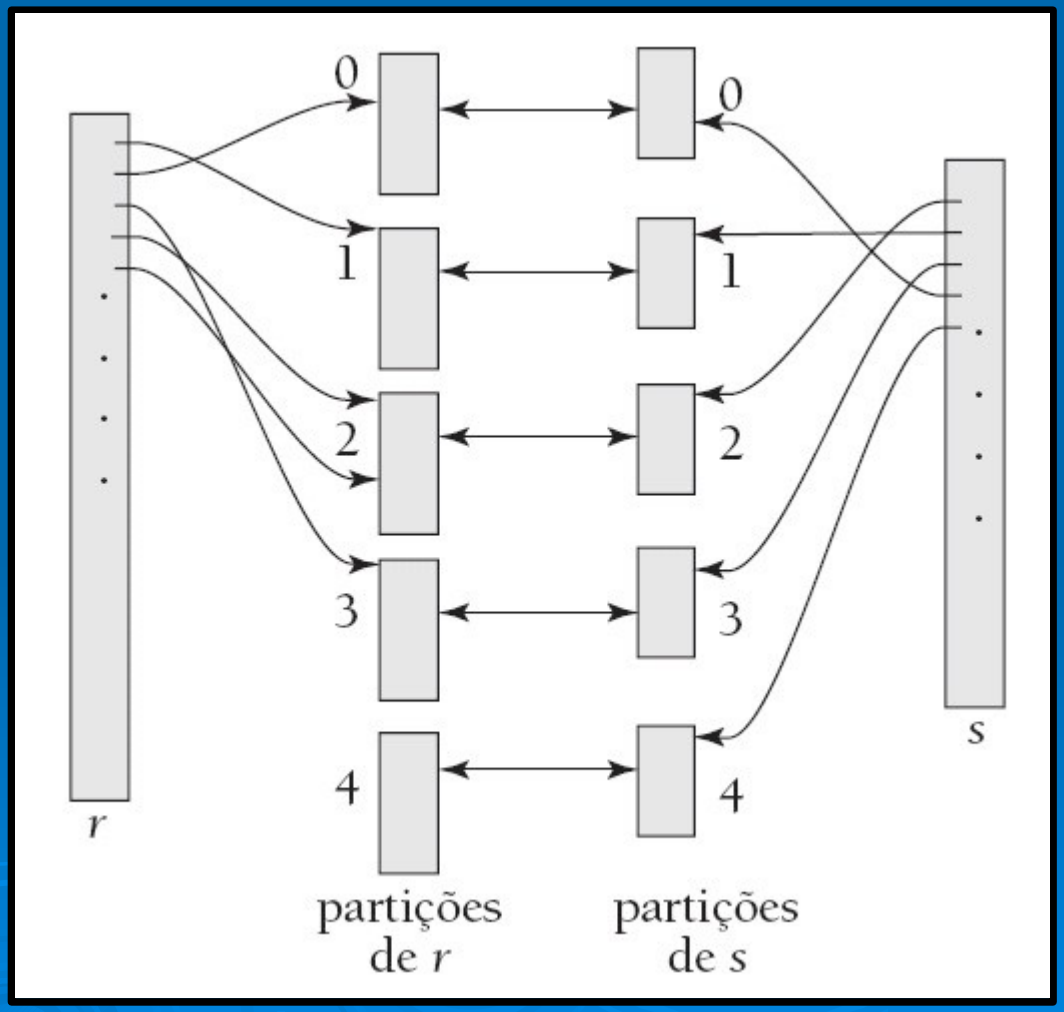
- Passos:
  - Classifique as duas relações por seu atributo de junção (se já não estiverem classificadas pelos atributos de junção).
  - Mescle as relações classificadas para juntá-las
    - A etapa de junção é semelhante ao estágio de mesclagem do algoritmo de sort-merge.
    - Cada bloco precisa ser lido uma vez (supondo que todas as tuplas para determinado valor dos atributos de junção caibam na memória)

Assim, o número de acessos a bloco para a junção merge é

$b_r + b_s$ + o custo da classificação se as relações não estiverem classificadas.

Junção baseada em Hash

- As tuplas da relação *r* em r_i só precisam ser comparadas com as tuplas da relação *s* em s_i. Elas não precisam ser comparadas com as tuplas *s* em qualquer outra partição.



Junção baseada em Hash

1. Particionamento da relação s usando a função de hashing *h*. Um bloco de memória é reservado como buffer de saída para cada partição.
2. Particionamento *r* de modo semelhante.
3. Para cada *i:*

   (a) Uma partição s_i na memória com um índice de hash na memória sobre ele usando o atributo de junção. Esse índice de hash usa uma função de hash diferente da anterior *h*.

   (b) Leitura das tuplas em r_i a partir do disco uma por uma.
   Para cada tupla t_r é localizada cada tupla correspondente t_s em s_i usando o índice de hash na memória.
   Concatenação de seus atributos.

Junção baseada em Hash

- Particionamento simples

  $3(b_r + b_s) + 2 * n_h$

- Particionamento recursivo

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

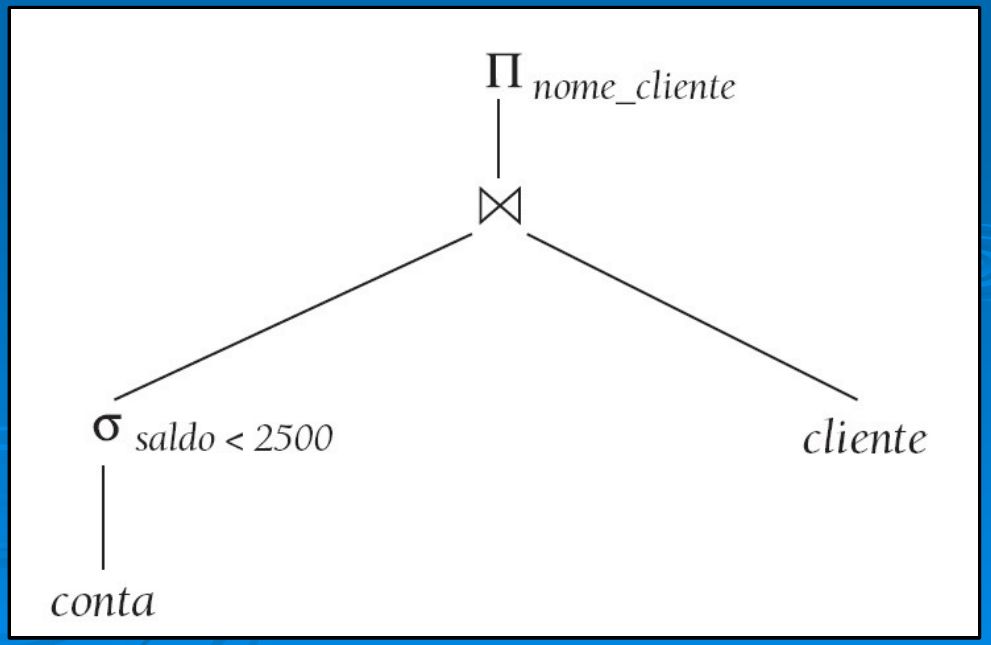
Melhor Caso (sem particionamento, índice hash em memória)

$b_r + b_s$

Avaliação de Consultas

- Exemplo: Select nome from cliente c join conta cc on c.id = cc.cliente_id where saldo < 2500

$$\Pi_{\text{nome cliente}} (\text{cliente} \blacktriangleright\!\blacktriangleleft (\sigma_{\text{saldo} < 2500} (\text{conta})))$$



Avaliação e Otimização

- Abordagem baseada em estimativa do custo das operações
  - Grande dependência de informações estatísticas sobre relações que o banco de dados precisa manter
    - número de tuplas, número de valores distintos para atributos de junção, etc.
  - Custo de expressões complexas: estatísticas para resultados intermediários
  - Equivalência de expressões da Álgebra Relacional

Avaliação e Otimização

- Etapas da Geração de planos de avaliação de uma consulta:
  - Geração de expressões logicamente equivalentes usando *regras de equivalência* para transformar uma expressão em uma equivalente.
  - Registro de expressões resultantes para obter planos de consulta alternativos
  - Escolha do plano mais barato com base no *custo estimado*

Avaliação e Otimização

- Duas expressões da álgebra relacional são consideradas equivalentes se em cada instância de banco de dados válida as duas expressões gerarem o mesmo conjunto de tuplas
  - A ordem das tuplas é irrelevante
  - Ponto base para aplicação de regras de equivalência

Avaliação e Otimização

- Alguns exemplos de regras de equivalência
  - Operações de seleção conjuntiva podem ser desmembradas em uma seqüência de seleções individuais.

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

  - Operações de seleção são comutativas.

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

Regras de equivalência (cont.)

- A operação de seleção distribui pela operação de junção teta sob as duas condições a seguir:

(a) Quando todos os atributos em θ_0 envolvem apenas os atributos de uma das expressões (E_1) sendo juntadas.

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

(b) Quando θ_1 envolve apenas os atributos de E_1 e θ_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.)