Processamento de Consultas

Profa. Dra. Sarajane Marques Peres
Escola de Artes, Ciências e Humanidades / USP
www.each.uspnet.br/sarajane

Introdução

- Processar consultas envolve:
  - Traduzi-las de uma linguagem de alto nível (como SQL) para expressões que podem ser implementadas no nível físico do sistema de banco de dados;
  - Otimizar a expressão destas consultas;
  - Avaliá-las (executá-las) e fornecer o resultado.

- Toda consulta tem um custo para o sistema. É interessante avaliar este custo de acordo com o número de acessos a disco necessários para executar a consulta.
  - O acesso a disco é lento quando comparado ao acesso em memória;
  - Normalmente o processamento em memória exigido para uma consulta é pequeno o suficiente para não influenciar fortemente o custo.

- Existem muitas estratégias para processar uma (mesma) consulta.
- Vale a pena o sistema gastar uma quantia de tempo para selecionar uma estratégia eficiente.
  - Abordagem heurística
  - Abordagem sistemática

O processador de consultas de um SGBD clássico

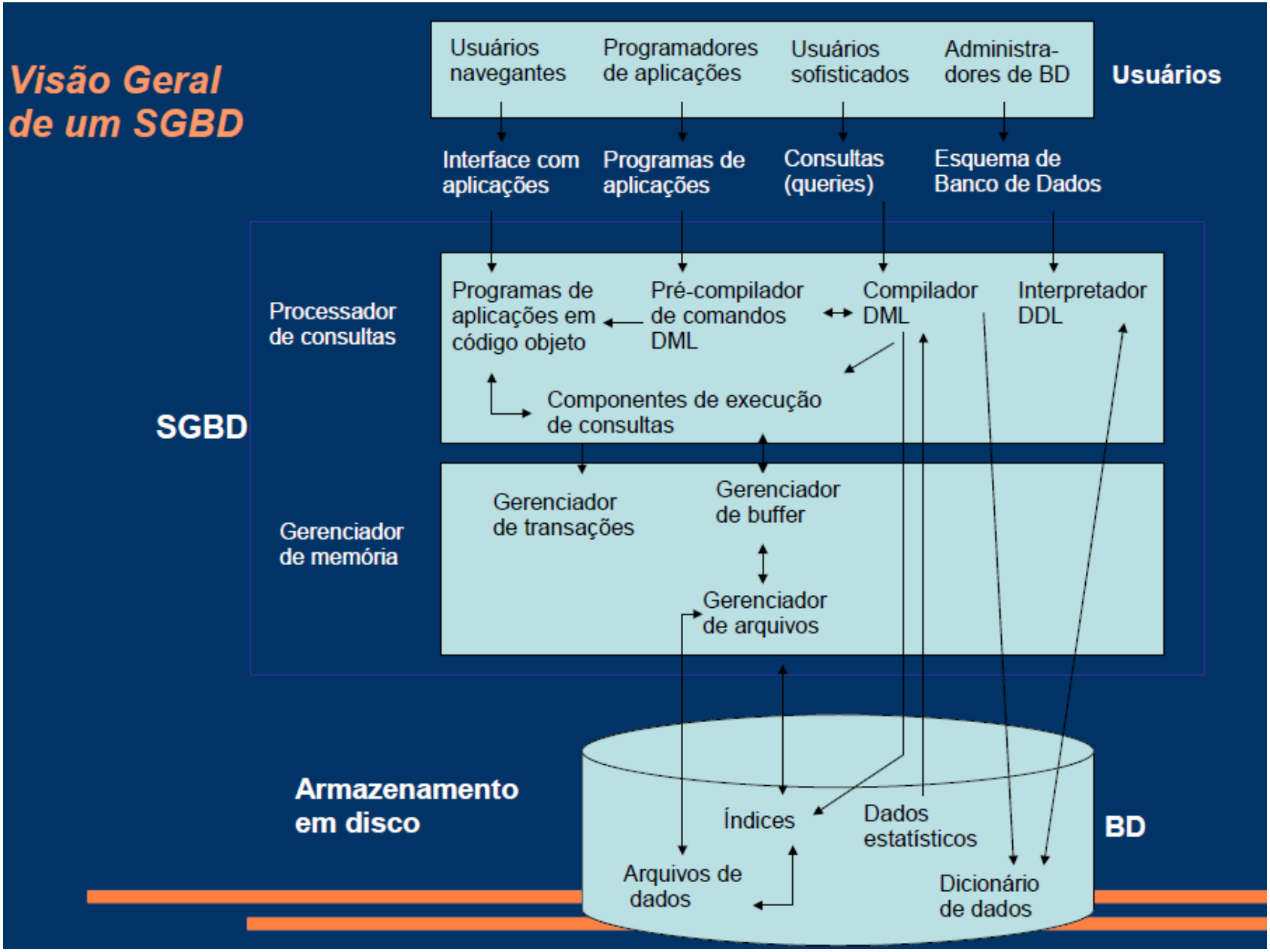
Linguagens de consulta de um SGBD

- Linguagem de Manipulação de Dados
  - Manipulação de dados = recuperação de informações **do banco** de dados e inserção, remoção e alteração de dados **no banco** de dados.
- A linguagem de manipulação de dados (do inglês *data manipulation language – DML)* é a linguagem que viabiliza o acesso e a manipulação dos dados.
- Pode ser:
  - Procedurais (record–a-time): o usuário especifica procedimentos para recuperar os dados que necessita.
  - Nao-procedurais (set-a-time): o usuário descreve os dados que necessita. (declarativas)

Linguagens de consulta de um SGBD

- Linguagem de Definição de Dados
  - Do inglês: *Data Definition Language - DDL.*
  - Definição de dados: viabiliza a criação dos esquemas de bancos de dados, ou seja, a implementação do banco de dados.

Visão Geral de um SGBD



Processador de Consultas

- **Compilador DML**: traduz comandos DML em instruções de baixo nível, entendidas pelo componente de execução de consultas. Além disso, otimiza a solicitação do usuário.

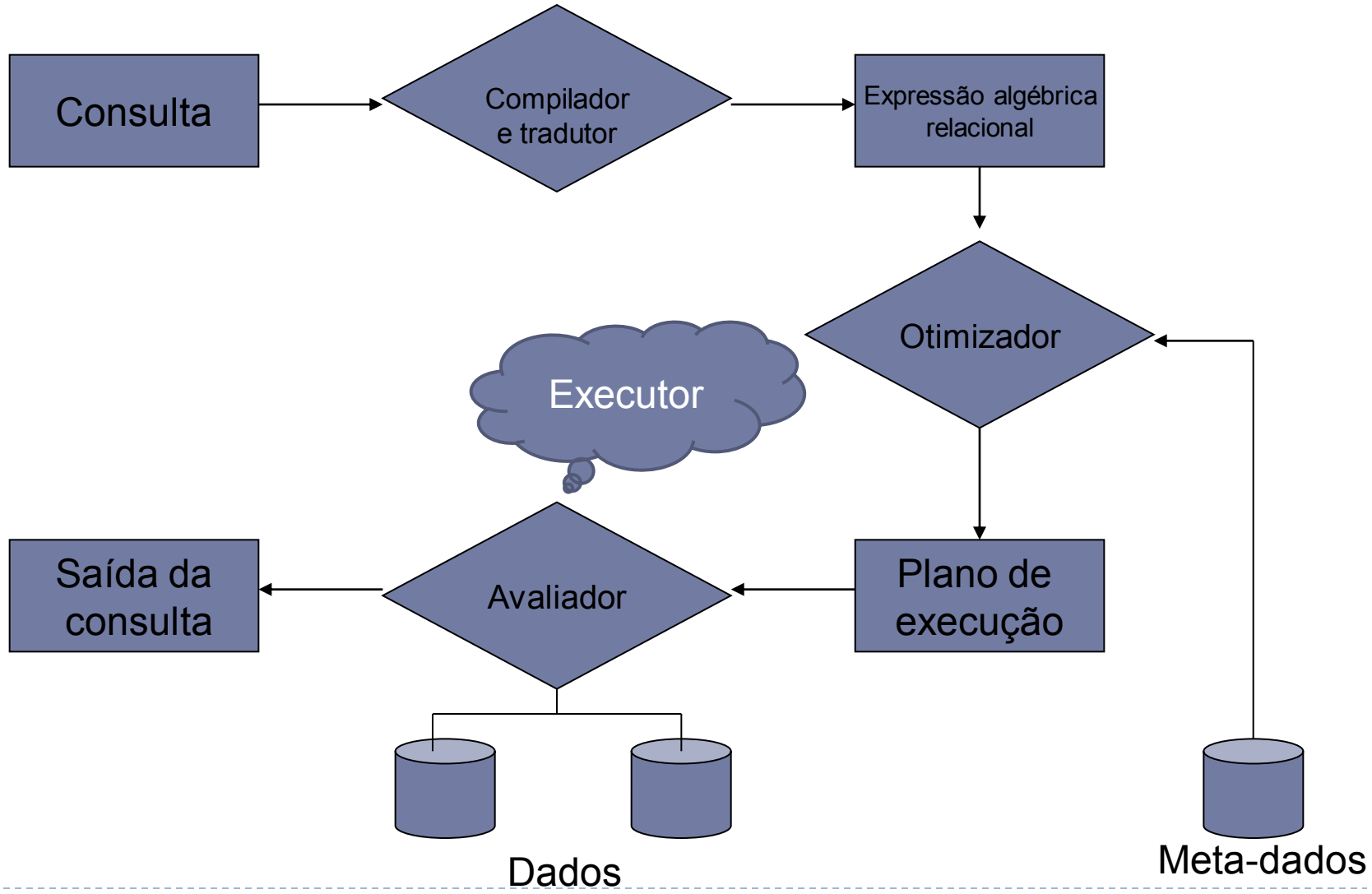
- **Pré-compilador para comandos DML**: converte comandos DML em chamadas de procedimentos normais da linguagem hospedeira. Interage com o compilador DML de modo a gerar o código apropriado.

- **Interpretador DDL**: interpreta os comandos DDL e os registra no dicionário de dados.

- **Componentes para tratamento de consultas**: executa instruções de baixo nível gerada pelo compilador DML.

O processamento de uma consulta

Introdução


Consulta
Compilador e tradutor
Expressão algébrica relacional
Otimizador
Executor
Saída da consulta
Avaliador
Plano de execução
Dados
Meta-dados

Introdução

- O compilador:
  - Análise léxica
  - Análise sintática
- O tradutor:
  - Tradução para uma expressão da álgebra relacional
  - Criação de uma estrutura de dados: uma árvore de consulta
- O otimizador (ou planejador):
  - Criação de planos de consulta equivalentes (diversas árvores de consulta)
  - Determinação do plano de consulta final por meio dos cálculos de custo
- O avaliador:
  - Quem de fato executa a consulta

Considerações

- Por que havia dificuldade de realizar otimizações automáticas para o modelo de rede e para o modelo hierárquico?
- ... Linguagens de consultas relacionais são declarativas ou algébricas:
  - Ser declarativa permite a especificação do que a consulta deve gerar e não como deve executar – permite mudar o plano de ação;
  - Ser algébrica permite realizar transformações criando planos de execução equivalentes.
- Exemplo ...
- Algumas tecnologias não usam a álgebra relacional, usam uma estrutura de árvore sintática que é construída a partir da consulta em SQL.

Um exemplo de consulta no modelo de rede

- Encontre todos os clientes da agencia Perryridge:

```
branch.branch-name := "Perryridge";
find any branch using branch-name;
find first account within account-branch;
while DB-status = 0 do
      begin
            find owner within depositor;
            get customer;
            print (customer.customer-name);
            find next account within account-branch;
      end
```

Exemplo de consulta no modelo relacional

select saldo
from conta
where saldo < 2500;

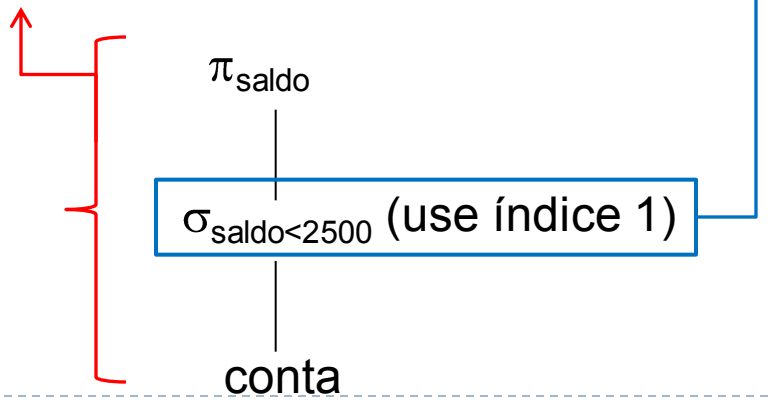
- Esta consulta pode ser traduzida nas duas expressões algébricas relacionais diferentes:

$$\sigma_{saldo<2500}(\pi_{saldo}(conta))$$
$$\pi_{saldo}(\sigma_{saldo<2500}(conta))$$

- Além desta variação, é possível executar cada operação algébrica relacional usando um entre diversos algoritmos diferentes. Por exemplo:
  - Para executar a seleção, podemos procurar em todas as tuplas de *conta* a fim de encontrar as tuplas com saldo menor que 2.500.
  - Se um índice árvore-B^+ estiver disponível no atributo *saldo*, podemos usar o índice em vez de localizar as tuplas.

...

- É necessário prover as expressões algébricas com anotações que permitam especificar como elas serão avaliadas.
- Uma operação algébrica relacional anotada com instruções sobre como ser avaliada é chamada de *avaliação primitiva*.
- Vária avaliações primitivas podem ser agrupadas em *pipeline*, e executadas em paralelo.
- Uma seqüência de operações primitivas é um plano de execução de consulta ou plano de avaliação de consulta.



Considerações

- Não se espera que o usuário escreva suas consultas de uma maneira que sugira o plano de avaliação mais eficiente. É responsabilidade do sistema construir um plano de avaliação de consulta que minimize seu custo.

- O cálculo do custo de cada plano e feito pelo otimizador usando informações estatísticas sobre o tamanho das relações e a profundidade dos índices. O otimizador estima o custo dos diferentes planos de avaliação.

- Se existe um índice disponível no atributo *saldo de conta,* então o plano *de* avaliação correspondente, no qual a seleção é feita usando o índice, tem chances de apresentar um custo menor.

- Uma vez escolhido o plano de consulta, a consulta e avaliada (executada) com aquele plano e o resultado da consulta e produzido.

Outro exemplo

Consulta composta por dois blocos:

π unome, pnome (σ salario> Constante (empregado)

Select unome, pnome
From empregado
Where salario > (**select** max (salario)
from empregado
where dno = 5);

= Constante

$\Im$ max(salário) (σ DNO = 5 (empregado))

O bloco mais interno é executado apenas uma vez e a resposta é uma constante.

Existem casos (consultas diferentes porém também aninhadas) em que o bloco interno é executado mais de uma vez.

Calculando custos (informativo)

Slides de outra aula!

Classificação

- Importância da classificação de dados:
  - Pode ser necessário mostrar o resultado de uma consulta de forma ordenada;
  - Diversas operações relacionais podem ser implementadas de forma mais eficaz se as relações envolvidas estiverem classificadas. (order by, select, join, union, intersect)
- Formas de ordenação:
  - Lógica: construção de um índice na chave de classificação, o qual será usado para **ler** a relação na ordem de classificação.
    - A leitura de tuplas na ordem de classificação pode conduzir a um acesso de disco para cada tupla.
  - Física: as tuplas são gravadas de forma ordenada no disco.

Classificação

- O problema de classificação pode ser tratado sob duas condições:
  - Quando a relação cabe completamente na memória principal e técnicas padrões de classificação (**quick-sort** entre outras) podem ser usadas.
  - Quando a relação é maior que a memória principal → *classificação externa*;
    - Algoritmo comum: **sort-merge externo**

Ordenação externa (sort-merge-externo)

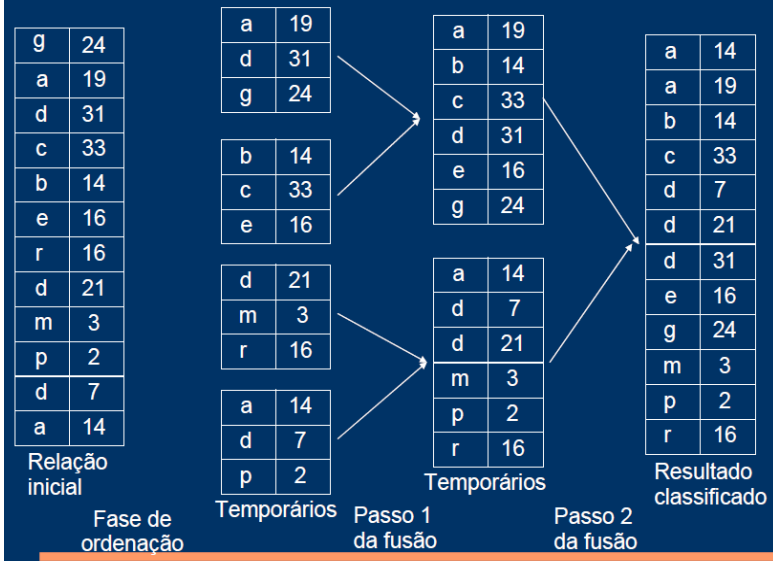
- Inicialização

  i = 1;
  j = b; // b é o tamanho do arquivo em blocos
  k = n_b; // n_b e o tamanho do *buffer* em blocos
  m = ⌈ (j/k) ⌉ // (função maior inteiro) m é o número de iterações da fase de ordenação

  Se o arquivo está armazenado em 10 blocos (j =10) e se cabem 5 blocos no buffer (k = 5) então m = 2 e significa que para colocar o arquivo no *buffer* serão necessários duas iterações. Na primeira entram 5 blocos, e na segunda (com o *buffer* esvaziado) entram mais 5 blocos.

Exemplo

j = 12 k = 3 m = 4 p = 2 j = 4 q = 2



- Fase de ordenação

Enquanto (i <= m)
faça
{
leia os próximo **k** blocos do arquivo para o buffer
ou se houver menos do que **k** blocos restantes,
leia os blocos restantes;

ordene os registos no buffer e grave-os como um
subarquivo temporário;

i = i + 1;
}

- o número de temporários criados é = m.

- Fase de fusão

K é o número de blocos que cabem no buffer!

Inicialize
i = 1; j = m;
p = $\lceil \log_{k-1} m \rceil$; \\ p é o número de passagens da fusão

enquanto (i <= p) **faça** {
n = 1;
q = $\lceil (j / (k-1)) \rceil$;\\ número de subarquivos para processar nesta passagem
enquanto (n <= q) **faça** {
leia os próximos k-1 subarquivos ou aqueles restantes,
um bloco por vez;
funda e grave como novo subarquivo, um bloco por
vez;
n = n + 1; }
j = q; i = i + 1; }

Cálculo do custo

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

Na primeira fase, todo bloco da relação é lido e escrito novamente, dando um total de **2b** acessos a disco. O número inicial de temporários é **m**.

Como o número de temporários diminui por um fator de k-1 em cada passo de merge, o número total de passos de merge necessários é dado por **$[\log_{k-1}(m)]$**.

Assim o número total de acessos ao disco para a classificação externa da relação é:

$$b (2 [\log_{k-1} (m)] + 2)$$

Operação de seleção

- É a varredura de arquivos: o operador de mais baixo nível para se ter acesso aos dados.
- São algoritmos de procura que localizam e recuperam os registros que estão de acordo com uma condição de seleção.
- Tem-se vários algoritmos que variam de acordo com a complexidade da seleção e o uso ou não de índices. Veremos dois deles - os básicos – com mais detalhes:
  - aquele que envolve uma busca linear: recupera cada registro do arquivo e testa se seus valores de atributos satisfazem a condição de seleção.
  - aquele que envolve uma busca binária: se a condição de seleção envolver uma comparação de igualdade em um atributo-chave para o qual o arquivo está ordenado, pode-se usar a busca binária – que é mais eficiente que a busca linear.
- Considere uma operação de seleção em uma relação cujas tuplas são armazenadas juntas em um único arquivo ...

Algoritmo 1 (busca linear) – S1

- Em uma busca linear, cada bloco de arquivo é varrido e todos os registros são testados para verificar se satisfazem a condição de seleção.
- Como todos os blocos precisam ser lidos, a estimativa de custo é $E_{s1} = b$, onde **b** é o número de blocos do arquivo.
- No caso da seleção ser aplicada em um atributo-chave, podemos **supor** que a metade dos blocos é varrida antes de o registro ser encontrado, ponto no qual a varredura termina. A estimativa de custo então será $E_{S1} = (b/2)$. No pior caso, $E_{S1} = b$.

Algumas considerações

- f_r: é o fator de bloco da relação r, ou seja, o número de tuplas da relação r que cabe em um bloco;
- V(A,r): é o número de valores distintos que aparecem na relação r para o atributo A. Se A é uma chave para a relação r, V(A,r) = ao número de tuplas de r (n_r).
- SC (A, r): é a cardinalidade de seleção do atributo A da relação r.
  - É o número médio de registros que satisfazem uma condição de igualdade no atributo A.
    - SC(A,r) = 1 se A é um atributo-chave de r;
    - Para um atributo que não é chave, estimamos que os valores distintos de V(A,r) são distribuídos uniformemente entre as tuplas, produzindo SC(A,r) = (n_r / V(A,r))

Algoritmo 2 – (busca binária) – S2

- Se o arquivo é ordenado em um atributo e a condição de seleção é uma comparação de igualdade neste atributo, podemos usar uma busca binária para localizar os registros que satisfazem a seleção.
- Neste caso a estimativa é:

$$E_{S2} = [\log_2(b)] + [SC(A,r)/f_r] - 1$$

  - O primeiro termo $[\log_2(b)]$ contabiliza o custo para localizar a primeira tupla por meio da busca binária nos blocos;
  - O número total de registros que satisfarão a seleção é SC(A,r), e esses registros ocuparão $[SC(A,r)/f_r]$ blocos, dos quais um já havia sido recuperado (por isso o -1).
  - Se a condição de igualdade estiver em um atributo-chave, então SC(A,r) = 1, e a estimativa se reduz a $E_{S2} = [\log_2(b)]$.

Exemplo

- Suponha as seguintes informações estatísticas para uma relação *conta*:
  - f_{conta} = 20 (ou seja, 20 tuplas de *conta* cabem em um único bloco);
  - V(nome_agência, conta) = 50 (ou seja, existem 50 agências com nomes diferentes);
  - V(saldo, conta) = 500 (ou seja, existe 500 valores diferentes de saldos nesta relação);
  - n_{conta} = 10.000 (ou seja, a relação *conta* possui 10.000 tuplas).
- Considere a consulta:

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

Exemplo

- Como a relação tem 10.000 tuplas, e cada bloco mantém 20 tuplas, o número de blocos é b_{conta} = 500 (10.000/20);
- Uma varredura de arquivo simples faria 500 acessos a blocos, supondo que o atributo da condição não fosse atributo-chave. Senão, seriam em média 250 acessos;
- Suponha que *conta* esteja ordenado por *nome_agência*.
- Como V(nome_agência, conta) = 50, esperamos que 10.000/50=200 tuplas da relação conta pertençam à agência Perryridge;
- Essas tuplas caberiam em 200/20 = 10 blocos;
- Uma busca binária acessa $[\log_2(500)]$ = 9 para encontrar o primeiro registro ;
- Assim o custo total seria: 9 + 10 – 1 = 18 acessos a bloco.

Projetos para teste e exercícios

Sistema Bancário simplificado

- Conta (<u>número da conta</u>, saldo, #nome da agência);
- Cliente (<u>nome do cliente</u>, rua, cidade);
- Empréstimo (<u>número do empréstimo</u>, valor, #nome da agência);
- Agência (<u>nome da agência</u>, fundos, cidade);
- Depositante (#<u>nome do cliente</u>, #<u>número da conta</u>);
- Devedor (#<u>nome do cliente</u>, #<u>número do empréstimo</u>);

nome_agência	cidade_agência	fundos
Downtown	Brooklyn	900000
Redwood	Palo Alto	210000
Perrydige	Horseneck	170000
Mianus	Horseneck	40000
Round Hill	Horseneck	8000000
Pownal	Bennington	30000
North Town	Rye	370000
Brighton	Brooklyn	710000

agência

nome_cliente	rua_cliente	cidade_cliente
Jones	Main	Harrison
Smith	North	Rye
Hayes	Main	Harrison
Curry	North	Rye
Lindsay	Park	Pittfield
Turner	Putnam	Stamford
Williams	Nassau	Princeton
Adams	Spring	Pittsfield
Johnson	Alma	Palo Alto
Glenn	Sand Hill	Woodside
Brooks	Senator	Brooklyn
Green	Walnut	Stamford

cliente

nome_cliente	número_conta
Johnson	A-101
Smith	A-215
Hayes	A-102
Turner	A-305
Johnson	A-201
Jones	A-217
Lindsay	A-222

depositante

nome_agência	número_conta	saldo
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Bringhton	A-201	900
Redwood	A-222	700
Bringhton	A-217	750

conta

Junções

- *equi_join*: designação para uma junção da forma r $|X|_{r.A=s.B}$ s, em que A e B são atributos ou conjuntos de atributos das relações *r* e *s*, respectivamente.
- O exemplo usado será:

depositante |X| cliente

Veremos dois algoritmos:

Junção de Laços Aninhados
Junção Sort-Merge

Informações de catálogo

- Suponha as seguintes informações de catálogo:
  - $n_{cliente}$ = 10.000
  - $f_{cliente}$ = 25, o que implica $b_{cliente}$ = 10.000/25 = 400
  - $n_{depositante}$ = 5.000
  - $f_{depositante}$ = 50, o que implica $b_{depositante}$ = 5.000/50 = 100
  - V(nome_cliente, depositante) = 2.500, o que implica que, em média, cada cliente tem duas contas
- Suponha ainda que nome-cliente em *depositante* seja uma chave estrangeira vinda de *cliente*.

Junção de Laço Aninhado

```
for each tupla t_r 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;
                se satisfizerem, adicione tr.ts ao
                resultado
            end
    end
```

r: relação externa
s: relação interna
$t_r.t_s$: tupla obtida concatenando os valores dos atributos das tuplas t_r e t_s

Junção de Laço Aninhado

- Este algoritmo não requer índices e pode ser usado seja qual for a condição de junção.
- É um algoritmo caro já que examina todos os pares de tuplas nas duas relações. O número de pares de tuplas a ser considerado é $n_r * n_s$ (para cada registro r tem-se que executar uma varredura completa em s).
- No pior caso o buffer pode manter apenas um bloco de cada relação, e um total de **br + (nr * bs)** acessos à blocos serão necessários (ou seja, os blocos da relação r (br) são lidos uma vez por ocasião do laço mais externo e, os blocos da relação s (bs) são lidos para cada vez que uma tupla de r precisa ser comparada com todas as tuplas de s por ocasião do laço mais interno)
- No melhor caso, há espaço suficiente para que ambas as relações caibam na memória, assim cada bloco terá de ser lido somente uma vez, conseqüentemente, apenas **br + bs** acessos à blocos serão necessários.
- Note que, se a relação menor couber completamente na memória, é melhor usar essa relação como a mais interna.

Junção de Laço Aninhado

- Uma situação intermediária: cabem mais de um bloco na memória, mas não cabem as duas relações de uma vez:
  - É vantajoso ler para a memória, de uma só vez, quantos blocos forem possíveis do arquivo cujos registros forem utilizados no laço externo (ou seja, $n_B - 2$ blocos).
  - - 2 porque é preciso reservar espaço para ler um bloco da outra relação e um bloco de buffer adicional é necessário para conter os registros resultantes, após sofrerem a junção. O conteúdo desse bloco deve ser descarregado para o disco
  - Em seguida, os blocos do segundo arquivo devem ser lidos, um por vez, para desempenharem o seu papel no laço interno
  - E o custo é:

$$b_r + (b_s * \lceil b_r / (n_B - 2) \rceil)$$

Exemplo

- Considere a junção natural de depositante e cliente. Suponha que não existem índices para estas relações. Suponha que *depositante* é a relação mais externa e *cliente* é a relação mais interna.
- 5.000 * 10.000 tuplas serão examinadas.
- Pior caso: 5.000 * 400 + 100 = 2.000.100 acessos à disco. (a relação menor é usada no laço externo)
- Melhor caso: 400 + 100 = 500 acessos à disco.
- Trocando as relações dos laços internos e externos: 10.000 * 100 + 400: 1.000.400 acessos à disco. (a relação menor é usada no laço interno)

Merge-junção

- *Junção sort-merge* (ordenação-fusão):
  - Se os registros de R e S estiverem classificados (ordenados) fisicamente pelos atributos de junção A e B, respectivamente, poderemos implementar a junção da maneira mais eficiente possível.
  - Ambos os arquivos são varridos simultaneamente na ordem dos atributos de junção, fazendo a correspondência dos registros que possuem os mesmos valores para A e B.
  - Se os arquivos não estiverem classificados, eles deverão ser classificados primeiro por meio de uma ordenação externa.
  - Pares de blocos de arquivos são ordenadamente copiados para *buffers* de memória, e os registros de cada arquivos são varridos apenas uma vez (a menos que A e B não sejam atributos chaves e aí o método precisa ser modificado).
  - Índices proporcionam a capacidade de acessar (varrer) os registros na ordem dos atributos de junção, mas os registros de fato estão ordenados de que forma no arquivo?

$T \leftarrow R \,|X|_{A=B}\, S$

Ordenar as n tuplas de R baseando-se no atributo A;
Ordenar as m tuplas de S baseando-se no atributo B;

```
Inicializar i ← 1 , j ← 1;
Enquanto (i <= n) e (j <= m) faça
{
    se Ri[A] > Sj[B]                  se o valor do primeiro A é maior que o valor
          então j ← j + 1             do primeiro B, avance em B

    senão se Ri[A] < Sj[B]            se o valor do primeiro A é menor que o valor
          então i ← i + 1             do primeiro B, avance em A
         senão
          {
                    bloco do próximo slide
          }
    i← i + 1; j ← j + 1;
}
```

```
{
(* Ri[A] = Sj[B], portanto realizamos o output de uma tupla: resultado da
   junção*)
   output a tupla combinada <Ri[A], Sj[B]> em T;

(* output outras tuplas correspondentes a Ri se houver*)
   l ← j + 1;
   enquanto (l<=m) and (Ri[A] = Sl[B]) faça
        output a tupla combinada <Ri[A], Sl[B]>  em T
        l ← l + 1

(* output outras tuplas correspondentes a S(j), se houver *)
   k ← i + 1;
   enquanto (k<=n) and (Rk[A] = Sj[B]) faça
        output a tupla combinada <Rk[A], Sj[B]>  em T
        k ← k + 1;
}
```

Junções complexas

- Junção com condição conjuntiva:

$$r \,|X|_{\theta_1 \wedge \theta_2 \wedge \ldots \theta} \, s_n$$

As junções nas condições individuais podem ser resolvidas, por exemplo, pelo algoritmo de junção por laços aninhados:

$r \,|X|_{\theta_1} s$, $r \,|X|_{\theta_2} s$, $r \,|X|_{\theta_n} s$ e assim por diante.

A junção global pode ser realizada calculando o resultado de uma dessas junções mais simples e depois testando (a esse resultado) as tuplas produzidas pelas outras junções.

Junções complexas

- Junção com condição disjuntiva:

$$r \,|X|_{\theta_1 \vee \theta_2 \vee \ldots \theta_n} \, s$$

- Neste caso, a junção pode ser calculada como a união dos registros nas junções individuais.

Junções complexas

- Considere uma junção envolvendo três relações:
  empréstimo |X| depositante |X| cliente
- Neste caso, além da escolha da estratégia para o processamento da junção, tem-se ainda que escolher qual junção calcular primeiro. Vejamos algumas estratégias:
  - **Estratégia 1**: calcule a junção depositante |X| cliente usando qualquer técnica. Usando o resultado intermediário, calcule: empréstimo |X| (depositante |X| cliente);
  - **Estratégia 2**: faça como na estratégia 1, mas calcule primeiro empréstimo |X| depositante, e então faça a junção do resultado com cliente.

  Ainda uma outra ordem de junções pode ser feita.

Junções complexas

- Estratégia 3: Em vez de executar duas junções, execute o par de junções, da seguinte forma:
  - Assuma dois índices:
    - Um para o número_empréstimo em empréstimo;
    - Um para o nome_cliente em cliente.
  - Considere cada tupla t em depositante. Para cada t, procure as tuplas correspondentes no índice *cliente* e as tuplas correspondentes no índice *empréstimo*.
  - Assim, cada tupla de depositante é examinada exatamente uma vez.

O custo relativo desse procedimento depende da forma como as relações estão armazenadas, da distribuição de valores dentro das colunas e da presença de índices.

Projeção $T \leftarrow \pi_{\text{<lista de atributos>}} (R)$

```
Criar uma tupla t[<lista de atributos>] em T' para cada tupla t de R;

(*T' contém o resultado da projeção ANTES da eliminação de duplicatas*)

Se <lista de atributos> incluir uma chave de R
    então T ← T';
Senão
    {
        ordenar as tuplas de T'  (assuma aqui o custo da ordenação)
        inicializar i ← 1, j← 2;
        enquanto i <= n faça
            {
                output a tupla T'[i] em T;
                enquanto T'[i] = T'[j] and j<= n faça
                    j ← j + 1; (* eliminar duplicatas)
                i ← j; j ← j + 1;
            }
    }
```

União $T \leftarrow R \cup S$

```
Ordenar as tuplas de R e S utilizando os mesmos atributos de ordenação;

Inicializar i ← 1; j ← 1;
Enquanto (i <= n) e (j <= m) faça
    {
        se R(i) > S(j) então
            {
                output S(j) em T;
                j ← j + 1;
            }
        se R(i) < S(j) então
            {
                output R(i) em T;
                i ← i + 1;
            }
        else
            j ← j + 1; (*R(i)=S(j), portanto, pular uma das tuplas duplicatas*)
    }
```

Intersecção $T \leftarrow R \cap S$

```
Ordenar as tuplas de R e S utilizando os mesmos atributos de ordenação;

Inicializar i ← 1; j ← 1;
Enquanto (i <= n) e (j <= m) faça
    {
        se R(i) > S(j) então
            {
                j ← j + 1;
            }
        else se R(i) < S(j) então
            {
                i ← i + 1;
            }
        else
            {
                output R(i) em T; (*R(i)=S(j), portanto, fazemos o output da tupla*)
                i ← i + 1; j ← j + 1;
            }
}
```

Diferença T ← R - S

Ordenar as tuplas de R e S utilizando os mesmos e únicos atributos de ordenação;

```
Inicializar i ← 1; j ← 1;
Enquanto (i <= n) e (j <= m) faça
    {
        se R(i) > S(j) então
                {
                        j ← j + 1;
                }
        else se R(i) < S(j) então
                {
                        output R(i) em T; (*R(i) não tem S(j) correspondente)
                        i ← i + 1;
                }
            else
                i ← i + 1; j ← + 1
    }
```

Estimativa do tamanho das junções

Estimativa do tamanho das junções

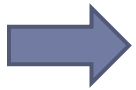
- O produto cartesiano r X s contem n_r * n_s tuplas.
- Cada tupla deste produto cartesiano ocupa s_r + s_s bytes.
- Assim podemos calcular o tamanho do produto cartesiano.

- Para junção natural ... Sejam r(R) e s(S) duas relações:
  - Se R ∩ S = *Vazio*, entao r |X| s é igual a r X s;
  - Se R ∩ S é uma chave para R, então sabemos que uma tupla de s juntar-se com no máximo uma tupla de r. Assim, o número de tuplas na junção não é maior que o número de tuplas de s.
  - Se R ∩ S é uma chave estrangeira para s – vinda de r – , então o número de tuplas em r |X| s é exatamente igual ao número de tuplas em s.

Estimativa do tamanho das junções

Esquema no Korth!

- No exemplo: depositante |x| cliente, nome_cliente em depositante e uma chave estrangeira vinda de cliente.
- O tamanho do resultado é exatamente $n_{depositante}$, que é 5.000 tuplas;
- Com calcular o tamanho da junção quando R ∩ S não é uma chave para R ou para S?



Estimativa do tamanho das junções

- Suponha que cada valor aparece com probabilidade igual.
- Considere uma tupla t de r e suponha R ∩ S = {A}.
- Estima-se que a tupla t produz

$n_s / V(A,s)$

Número de tuplas na relação **s**!

Número de valores distintos que aparecem na relação **s** para o atributo **A**.

tuplas em r |X| s, uma vez que esse é o número médio de tuplas em s com um determinado valor para os atributos A.

- Considerando todas as tuplas em r, estima-se que há

$$n_r * n_s / V(A, s)$$

tuplas em r |X| s.

Estimativa do tamanho das junções

- Observe que se invertermos os papeis de r e s, as estimativas resultariam em valores diferentes se V(A,r) <> V(A,s).

- Se isso acontece, há a probabilidade de haver tuplas pendentes que não participam da junção, pois na estimativa do slide anterior estamos considerando que todas as tuplas de uma relação terão sua correspondente na outra relação.

- Técnicas mais sofisticadas para a estimativa do tamanho da junção devem ser usadas se a hipótese de distribuição uniforme não puder ser considerada.

Estimativa do tamanho das junções

- Calculando a estimativa do tamanho para depositante |X| clientes, sem utilizar informações sobre chaves estrangeiras.

- Como
  - n_r = 10.000
  - n_s = 5.000
  - V(nome_cliente, depositante) = 2.500 e
  - V(nome_cliente, cliente) = 10.000,

- as duas estimativas que obtemos são:

(10.000 * 5.000) / 2.500 = 20.000

(5.000 * 10.000)/10.000 = 5.000

Equivalência

Equivalência de Expressões

- Encontre os nomes de todos os clientes que possuem uma conta em qualquer agencia localizada no Brooklyn.

$$\pi_{\text{nome\_cliente}} (\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia} \mid X \mid (\text{conta} \mid X \mid \text{depositante})))$$

- Para resolver esta expressão, seguindo a forma como ela esta escrita, é necessário criar uma relação intermediária grande (a junção das três relações).

- Entretanto, somente as tuplas que pertencem às agências localizadas no “Brooklyn” são interessantes

- Vamos então reescrever a consulta, eliminando a necessidade de considerar as tuplas que não tem cidade_agencia = “Brooklyn”, reduzindo o tamanho do resultado intermediario:

$$\pi_{\text{nome\_cliente}} ((\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia})) \mid X \mid (\text{conta} \mid X \mid \text{depositante}))$$

Regras de transformação

1. Cascata de σ: uma condição de seleção conjuntiva pode ser quebrada em uma cascata (ou seja, uma seqüência) de operações σ individuais.

2. Comutatividade σ: a operação σ é comutativa.

3. Cascata de π: em uma cascata (seqüência) de operações π, todas, exceto a última, podem ser ignoradas.

4. Comutatividade de σ e π: se a condição de seleção **c** envolver apenas aqueles atributos A1, ... An da lista de projeção, as duas operações podem ser comutadas.

5. Comutatividade de |X| e X: a operação de |X| é comutativa, assim como X

6. Comutatividade de σ e |X| (ou X): Se todos os atributos da condição de seleção **c** envolverem apenas os atributos de uma das relações participantes da junção – digamos R -, as duas operações podem ser comutadas como segue:

$$\sigma_C (R \,|X|\, S) \equiv (\sigma_C (R)) \,|X|\, S$$

Regras de transformação

7. Comutatividade de π e |X| (ou X): similar à 6.

8. Comutatividade de operações de conjuntos: as operações de $\cap$ e $\cup$ são comutativas, porém – não é.

9. Associatividade de |X|, X, $\cap$ e $\cup$: essas quatro operação são associativas individualmente, ou seja, se θ significa qualquer uma dessas quatro operações, temos

$$(R \theta S) \theta T \equiv R \theta (S \theta T)$$

10. Comutatividade de σ e das operações de conjunto: a operação σ comuta com as operações $\cap$, $\cup$ e -. Se θ significar qualquer uma dessas três operações, temos

$$\sigma_c(R \theta S) \equiv (\sigma_c(R)) \theta (\sigma_c(S))$$

11. A operação π comuta com $\cup$

12. Conversão de uma seqüência (σ, X) em |X|, se os atributos da selação correspondem aos atributos de condição da junção.

Exemplos

- Exemplo 1:

$$\pi_{\text{nome\_cliente}} (\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia} \mid X \mid (\text{conta} \mid X \mid \text{depositante})))$$

$$\pi_{\text{nome\_cliente}} ((\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia})) \mid X \mid (\text{conta} \mid X \mid \text{depositante}))$$

- Exemplo 2:

$$\pi_{\text{nome\_cliente}} (\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”} \text{ and } \text{saldo} > 1000} (\text{agencia} \mid X \mid (\text{conta} \mid X \mid \text{depositante})))$$

$$\pi_{\text{nome\_cliente}} (\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”} \text{ and } \text{saldo} > 1000} ((\text{agencia} \mid X \mid \text{conta}) \mid X \mid \text{depositante}))$$

$$(\pi_{\text{nome\_cliente}} (\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”} \text{ and } \text{saldo} > 1000} (\text{agencia} \mid X \mid \text{conta})) \mid X \mid \text{depositante})$$

- Exemplo3: Examinando uma subexpressão interna:

$$\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”} \text{ and } \text{saldo} > 1000} (\text{agencia} \mid X \mid \text{conta}))$$

$$\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\sigma_{\text{saldo} > 1000} (\text{agencia} \mid X \mid \text{conta}))$$

$$\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia}) \mid X \mid \sigma_{\text{saldo} > 1000} (\text{conta})$$

- Exemplo 4: Usando projeções

$$\pi_{\text{nome\_cliente}} ((\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia}) \mid X \mid \text{conta}) \mid X \mid \text{depositante})$$

$$\pi_{\text{nome\_cliente}} ((\pi_{\text{numero\_conta}} ((\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}} (\text{agencia})) \mid X \mid \text{conta})) \mid X \mid \text{depositante})$$

Considerações

- Uma boa ordenação de operações de junção é importante para reduzir o tamanho dos resultados intermediários.

$$\pi_{\text{nome\_cliente}}((\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}}(\text{agencia})) \,|X|\, \text{conta} \,|X|\, \text{depositante})$$

- Poderíamos executar conta |X| depositante primeiro e, então, fazer a junção do resultado com:

$$\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}}(\text{agencia})$$

- Entretanto, conta |X| depositante provavelmente é uma relação grande, já que contém uma tupla para cada conta.

- Em contrapartida

$$\sigma_{\text{cidade\_agencia} = \text{“Brooklyn”}}(\text{agencia}) \,|X|\, \text{conta}$$

- e, provavelmente, uma relação pequena.

Considerações

- Para confirmar, observe que, como o banco tem um grande número de agências amplamente distribuídas, e provável que apenas uma fração pequena dos clientes do banco tenha conta em agências localizadas no Brooklyn.

- Assim, a expressão precedente resulta em uma tupla para cada conta mantida em uma agencia localizada no Brooklyn. Então, a relação temporária que é preciso armazenar é menor que a que ter-se-ia obtido fazendo conta |X| depositante primeiro.

Junção (conta |X| depositante)

nome_agência	número_conta	saldo
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Bringhton	A-201	900
Redwood	A-222	700
Bringhton	A-217	750

nome_cliente	número_conta
Johnson	A-101
Smith	A-215
Hayes	A-102
Turner	A-305
Johson	A-201
Lindsay	A-222
Jones	A-217

Junção (agência |X| (conta |X| depositante))

nome_agência	número_conta	saldo
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Bringhton	A-201	900
Redwood	A-222	700
Bringhton	A-217	750

nome_cliente
Johnson
Smith
Hayes
Turner
Johnson
Lindsay
Jones

nome_agência	cidade_agência	fundos
Downtown	Brooklyn	900000
Mianus	Horseneck	40000
Perrydige	Horseneck	170000
Round Hill	Horseneck	8000000
Brighton	Brooklyn	710000
Redwood	Palo Alto	210000
Brighton	Brooklyn	710000

Junção (conta |X| depositante)

nome_agência	número_conta	saldo
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Bringhton	A-201	900
Redwood	A-222	700
Bringhton	A-217	750

nome_cliente	número_conta
Johnson	A-101
Smith	A-215
Hayes	A-102
Turner	A-305
Johnson	A-201
Lindsay	A-222
Jones	A-217

$\sigma_{\text{cidade\_agência} = \text{"Brooklyn"}}$ (agência)

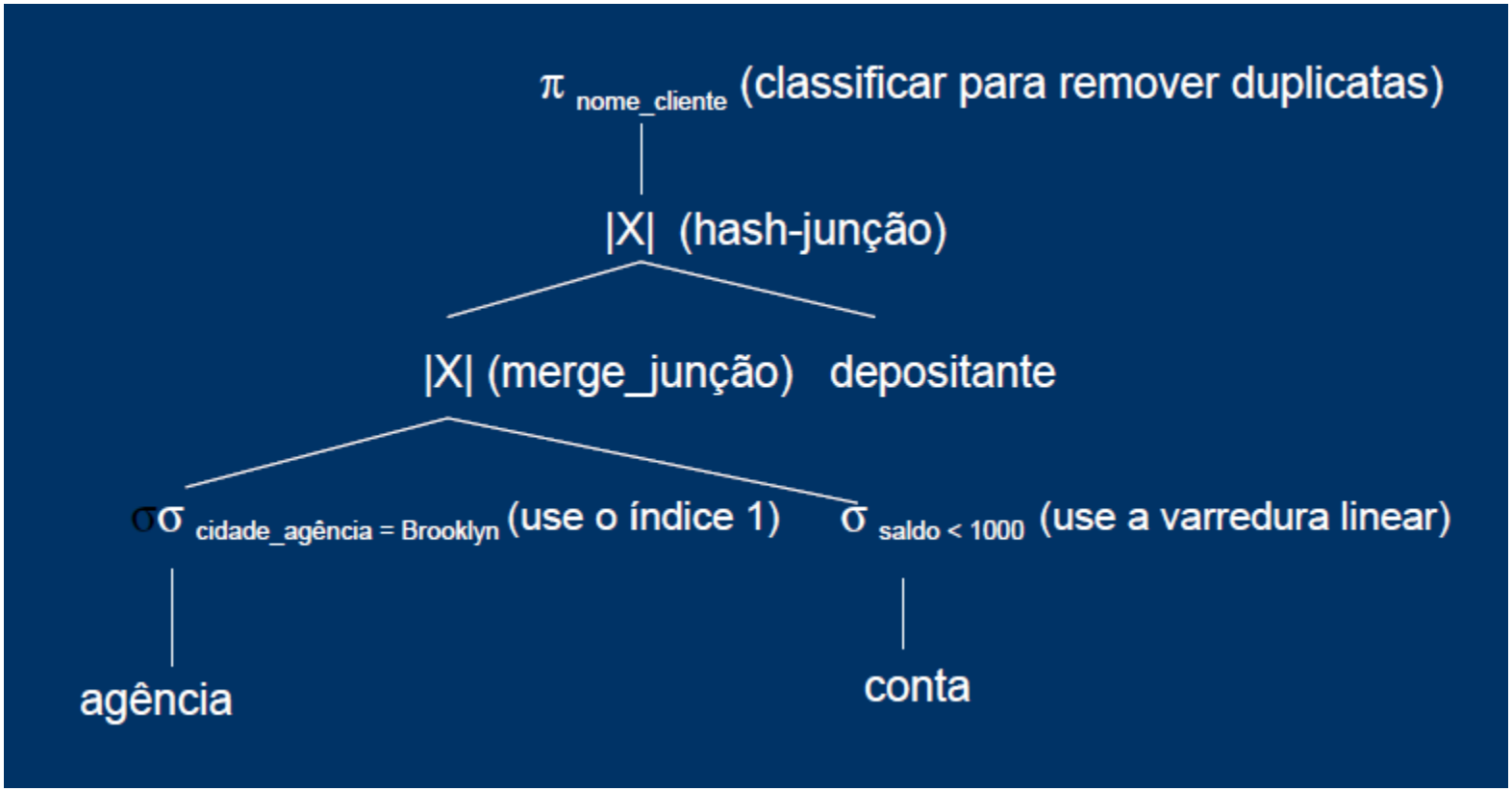
nome_agência	cidade_agência	fundos
Downtown	Brooklyn	900000
Brighton	Brooklyn	710000

($\sigma_{\text{cidade\_agência} = \text{"Brooklyn"}}$ (agência)) |X| (conta |X| depositante)

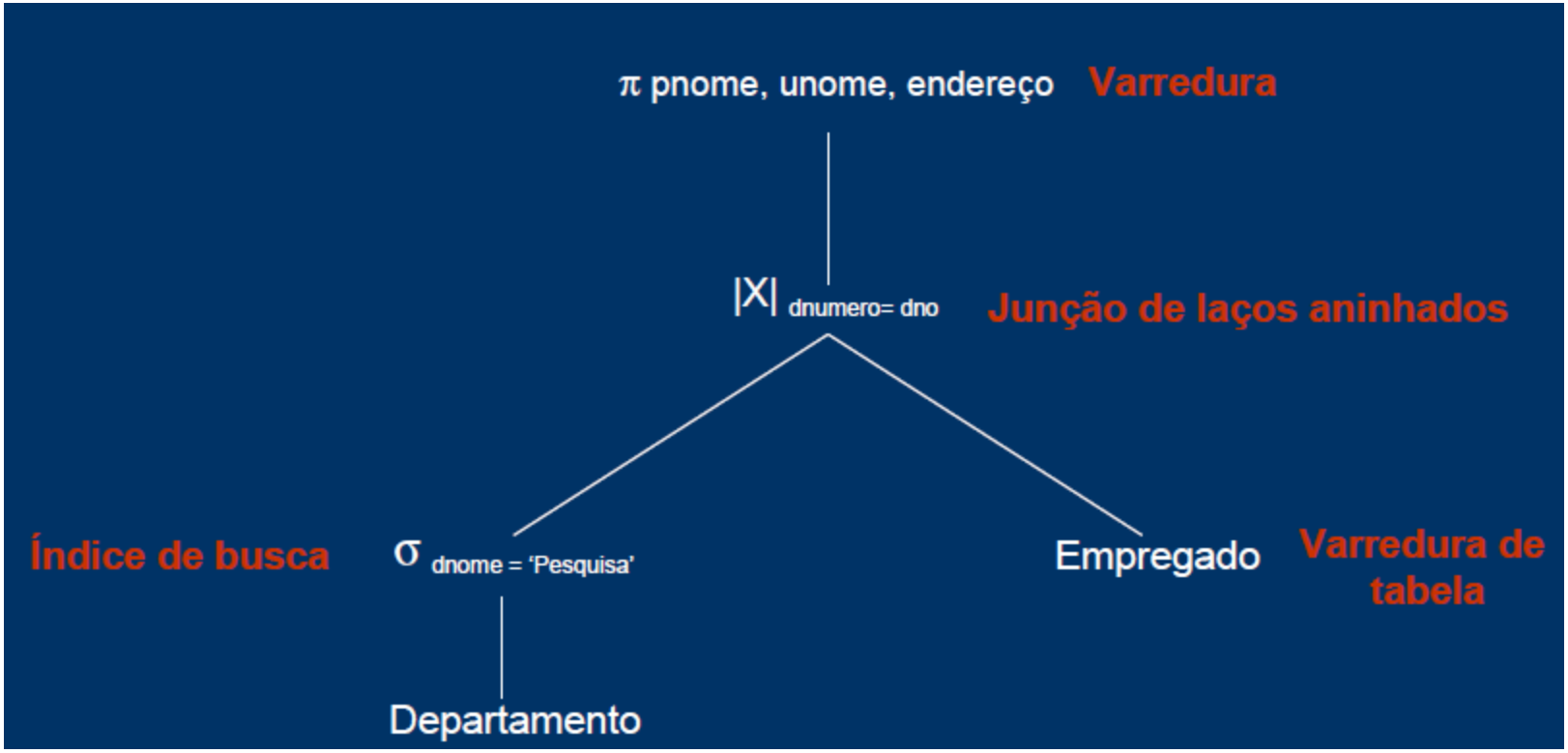
nome_agência	número_conta	saldo	nome_cliente	cidade_agência	fundos
Downtown	A-101	500	Johnson	Brooklyn	900000
Bringhton	A-201	900	Johnson	Brooklyn	710000
Bringhton	A-217	750	Jones	Brooklyn	710000

Árvores de Consulta / Planos de Execução

Árvores de Consulta / Planos de Execução



Árvores de Consulta / Planos de Execução

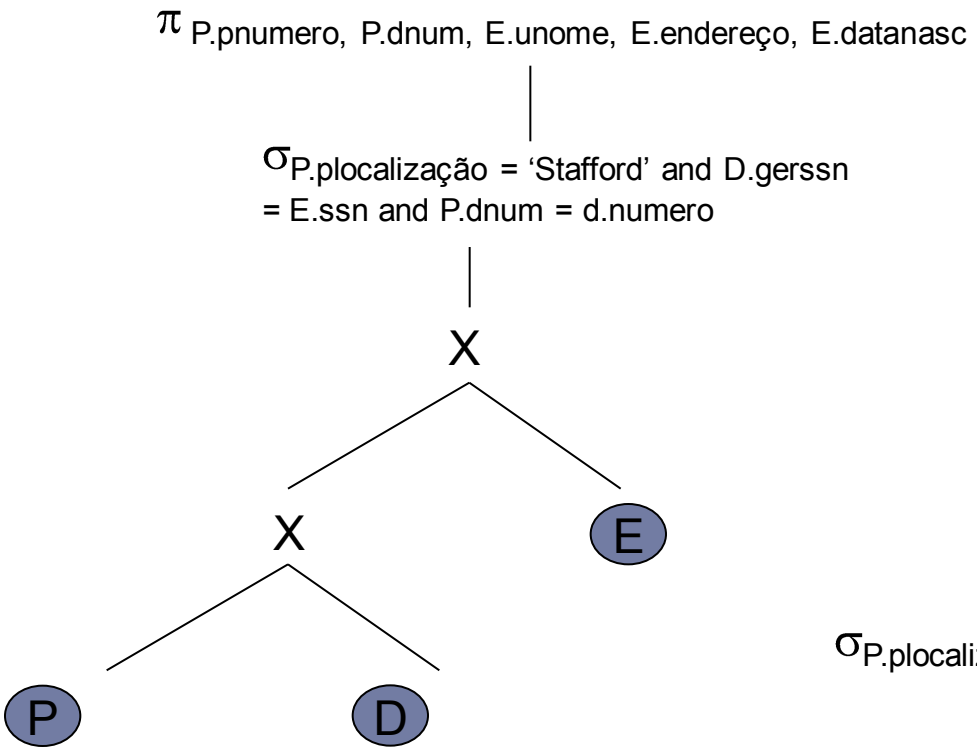


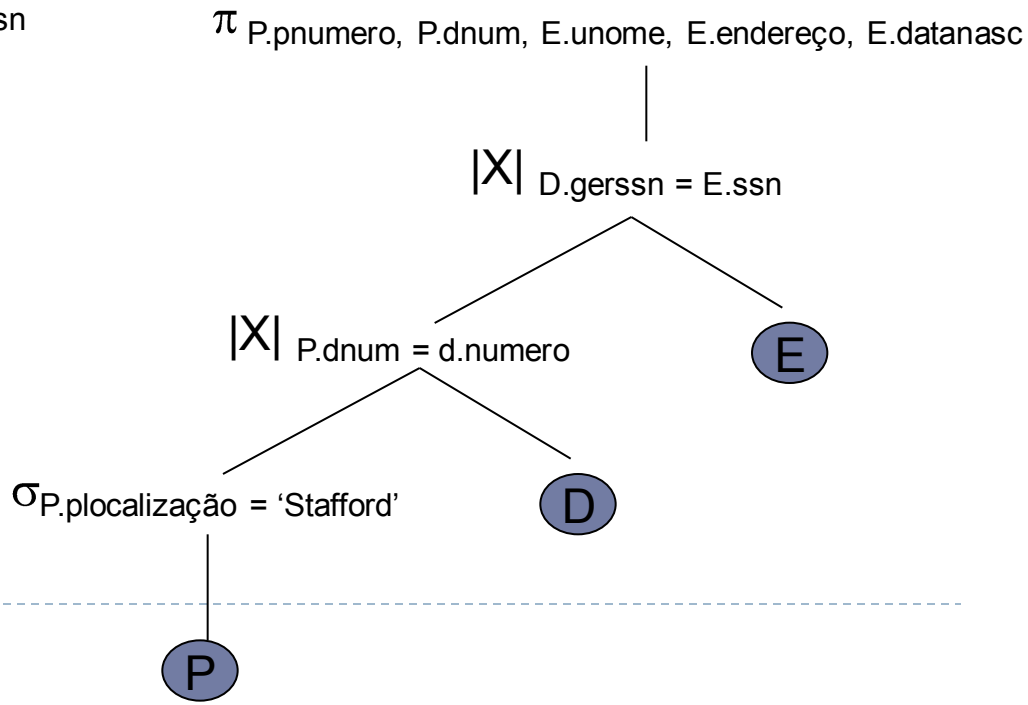
Árvores de Consulta

Select P.numero, P.dnum, E.unome, E.endereço, E.datanasc

From projeto **as** P, departamento **as** D, empregado **as** E

Where P.dnum = D.dnumero **and** D.gerssn = E.ssn and P.plocalização = ‘Stafford’





Escolha de um plano de execução

- Um modo de escolher um plano de avaliação para uma expressão de consulta e simplesmente escolher o algoritmo mais barato para avaliar cada operação. E, olhando para os níveis da árvore, escolhe-se qualquer ordenamento para a execução das operações, desde que as operações nas camadas mais baixas da arvore sejam executadas antes das operações nas camadas mais altas.

- Entretanto, essa estratégia pode não ser a melhor. Embora uma merge-juncao, sob certas condições, possa ser mais cara que uma hash-juncao, ela consegue prover um resultado classificado que torna mais barata a avaliação de uma operação posterior (como uma eliminação de duplicatas ou uma outra merge-juncao).

- Para escolher o melhor algoritmo global, se deve considerar ate mesmo os algoritmos que não são os melhores para as operações individuais.

- Abordagens para escolha do melhor plano de avaliação:
  - Baseada no custo de todos os planos;
  - Heurística

Otimização Baseada em Custo

- O otimizador baseado no custo gera uma série de planos de avaliação a partir de uma determinada consulta usando as regras de equivalência e escolhe aquele de menor custo.

- Para uma consulta complexa, o numero de planos diferentes por ser muito grande.

- Diferentes técnicas podem ser usadas para diminuir o numero de planos a serem avaliados:
  - Quando examinamos os planos para uma expressão, podemos terminar após examinarmos apenas uma parte da expressão se determinarmos que o plano mais barato para aquela parte já está mais caro que a avaliação mais barata para uma expressão completa examinada anteriormente.

Interpretando as regras de equivalência

- A regra 1, ao ser usada, quebra quaisquer operações SELECT com condições conjuntivas em uma cascata de operações SELECT, permitindo um maior grau de liberdade para transferir operações SELECT para ramos diferentes e abaixo na árvore.

- Usando as regras 2, 4, 6, e 10 relativas à comutatividade do SELECT com outras operações, move-se cada operação SELECT o mais longe para baixo na árvore que for permitido pelos atributos envolvidos na condição de seleção.

- Usando as regras 5 e 9, relativas à comutatividade e associatividade de operações binárias, rearraja-se os nós folhas da árvore utilizando o seguinte critério:
  - Posiciona as relações do nó folha com operações de SELECT mais restritvas, de forma que elas possam ser executadas o quanto antes.

- Usando a regra 12, combina-se uma operação de PRODUTO CARTESIANO com uma operação SELECT subseqüente na árvore, gerando uma operação JOIN se a condição representa uma condição de junção.

- Usando as regras 3, 4, 7 e 11, relativas à cascata de PROJECT e à comutação de PROJECT com outras operações:
  - Quebra-se e transfere-se as listas de atributos de projeção para baixo na árvore.
  - Identifica subárvores que representam grupos de operações que podem ser executadas por um único algoritmo.

Otimização Heurística

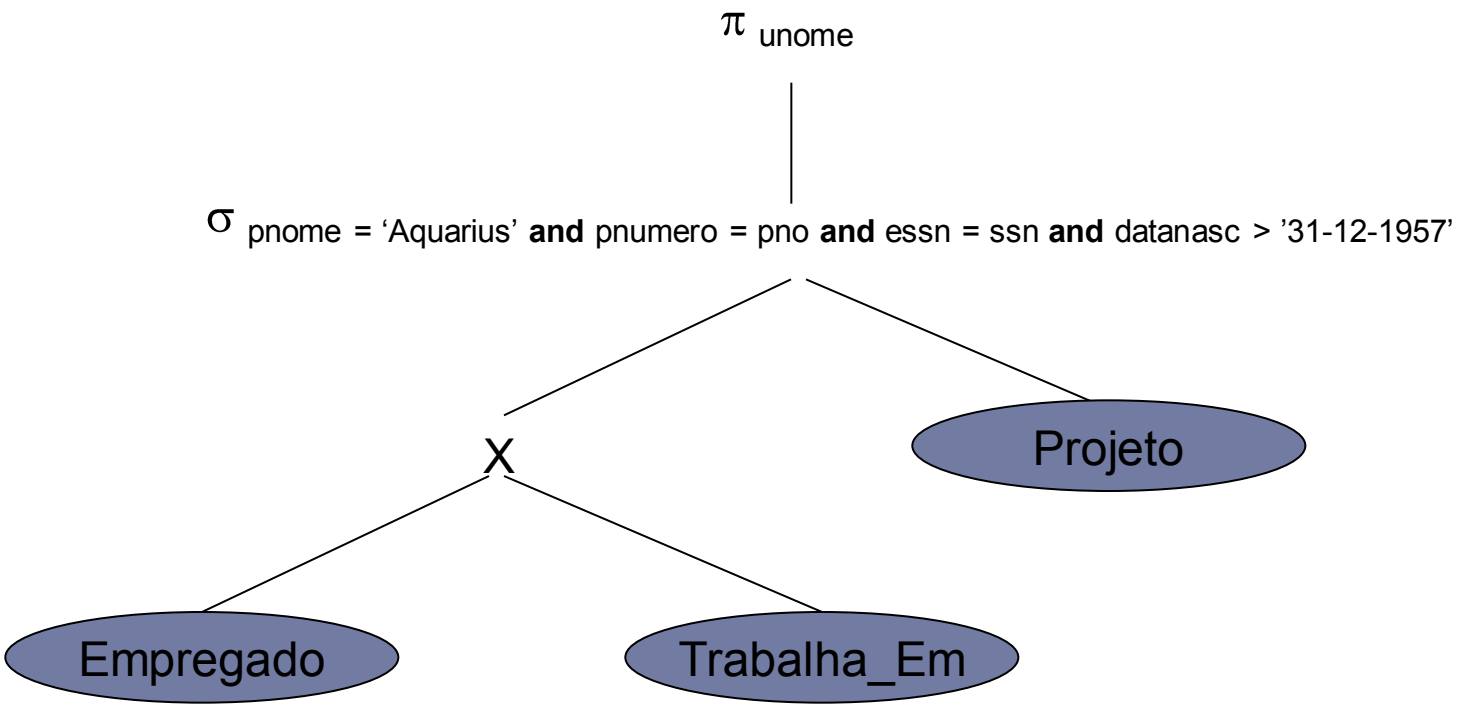
- Uma árvore de consulta pode ser transformada passo a passo em outra árvore de consulta mais eficiente.
- Entretanto é preciso assegurar que os passos de transformação sempre levem a uma árvore de consulta equivalente.
- Determinadas regras de transformação preservam essa equivalência.
- Regras heurística são utilizadas para transformar consultas da álgebra relacional:
  - Execute as operações de seleção assim que possível;
  - Execute projeções antes;
- Passos de um algoritmo de otimização heurística:
- .....

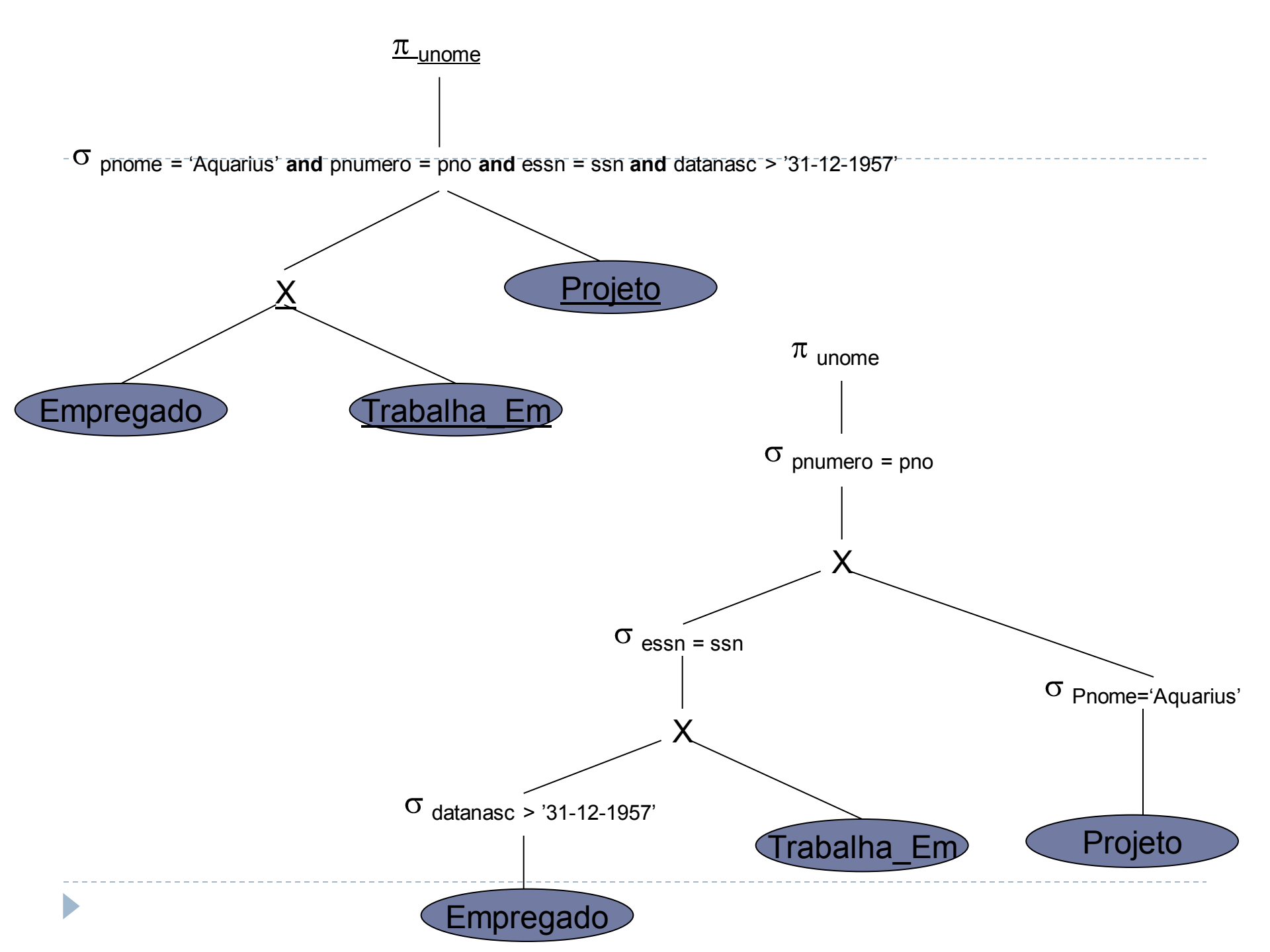
Passos ...

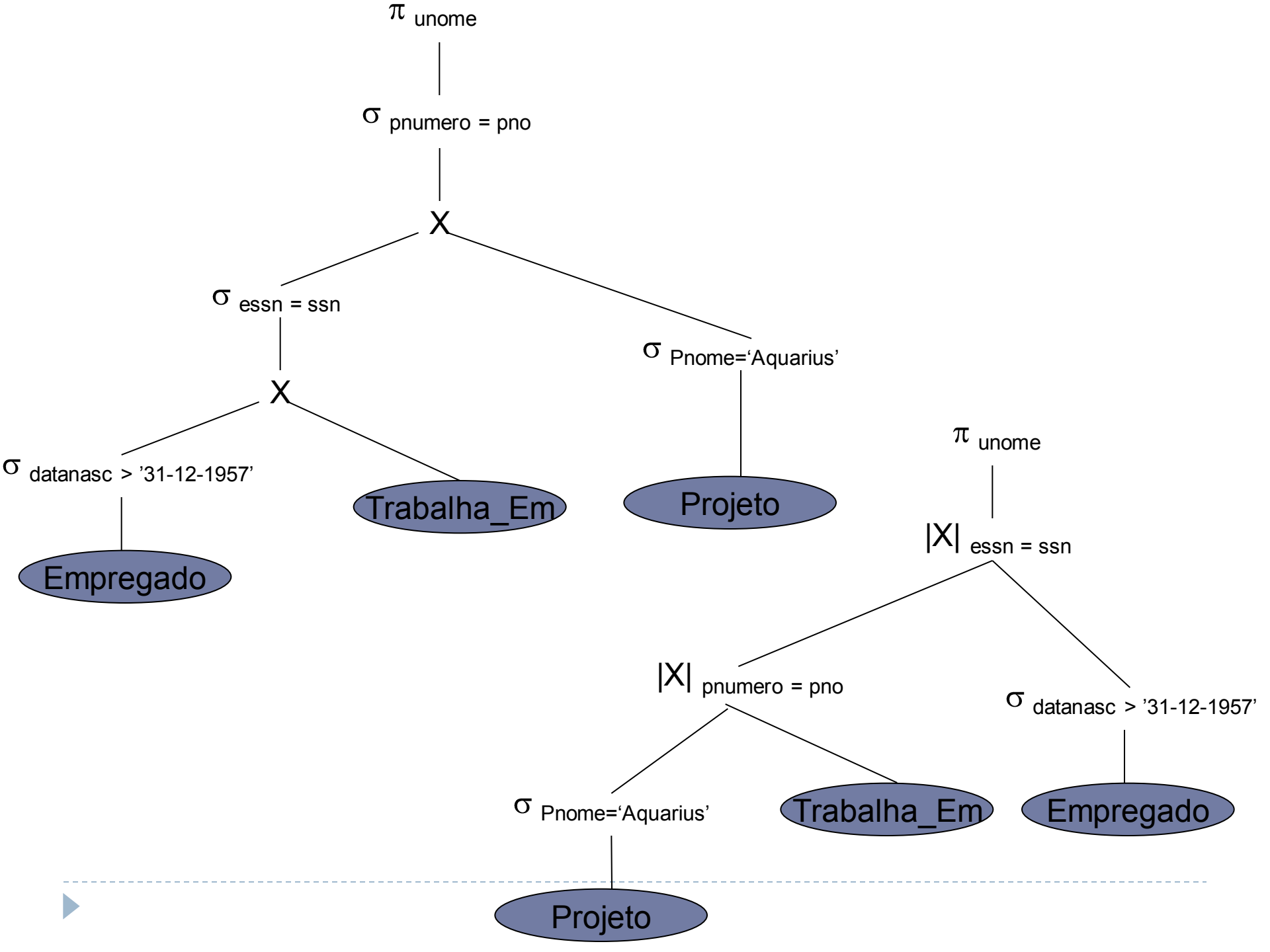
- Separe as seleções conjuntivas em uma seqüência de operações de seleção isoladas. Assim pode-se mover as operações de seleção para baixo na árvore.
- Movas as operações de seleção para baixo na árvore de consulta para que sua execução ocorra o mais cedo possível. A execução, o mais cedo possível das seleções reduz o custo de classificar e fazer o merge dos resultados intermediários grandes.
- Determine quais operações de seleção e de junção produzirão as menores relações. Reorganize a arvore de tal forma que as relações dos nos folhas com essas seleções sejam executados primeiro.
- Substitua as operações de produto cartesiano que são seguidas por uma condição de seleção por operações de junção.
- Separe as listas de projeção e mova-as o mais para baixo possível, criando projeções novas onde forem necessárias.
- Identifique aquelas subarvores cujas operações podem ser colocadas em *pipeline* e execute-as usando *pipelining*.
- OBS: No otimizador heurístico, a estratégia mais eficiente para cada operacao é escolhida.

Otimização Heurística - exemplo

Select unome
From Empregado, Trabalha_Em, Projeto
Where pnome = ‘Aquarius’ **and** pnumero = pno **and** essn = ssn **and** datanasc > ’31-12-1957’;




π unome
σ pnome = 'Aquarius' and pnumero = pno and essn = ssn and datanasc > '31-12-1957'
X
Projeto
Empregado
Trabalha_Em
π unome
σ pnumero = pno
X
σ essn = ssn
σ Pnome='Aquarius'
X
σ datanasc > '31-12-1957'
Trabalha_Em
Projeto
Empregado


π unome
σ pnumero = pno
X
σ essn = ssn
σ Pnome='Aquarius'
X
σ datanasc > '31-12-1957'
Trabalha_Em
Projeto
Empregado
π unome
|X| essn = ssn
|X| pnumero = pno
σ datanasc > '31-12-1957'
σ Pnome='Aquarius'
Trabalha_Em
Empregado
Projeto

Pensando em Sistemas Distribuídos

Processamento de Consultas Distribuído

- Para um sistema centralizado, o custo do processamento de consultas é baseado no número de acessos a disco.

- Para um sistema distribuído, ainda deve-se considerar:
  - O custo de transmissão dos dados via rede;
  - O potencial de desempenho ganho diante do fato de que diversos sites podem processar parte da consulta em paralelo.

- O objetivo então é encontrar a forma de executar uma consulta, que tenha a melhor relação custo/benefício.

Fragmentação
Replicação

Consulta Global
Consulta Local

Transformação de Consultas

- **Consulta: “ *Encontre todas as tuplas da relação conta”.***

- É uma consulta simples mas que pode ter um processamento não trivial em um sistema de banco de dados distribuídos. Isso áse a relação conta estiver fragmentada ou replicada ou ambas.

- Dependendo de como a relação se encontra, o número de estratégias possíveis para o processamento da consulta pode ser muito grande.

Considerações

- Se a relação **conta** estiver replicada é necessário escolher qual réplica utilizar para realizar a leitura.
- Se a réplica não está fragmentada escolhe-se a réplica cujo custo de transmissão, para o *site* onde a consulta será realizada, seja menor.
- Se a réplica estiver fragmentada, será necessário efetuar diversas junções ou uniões para a reconstrução da relação **conta.** E a escolha de quais réplicas usar será mais complicada.
- O otimizador de consultas deverá implementar muitas estratégias de avaliação e calcular os custos para realizar a escolha.

- Agora considere o exemplo: *"Encontre todas as tuplas da relação **conta** localizadas na cidade de Hillside".*

Transformação de consultas

$$\sigma_{\text{nome\_agencia} = \text{“Hillside”}}(\text{conta})$$

- Se Conta = conta1 U conta2 (fragmentação horizontal) então

$$\sigma_{\text{nome\_agencia} = \text{“Hillside”}}(\text{conta1} \cup \text{conta2})$$

- Usando o otimizador de consultas

$$\sigma_{\text{nome\_agencia} = \text{“Hillside”}}(\text{conta1}) \cup \sigma_{\text{nome\_agencia} = \text{“Hillside”}}(\text{conta2})$$

que inclui duas sub-expressões. Uma será executada no *site* da conta1 e a outra será executada no *site* da conta2. (paralelismo proporcionado pela distribuição de dados.)

Transformação de consultas

Usando o otimizador de consultas e conhecendo as definições dos fragmentos

- Se no *site* da conta1 só existem tuplas cujo nome-agencia = Hillside então a seleção pode ser eliminada.
- Se no site da conta2 só existem tuplas cujo nome-agencia = Valleyview, a definição do fragmento de Valleyview pode ser usada mas implicará num resultado vazio.
- Assim a estratégia final tem conta1 como resultado da consulta.

Processamento de junção simples

- Considere a seguinte expressão:

S1.conta |X| S2.depositante |X| S3.agencia

onde nenhuma das relações são replicadas ou fragmentadas e que o *site* **S1** originou a consulta → o resultado precisa ser produzido no **S1**.

- **Quais são as estratégias de processamento (???)**

Processamento de junção simples

- Enviar cópias das duas relações, que não se encontram no site S1, para o *site* S1.
- Usar uma estratégia comum aos SGBDs centralizados para processar localmente a consulta, no *site* S1.
  - apesar do processamento ficar restrito ao *site* S1, a consulta é global, pois em algum momento envolveu outros sites

- Enviar cópia da relação conta para o *site* S2 e lá obter temp1 = conta |X| depositante.
- Enviar temp1 do *site* S2 para o *site* S3 e obter temp2 = temp1 |X| agência em S3. Enviar o resultado de temp2 para S1.

- Aplicar estratégia similar a anterior, alterando as regras entre S1, S2 e S3.

Processamento de junção simples?

- Qual estratégia e a melhor?

  - Qual estratégia transporta o menor nímero de dados?
  - Qual e o custo de transmissão de dados entre dois determinados *sites*?
  - Qual e a velocidade de processamento em cada *site?*
  - Qual e o custo de transmissão de índices?
  - Qual e o custo de criação de índices?

Estratégia de Junção Parcial

- Suponha que se quer processar a junção r1 |X| r2 em que r1 e r2, com esquemas R1 e R2, são armazenadas nos *sites* S1 e S2, respectivamente. O resultado deve estar em S1.

- Se tivermos uma situação onde diversas tuplas de r2 não possam se juntar a nenhuma tupla de r1, então o transporte de r2 para S1 implica no transporte de tuplas que não contribuirão no resultado.

- É preferível a remoção dessas tuplas antes do transporte de dados.

- **Opção de estratégia de processamento ...**

Estratégia de Junção Parcial

1. Obter temp1 ← $\Pi_{R1 \cap R2}(r_1)$ em S1.
2. Transportar temp1 de S1 para S2.
3. Obter temp2 ← r_2 |X| temp1 em S2.
4. Transportar temp2 de S2 para S1.
5. Obter r_1 |X| temp2 em S1.

A relação resultante será a mesma de r1 |X| r2?

No passo 3, temp2 possui o resultado de r_2 |X| $\Pi_{R1 \cap R2}(r_1)$

No passo 5, calcula-se: r_1 |X| r_2 |X| $\Pi_{R1 \cap R2}(r_1)$

Podemos reescrever a expressão como segue: r_1 |X| $\Pi_{R1 \cap R2}(r_1)$ |X| r_2

E r_1 |X| $\Pi_{R1 \cap R2}(r_1)$ = r1 logo r_1 |X| r_2

Estratégia de Junção Parcial

- A estratégia é eficiente?
- Ela é vantajosa quando há poucas tuplas de r2 participando da junção.
- A economia de custo dessa estratégia provem do fato de ser necessário transportar somente temp2, em vez de todas as tuplas de r2 para S1.
- Um custo adicional é decorrente do transporte de temp1 para S2.

Estratégia de Junção X Paralelismo

S1.r1 |X| S2.r2 |X| S3.r3 |X| S4.r4

- r1 e transportada para S2 onde r1 |X| r2 e obtida;
- ao mesmo tempo, r3 e transportada para S4 onde r3 |X| r4 e obtida;
- S2 pode transportar as tuplas de r1 |X| r2 para S1 a medida que forem sendo produzidas;
- S4 pode transportar as tuplas de r3 |X| r4 para S1 a medida que forem sendo produzidas;
- Assim que as tuplas vão chegando em S1, a execução de (r1 |X| r2) |X| (r3 |X| r4) pode ter início.

Tuning de Sistemas Gerenciadores de Banco de Dados Relacionais

Baseado em Mini-Curso do 20º SBBD

Caetano Traina Jr.

Introdução

- Conceito – Database Tuning
  - Ajuste fino em banco de dados: refere-se às técnicas usadas pelo DBA para melhorar o desempenho de um SGBD frente a uma carga de consulta.
    - Preparo das consultas
    - Configuração do servidor
  - Conhecimento exigido
    - Funcionamento do SGBD
    - Processamento de consultas
  - Estratégia: conhecer o problema globalmente e executar ações localmente

Introdução

- Não se define o melhor ajuste
  - O bom ajuste depende das condições de otimização oferecidas pelo SGBD e da coleção de consultas que são submetidas a ele.

- As atividades de tuning são realizadas em todos os componentes de software de um empreendimento, inclusive em procedimentos operacionais.

Introdução

- Possibilidade de uso do senso comum
  - Facilita porque não é imprescindível o conhecimento de fórmulas e teoremas
  - Dificulta porque exige entendimento amplo e profundo
    - Da aplicação
    - Da construção do gerenciador
    - Do sistema operacional
    - Dos princípios físicos do hardware
  - Experiência em casos anteriores é bem-vinda

Introdução

- Pontos interessantes:
  - Quais são as otimizações em consultas que o gerenciador faz?
  - Quais são as otimizações em consultas que o gerenciador não faz?
  - Como tratar as operações de acesso indexado?
  - Como ajustar o controle de concorrência?
  - Como ajustar o log e os parâmetros de acesso físico do servidor?

Fluxos de Execução

- Todos os comandos passam pelo interpretador:
  - Linguagem de Controle de Dados (criação de base e estabelecimento de privilégios e condições para execução de transações):
    - Execução sequencial (sem otimização)
    - Existem procedimentos específicos para executar esses comandos no módulo executor
    - Processamento realizado de maneira direta
  - Linguagem de Definição de Dados (criação e manutenção de relações, índices, gatilhos, etc)
    - Relações do sistema e relações do usuário
    - Não requerem procedimentos específicos, apenas a conversão dos seus comandos em comandos DCL e DML

Fluxos de Execução

- Exemplo:
  - Quando o servidor recebe um comando DDL de criação de uma nova relação (uma relação definida pelo usuário), esse comando é substituído por:
    - Um comando da DML para inserir mais uma tupla na relação das relações;
    - Tantos comandos da DML quanto necessários para inserir/atualizar tuplas na relação de atributos;
    - E um comando DCL para criar fisicamente a nova relação vazia em disco.
- Cada comando DDL é interpretado no Interpretador de Consultas, e como resultado, a árvore de comandos resultante é uma seqüência de instruções DML e DDL.

Fluxos de Execução

- Linguagem de Manipulação de dados (insert, delete, update e select).
  - **Insert**: não é baseado em nenhum operador algébrico e o módulo executor possui um procedimento específico para tratar a inserção de novas tuplas;
  - **Delete**: não é baseado em nenhum operador algébrico, mas o procedimento específico para ele no módulo executor é precedido por uma operação de **select**.
  - **Update**: tratamento similar ao do delete.
  - **Select**: essência do processamento otimizado.

Fluxos de Execução

- Todos os comandos passam pelo interpretador:
  - Linguagem de Controle de Dados (criação de base e estabelecimento de privilégios e condições para execução de transações):
    - Execução sequencial (sem otimização)
    - Existem procedimentos específicos para executar esses comandos no módulo executor
    - Processamento realizado de maneira direta
  - Linguagem de Definição de Dados (criação e manutenção de relações, índices, gatilhos, etc)
    - Relações do sistema e relações do usuário
    - Não requerem procedimentos específicos, apenas a conversão dos seus comandos em comandos DCL e DML

Fluxos de Execução

- Exemplo:
  - Quando o servidor recebe um comando DDL de criação de uma nova relação (uma relação definida pelo usuário), esse comando é substituído por:
    - Um comando da DML para inserir mais uma tupla na relação das relações;
    - Tantos comandos da DML quanto necessários para inserir/atualizar tuplas na relação de atributos;
    - E um comando DCL para criar fisicamente a nova relação vazia em disco.
- Cada comando DDL é interpretado no Interpretador de Consultas, e como resultado, a árvore de comandos resultante é uma seqüência de instruções DML e DDL.

Fluxos de Execução

- Linguagem de Manipulação de dados (insert, delete, update e select).
  - **Insert**: não é baseado em nenhum operador algébrico e o módulo executor possui um procedimento específico para tratar a inserção de novas tuplas;
  - **Delete**: não é baseado em nenhum operador algébrico, mas o procedimento específico para ele no módulo executor é precedido por uma operação de **select**.
  - **Update**: tratamento similar ao do delete.
  - **Select**: essência do processamento otimizado.

Otimizações que o gerenciador faz

- Não adianta o programador alterar consultas no sentido de querer forçar, por exemplo, uma ordem de junções. Isso é papel do gerenciador e seja lá qual for a opção do programador, o plano de execução que o gerenciador utiliza será baseado nas suas regras de otimização.

- Mas algumas coisas podem ser feitas para influenciar a escolha dos otimizadores.
  - Manutenção das métricas
  - Criação de chaves
  - Alocação de memória
  - Manutenção de regras para dependências funcionais e Triggers

Manutenção de métricas

- Quando uma base de dados é inicializada, as métricas (informações estatística sobre os dados) são colocadas em uma situação *default*.
- A partir de então, as métricas que não são atualizadas por comandos de consulta regulares (depende da tecnologia) ficam sem correção até que os procedimentos de levantamento de métricas seja disparado.
- Isto pode acontecer automaticamente ou por comando específico do usuário (DBA).
  - O levantamento é uma operação cara e enquanto acontece sua execução, o desempenho do sistema cai.
- O DBA deve monitorar o desempenho do sistema (com programas específicos) e na detecção de queda de desempenho, pode solicitar o levamentamento de métricas.
  - Analyse table
  - Analyse index

Comandos no PostgreSQL

`ANALYZE` collects statistics about the contents of tables in the database, and stores the results in the `pg_statistic` system catalog. Subsequently, the query planner uses these statistics to help determine the most efficient execution plans for queries.

With no parameter, `ANALYZE` examines every table in the current database. With a parameter, `ANALYZE` examines only that table. It is further possible to give a list of column names, in which case only the statistics for those columns are collected.

`VACUUM` reclaims storage occupied by dead tuples. In normal PostgreSQL operation, tuples that are deleted or obsoleted by an update are not physically removed from their table; they remain present until a `VACUUM` is done. Therefore it's necessary to do `VACUUM` periodically, especially on frequently-updated tables.

With no parameter, `VACUUM` processes every table in the current database. With a parameter, `VACUUM` processes only that table.

Criação de chaves para os métodos de acesso

- A criação de índice agiliza as consultas, contudo, é uma estrutura de manutenção cara.
  - Sua manutenção ocorre diante de alterações nos dados da chave que é o objeto do índice.
- Tomada de decisão:
  - Verificar quais índices serão utilizados de fato em operações de consulta;
  - Verificar se existem estados com carga de atividade diferente, bem determinados, em que as taxas de consulta de busca e de atualização mudam significativamente;
    - Relações estáticas X relações dinâmicas
      - Relações dinâmicas: base acordada e base dormindo
  - Escolher os métodos de acesso adequados para os tipos de consulta mais freqüentes que utilizarão o índice;
- Solução balanceada no uso da normalilzação!

Alocação de memória

- Cuidar da alocação de hardware para os recursos que exigem leitura e gravação.

- Por exemplo;
  - Todas as consultas geram uma carga de escrita relativamente alta para *logs*, e a seqüência de escritas é apenas escrita seqüêncial no final do arquivo de *log*.
  - Se for alocado um disco especial para o *log,* a cabeça de gravação ficará quase sempre parada sobre o cilindro que está sendo escrito.

Logs!!!

Manutenção de restrições

- O tratamento das restrições de integridade de cada transação pode ser feito de maneira separada da execução de cada comando em si.
- Uso de 3 *threads* por uma consulta*:*
  - O primeiro, *thread da consulta*, é criado quando o comando inicialmente entra em fase de processamento e é submetido para o interpretador de consultas: é responsável pela geração dos dados de resposta e/ou erros de sintaxe, de permissões de acesso e de acesso concorrente. É ele que passa por todos os módulos da arquitetura do processador de consultas;
  - O segundo e o terceiro são criados quando o controlador de concorrência autoriza a execução da consulta.
    - Um para avaliar as condições de validação das dependências de chave estrangeira e as execuções dos *triggers*;
    - Um para ser executado no final da transação, quando são verificadas as regras de consistência da transação;
      - Regras gerais, da aplicação, que verificam condições que devem valer quando a transação termina, mas que podem ficar temporariamente inválidas entre comandos distintos durante o processamento da transação.

Criação de chaves para os métodos de acesso

- A criação de índice agiliza as consultas, contudo, é uma estrutura de manutenção cara.
  - Sua manutenção ocorre diante de alterações nos dados da chave que é o objeto do índice.
- Tomada de decisão:
  - Verificar quais índices serão utilizados de fato em operações de consulta;
  - Verificar se existem estados com carga de atividade diferente, bem determinados, em que as taxas de consulta de busca e de atualização mudam significativamente;
    - Relações estáticas X relações dinâmicas
      - Relações dinâmicas: base acordada e base dormindo
  - Escolher os métodos de acesso adequados para os tipos de consulta mais freqüentes que utilizarão o índice;
- Solução balanceada no uso da normalização!

Alocação de memória

- Cuidar da alocação de hardware para os recursos que exigem leitura e gravação.

- Por exemplo;
  - Todas as consultas geram uma carga de escrita relativamente alta para *logs*, e a seqüência de escritas é apenas escrita seqüêncial no final do arquivo de *log*.
  - Se for alocado um disco especial para o *log,* a cabeça de gravação ficará quase sempre parada sobre o cilindro que está sendo escrito.

Logs!!!

Manutenção de restrições

- O tratamento das restrições de integridade de cada transação pode ser feito de maneira separada da execução de cada comando em si.
- Uso de 3 *threads* por uma consulta*:*
  - O primeiro, *thread da consulta*, é criado quando o comando inicialmente entra em fase de processamento e é submetido para o interpretador de consultas: é responsável pela geração dos dados de resposta e/ou erros de sintaxe, de permissões de acesso e de acesso concorrente. É ele que passa por todos os módulos da arquitetura do processador de consultas;
  - O segundo e o terceiro são criados quando o controlador de concorrência autoriza a execução da consulta.
    - Um para avaliar as condições de validação das dependências de chave estrangeira e as execuções dos *triggers*;
    - Um para ser executado no final da transação, quando são verificadas as regras de consistência da transação;
      - Regras gerais, da aplicação, que verificam condições que devem valer quando a transação termina, mas que podem ficar temporariamente inválidas entre comandos distintos durante o processamento da transação.

Otimizações que o gerenciador não faz

- Tomada de decisões
  - Índices:
    - O gerenciador usa índices existentes, mas não os cria; quem os cria é o desenvolvedor do SBD (a menos do primário);
    - Consulte a tecnologia.
  - Estudar a semântica da aplicação para definir regras e transações;
  - Alocação de área de memória
    - Consulte a tecnologia
- Aplicativo útil:
  - um simulador de carga com um bom conjunto de relatórios estatísticos de operação do sistema (tanto em relação a consultas quanto em relação a execução de transações).

Otimizações que o gerenciador não faz

- Otimização do sistema através da parametrização da instalação
  - Nível de isolamento (pode diminuir concorrência);
  - Declaração de cursores para alteração (permite execução procedimental – com loops – diferente da execução relacional que é seqüêncial);
  - Auto-commit (commit implícito para comandos DDL dentro de transações);
  - Regular o tamanho da Cache (trazer responsabilidades de gerenciamento de memória ao SGBD).

Ajuste no acesso indexado

- Tipos de consultas:
  - **Point Queries:** retornam uma única tupla, e são baseadas em uma condição de identidade sobre uma chave:

```
Select nome
From pessoa
Where cpf = 99999999999;
```

  - **Multipoint Queries:** retornam várias tuplas, e são baseadas em uma condição de identidade sobre um atributo não-chave:

```
Select nome
From pessoa
Where idade = 20;
```

Ajuste no acesso indexado

- **Range queries:** retornam várias tuplas, e são baseadas em uma condiçao de intervalo:

```
Select nome
From pessoa
Where idade > 30;
```

- **Prefix match queries:** retornam várias tuplas, e são baseadas em uma condição de prefixo:

```
Select nome
From pessoa
Where nome like ‘Maria %’;
```

- ...o são também aquelas que fazem comparação de igualdade com um atributo que não tem índice definido, mas é o primeiro (ou os primeiros) atributo de um índice que inclui ao menos um outro atributo (índice composto).

Ajuste no acesso indexado

- **Extremal queries:** retornam uma tupla, mas tem a condição de seleção baseada em algum valor calculado sobre várias outras tuplas da relação:

```
Select nome
From pessoas
Where salario = Max (select salario from pessoa);
```

- **Grouping queries:** retornam várias tuplas, baseadas em valores de agregação sobre valores de seus atributos:

```
Select min(salario), max(salario)
From Pessoas
Group By idade;
```

Ajuste no acesso indexado

- **Sorting queries:** ordenam as tuplas de uma (sub-)relação

```
Select nome, idade
From pessoa
Order by nome;
```

- **Equi-join queries:** realizam uma operação de junção usando o operador de igualdade:

```
Select p.nome, d.nome
From pessoa p, departamento d
Where p.depto = d.id
```

- **Non-equi-join queries:** realizam uma operação de junção usando o operador de desigualdade:

```
Select c.nome, e.nome
From cidade c, estado e
Where c.orcamento > e.orcamento
```

Ajuste no acesso indexado

- Tipos de índices
  - ISAM – Indexed Sequential Access Method
    - Método de acesso seqüencial indexado
    - Pode ser implementado com árvores
  - Hash
    - Fornece-se uma função h, chamada função hash, que, aplicada ao valor do campo de hash de um registro, gere o endereço do bloco de disco no qual o registro está armazenado.
  - Arquivo Invertido
    - Mecanismo orientado à palavra para indexar uma coleção de texto a fim de aumentar a velocidade da tarefa de busca.
    - Um *arquivo invertido* é constituído de uma lista ordenada (ou índice) de palavras-chave (atributos), onde cada palavra-chave tem uma lista de apontadores para os documentos (arquivos) que contêm aquela palavra-chave. Este é o tipo de índice utilizado pela maioria dos sistemas para recuperação em arquivos constituídos de texto.

Ajuste no acesso indexado

- Tipo de consulta X Tipo de índice

Tipo de consulta	ISAM	Hash	Arquivo Invertido
Point query	Média	Média	Média
Multipoint query	Média	Média	Ruim
Prefix match query	Boa	Ruim	Ruim
Extremal query	Boa	Ruim	Ruim
Grouping query	Boa	Boa	Ruim
Sorting query	Média	Ruim	Ruim
Equi-join query	Média	Ruim	Boa
Non-equi-join query	Boa	Ruim	Ruim

Re-escrita de consultas

- Para permitir que índices já existentes sejam usados, evite:

  - **Expressões com atributos indexados:** quando atributos são utilizados em expressões, eles não podem ser acessados usando índices. Por exemplo,

Select * From pessoa Where (salarioanual / 12) < 1000;

  - não pode usar nenhum índice que existe sobre o atributo *salarioanual.* Re-escreva,

Select * from pessoa Where salarioanual < 12000.

Consulte a tecnologia!!!!!

Re-escrita de consultas

- Para permitir que índices já existentes sejam usados, evite:
  - Uso de funções e / ou substrings: o uso de funções ou de regras de edição sobre atributos impedem o uso de índices dos atributos envolvidos. Por exemplo, a consulta

Select * From pessoa Where substr(nome, 1, 5) = ‘Maria’;

  - não pode usar nenhum índice que exista sobre o atributo *nome.*

Consulte a tecnologia!!!!!

Re-escrita de consultas

- Para permitir que índices já existentes sejam usados, evite:
  - Comparação com tipos diferentes de atributos: funções conversoras de tipos impedem o uso de índice;
    - por exemplo, suponha que o atributo salário tenha sido declarado salário decimal (10,2) na relação pessoa.
    - Então a consulta:

```
Select * From pessoa Where salario < 1000;
```

    - precisa converter 1000 para um valor decimal (10,2).
    - reescrevendo a condição para < 1000,00 habilita o uso do índice.

Consulte a tecnologia!!!!!

Re-escrita de consultas

- Comparação com NULL:
  - Note-se que valores NULL não são indexados. Assim o uso da palavra reservada pode causar busca seqüencial.
- Minimizar o uso da cláusula *distinct:*
  - Se os atributos projetados incluem uma chave, ou a semântica da consulta puder garantir que não ocorrerão tuplas repetidas, a cláusula distinct é inócua.
  - No entanto, se for colocada no comando, os otimizadores não a removerão, fazendo com que o resultado seja submetido ao processo de eliminação de tuplas repetidas de qualquer maneira.

Consulte a tecnologia!!!!!

Re-escrita de consultas

- Colocar o máximo de comandos em uma mesma consulta:
  - Deve-se evitar o uso de condições em várias consultas separadas sempre que a consulta puder ser feita em uma consulta apenas.
  - Cada critério de comparação é útil para aumentar a seletividade dos comandos e permitir que o otimizador antecipe as operações de junção e seleção que tenham os critérios mais seletivos.
  - Evite o uso do sub-select

Consulte a tecnologia!!!!!

Re-escrita de consultas

- Outras regras práticas:
  - Condições de operações de junções: sempre que possível, use condições de junção sobre atributos indexados, e de preferência que possuam índices em ambas as relações da junção;
  - Dê preferência a junções com condições sobre atributos numéricos às condições sobre atributos textuais;
- Evite condições disjuntivas:
  - As condições conjuntivas (AND) podem ser otimizadas pelo interpretador, mas as disjuntivas (OR) em geral não são tratadas;

Consulte a tecnologia!!!!!

Mudanças gerais

- Observe que re-escrita de consultas são mudanças locais.

- Mudanças gerais são:
  - Criação de índices;
  - Modificação do esquema das relações
  - Modificação da definição das transações

Consulte a tecnologia!!!!!

Criação de índices

- Devem ser criados índices sobre atributos muito usados em *point* e *multipoint queries*, que retornam poucos dados.
- Devem ser criados índices sobre ambos os atributos de uma condição de junção comparada por igualdade.
- Não é útil criar índices sobre atributos pouco usados em *multipoint queries*, ou sobre atributos discretos com poucos valores mas que tenham muitas tuplas com o mesmo valor.
- Cuidado com um problema potencial no uso dos índices *hash*: excesso de colisões podem originar grandes listas ligadas de ponteiros para tuplas.
- Não devem ser criados índices sobre atributos usados em consultas que costumam retornar mais do que **5 a 10% (?)** da base (tabela) de dados, pois nesse caso a busca seqüencial será menos custosa. **Pesquisar!!**

Consulte a tecnologia!!!!!

Criação de índices

- Não devem ser criados índices em relações pequenas, que possam ser mantidas na memória.

- Nos índices sobre chaves compostas, devem ser colocados primeiro os atributos que:
  - Têm mais chance de serem usados como parte de acesso da relação;
  - Têm menos instâncias para cada valor individual do atributo;
  - Sejam mais curtos (ocupem menos bytes).

- Se existem consultas executadas freqüentemente sobre partes da chave, deve-se criar diversos índices em diferentes ordens, sobre o mesmo conjunto de atributos, um para cada consulta freqüente que use aquela parte da chave.

Consulte a tecnologia!!!!!

Modificação do esquema das relações

- Se mais de uma chave é usada com freqüência para acesso em uma relação, e a relação é pouco atualizada, pode ser interessante duplicar a relação, e criar a chave primária em diferentes chaves em cada cópia.

- Normalizar X desnormalizar uma relação – sempre que possível minimize a quantidade de operações de junção que precisam ocorrer com freqüência em diversas consultas.

Consulte a tecnologia!!!!!

Ajuste no Controle de Concorrência

- **Granularidade de travamento:**
  - Row lock: travamento por predicado: indicado quando a transação vai usar poucas tuplas;
  - Page lock: para páginas shadow;
  - Table lock: para toda a relação.
- **Procure garantir as duas regras seguintes:**
  - Use o menor grau de isolamento que seja seguro. Ei-los (em ordem de menor para maior segurança, e que permitem ao mesmo tempo, de maior para menor concorrência):
  - Non-repeatable read;
  - Read uncommitted;
  - Read committed;
  - Serializable.
  - Elimine todos os locks desnecessários. Eles não são necessários:
    - Quando apenas uma transação está rodando (ex.: carga de dados)
    - Quando existem apenas transações read-only (ex.: geração de relatórios)
    - Sobre toda base ou sobre um assunto em particular.

Consulte a tecnologia!!!!!

SET TRANSACTION transaction_mode [, ...]

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

dirty read

A transaction reads data written by a concurrent uncommitted transaction.

nonrepeatable read

A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

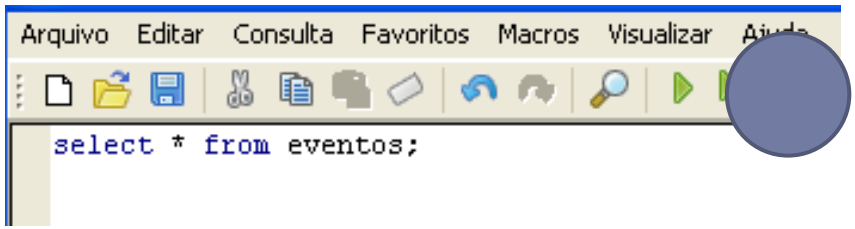
phantom read

A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.

O comando EXPLAIN

```
EXPLAIN SELECT * FROM tenk1 t1, tenk2 t2 WHERE t1.unique1 < 100 AND t1.unique2 = t2.unique2;

                                      QUERY PLAN
-------------------------------------------------------------------------------------
 Nested Loop  (cost=2.37..553.11 rows=106 width=488)
   ->  Bitmap Heap Scan on tenk1 t1  (cost=2.37..232.35 rows=106 width=244)
         Recheck Cond: (unique1 < 100)
         ->  Bitmap Index Scan on tenk1_unique1  (cost=0.00..2.37 rows=106 width=0)
               Index Cond: (unique1 < 100)
   ->  Index Scan using tenk2_unique2 on tenk2 t2  (cost=0.00..3.01 rows=1 width=244)
         Index Cond: (t2.unique2 = t1.unique2)
```



Considerações sobre o PostGreSQL

http://pgdocptbr.sourceforge.net/pg80/overview.html#QUERY-PATH

41.1. O caminho do comando

Nesta seção é dada uma visão geral resumida dos estágios pelos quais o comando tem que passar para chegar ao resultado.

1. Deve ser estabelecida uma conexão entre o programa aplicativo e o servidor PostgreSQL. O programa aplicativo transmite um comando para o servidor, e aguarda para receber de volta os resultados transmitidos pelo servidor.

2. O *estágio de análise* verifica o comando transmitido pelo programa aplicativo com relação à correção da sintaxe, e cria a *árvore de comando*.

3. O *sistema de reescrita* recebe a árvore de comando criada pelo estágio de análise, e procura por alguma *regra* (armazenada nos *catálogos do sistema*) a ser aplicada na árvore de comando. Realiza as transformações especificadas no *corpo das regras*.

   Uma das aplicações do sistema de reescrita é a criação de *visões*. Sempre que é executado um comando em uma visão (ou seja, *uma tabela virtual*), o sistema de reescrita reescreve o comando do usuário como um comando acessando as *tabelas base* especificadas na *definição da visão*, em vez da visão.

4. O *planejador/otimizador* recebe a árvore de comando (reescrita), e cria o *plano de comando* que será a entrada do *executor*.

   Isto é feito criando primeiro todos os *caminhos* possíveis que levam ao mesmo resultado. Por exemplo, se existe um índice em uma relação a ser varrido, existem dois caminhos para a varredura. Uma possibilidade é uma varredura seqüencial simples, e a outra possibilidade é utilizar o índice. Em seguida é estimado o custo de execução de cada um dos caminhos, e escolhido o mais barato. O caminho mais barato é expandido em um plano completo para que o executor possa utilizá-lo.

5. O executor caminha recursivamente através da *árvore do plano*, e traz as linhas no caminho representado pelo plano. O executor faz uso do *sistema de armazenamento* ao varrer as relações, realiza *classificações* e *junções*, avalia as *qualificações* e, por fim, envia de volta as linhas derivadas.

41.5. Planejador/Otimizador

A tarefa do *planejador/otimizador* é criar um plano de execução ótimo. Um dado comando SQL (e, portanto, uma árvore de comando) pode, na verdade, ser executada de várias maneiras diferentes, cada uma das quais produzindo o mesmo conjunto de resultados. Se for computacionalmente praticável, o otimizador de comandos examina cada um dos planos de execução possíveis para, no fim, selecionar o plano de execução que espera ser o mais rápido.

> **Nota:** Em algumas situações, o exame de todas as formas pelas quais um comando pode ser executado leva a um consumo excessivo de tempo e de espaço em memória. Em particular, estas situações ocorrem quando se executa comandos que envolvem um grande número de operações de junção. Para ser possível determinar um plano de comando razoável (não o ótimo), em um espaço de tempo razoável, o PostgreSQL utiliza o *Genetic Query Optimizer*.

Na verdade, o procedimento de procura do planejador trabalha com estruturas de dados chamadas de *caminhos* (paths), que são simplesmente representações reduzidas dos planos, contendo somente as informações necessárias para o planejador tomar suas decisões. Após ser determinado o caminho mais barato, é construída a *árvore de plano* pronta para ser passada para o executor. Esta árvore representa o plano de execução desejado no nível de detalhamento suficiente para o executor processá-la. No restante desta seção será ignorada a distinção entre caminhos e planos.

http://pgdocptbr.sourceforge.net/pg80/planner-optimizer.html

Capítulo 47. Genetic Query Optimizer

Sumário

Autor: Written by Martin Utesch (<utesch@aut.tu-freiberg.de>) for the Institute of Automatic Control at the University of Mining and Technology in Freiberg, Germany.

http://pgdocptbr.sourceforge.net/pg80/geqo.html

Capítulo 41. Visão geral da estrutura interna do PostgreSQL

Sumário

Autor: Este capítulo se originou como parte da Tese de Mestrado de Stefan Simkovics, preparada na Universidade de Tecnologia de Viena, sob a direção de O.Univ.Prof.Dr. Georg Gottlob e Univ.Ass.Mag. Katrin Seyr, *Enhancement of the ANSI SQL Implementation of PostgreSQL*.

http://pgdocptbr.sourceforge.net/pg80/overview.html

Bibliografia

- Estes slides estão baseados nos livros:
  - Elmasri, R. & Navathe, S. B.. Sistemas de Banco de Dados. São Paulo: Addison Wesley, 2005.
  - Silberschatz, A., Korth, H. F. & Sudarshan, S.. Sistemas de Banco de Dados. São Paulo: Makron Books, 2006.
- E documentação online do SGBD PostgreSQL:
  - http://pgdocptbr.sourceforge.net/pg80/preface.html
- Sugestão de consulta:
  - http://troels.arvin.dk/db/rdbms/#features-views
  - http://www.postgresql.org.br/Documenta%C3%A7%C3%A3o
  - http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems (curiosidades)