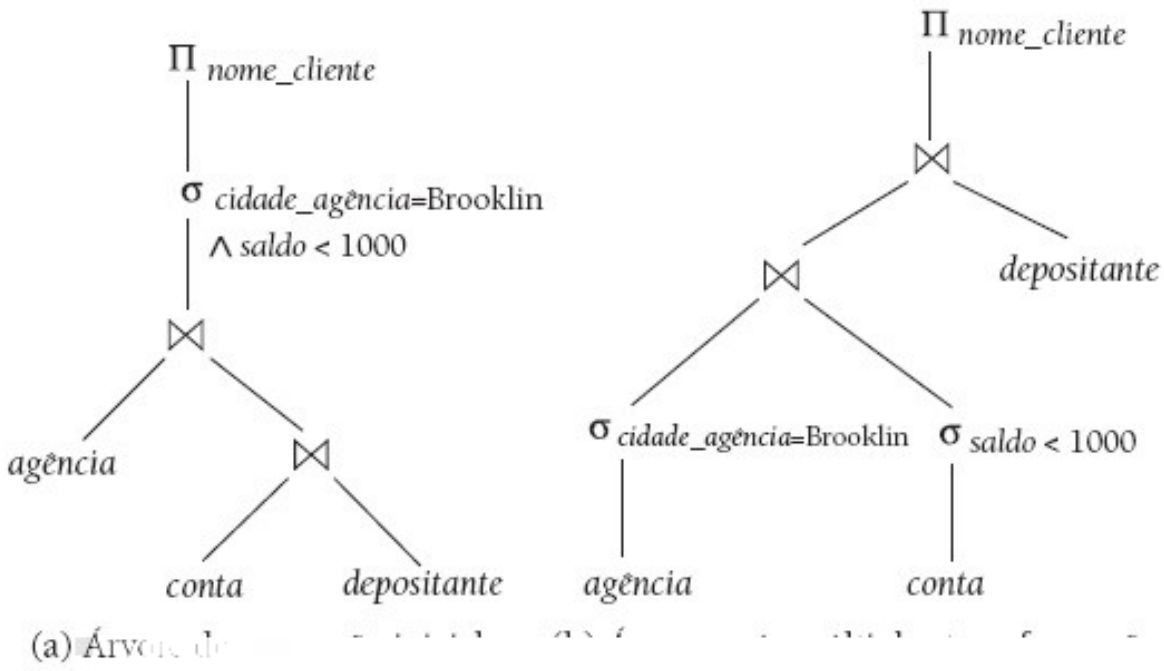
# Otimização de consultas

# Múltiplas transformações (cont.)




(a) Árvore de

# Exemplo de ordenação de junção

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

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

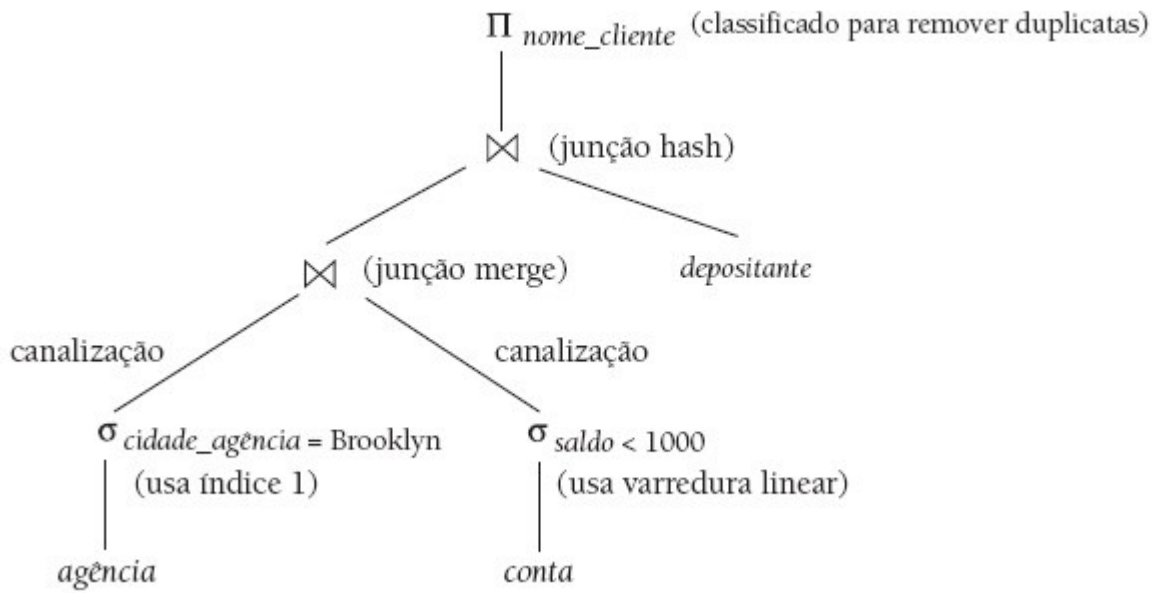
Se $r_2 \bowtie r_3$ for muito grande e $r_1 \bowtie r_2$ for pequeno, escolhemos

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

de modo que calculamos e armazenamos uma relação temporária menor.

# Plano de avaliação

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



# Escolha dos planos de avaliação

Necessário considerar a interação das técnicas de avaliação quando escolher planos de avaliação. A escolha do algoritmo mais barato para cada operação independentemente pode não gerar o melhor algoritmo geral. Por exemplo:

junção merge pode ser mais dispendiosa que a junção de hash, mas pode oferecer uma saída classificada que reduz o custo para um nível de agregação externo.

a junção de loop aninhado oferece oportunidade para pipelining

Otimizadores de consulta práticos incorporam elementos das duas técnicas gerais:

1. Pesquisar todos os planos e escolher o melhor plano com base no custo.
2. Usar heurística para escolher um plano.

# Otimização baseada em custo

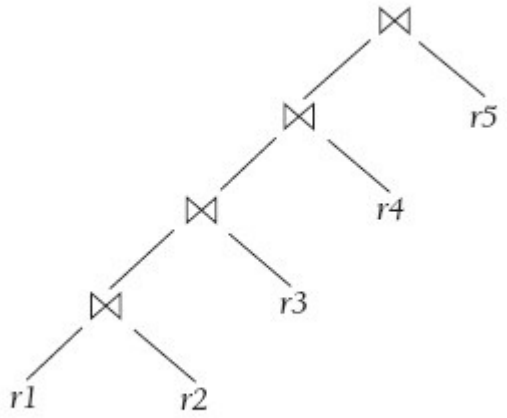
Considere encontrar a melhor ordem de junção para $r_1 \quad r_2 \quad \ldots \; r_n$

Existem $(2(n-1))!/(n-1)!$ ordens de junção diferentes para a expressão acima.
Com $n = 7$, o número é 665280, com $n = 10$, o número é maior que 176 bilhões!

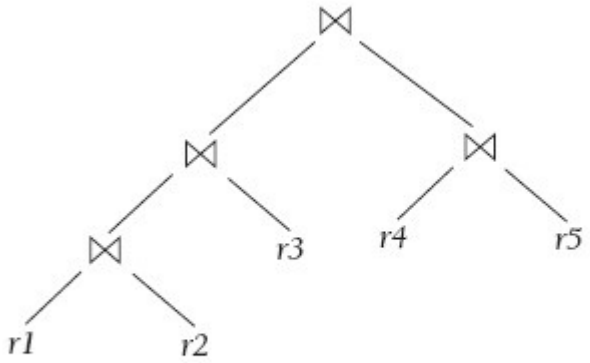
Não é preciso gera.,ene naiorden de junmsor ordem de junçãm é,elhoa.

# Árvores de junção esquerda profunda

**Em *árvores de junção profunda esquerda*, a entrada do lado direito (relação interna) para cada junção é uma relação, e não o resultado de uma junção intermediária.**




(a) Árvore de junção esquerda profunda




(b) Árvore de junção esquerda não profunda

# Custo da otimização

Com o tempo de programação dinâmico, a complexidade da otimização com árvores de muitas folhas é $O(3^n)$.

Com $n = 10$, esse número é 59000 em vez de 176 bilhões!

A complexidade de espaço é $O(2^n)$

Para encontrar a melhor árvore de junção profunda esquerda para um conjunto de $n$ relações:

Considere $n$ a1M□tivas com uma relação como a entrada do lado direito e as outras relações como a entrada do lado esquerdo.

Usando a ordem de junção de menor custo (ca1julada e armazenada recursivamente) para cada alt□tiva no lado esquerdo, escolha a mais barata das $n$ a1M□tivas.

Se apenas árvores profundas esquerdas forem consideradas, a complexidade de tempo para encontrar a melhor ordem de junção é $O(n\ 2^n)$

A complexidade de espaço permanece em $O(2^n)$

A otimização baseada em custo é dispendiosa, mas compensa para consu1Ms□ sobre datasets grandes

- consultt□picas possuem n pequeno, geralmente < 10

# Otimização heurística

A otimização baseada em custo é dispendiosa, mesmo com a programação dinâmica.

Os sistemas podem usar *heurísticas* para reduzir o número de escolhas que precisam ser feitas com base no custo.

A otimização heurística transforma a árvore de consulta usando um conjunto de regras que normalmente (mas não em todos os casos) melhoram o desempenho da execução:

- Realizar a seleção cedo (reduz o número de tuplas)
- Realizar a projeção cedo (reduz o número de atributos)
- Realizar as operações de seleção e junção mais restritivas antes de outras operações semelhantes.

Alguns sistemas utilizam apenas heurísticas, outros combinam heurísticas com otimização parcial baseada em custo.

# Estrutura dos otimizadores de consulta (cont.)

Alguns otimizadores de consulta integram a seleção heurística e a geração de planos de acesso alternativos.

Mesmo com o uso da heurística, a otimização de consulta baseada em custo impõe uma sobrecarga substancial.

Essa despesa normalmente é mais do que compensada pelas economias no tempo de execução da consulta, particularmente reduzindo o número de acessos ao disco.

Detalhes da SQL complicam a otimização da consulta

- Por exemplo, subconsultas aninhadas

# Otimizando subconsultas aninhadas**

**A SQL por conceito trata as subconsultas aninhadas na cláusula where como funções que apanham parâmetros e retornam um único valor ou conjunto de valores**

**Parâmetros são variáveis da consulta de nível externo que são usadas na subconsulta aninhada; essas variáveis são denominadas variáveis de correlação**

**Por exemplo:**

```
select nome-cliente
    from tomador
    where exists (select *
                  from depositante
                  where depositante.nome-cliente =
                                        tomador.nome-cliente)
```

**Por conceito, a subconsulta aninhada é executada uma vez para cada tupla no produto cruzado gerado pela cláusula from de nível externo**

**Essa avaliação é chamada de avaliação correlacionada**

# Otimizando subconsultas aninhadas (cont.)

- **A avaliação correlacionada pode ser muito ineficaz, pois**
  - **uma grande quantidade de chamadas pode ser feita à consulta aninhada**
  - **pode haver E/S aleatória desnecessária como resultado**
  - **otimizadores SQL tentam transformar subconsultas aninhadas a junções onde for possível, permitindo o uso de técnicas de junção eficientes**
- **Por exemplo: consulta aninhada mais cedo pode ser reescrita como**

select ***nome-cliente***
from ***tomador, depositante***
where ***depositante.nome-cliente = tomador.nome-cliente***

- **Nota: essa consulta não trata corretamente de duplicatas, mas pode ser modificada para fazer isso, como veremos**
- **Em geral, não é possível/simples mover a cláusula from da subconsulta aninhada inteira para a cláusula from da consulta de nível externo**
  - **Uma relação temporária é criada em vez disso e usada no corpo da consulta de nível externo**

# Otimizando subconsultas aninhadas (cont.)

**Em geral, consultas SQL do formato a seguir podem ser reescritas conforme mostramos**

**Reescreva:** select …
from $L_1$
where $P_1$ and exists (select *
from $L_2$
where $P_2$)

**Para:** create table $t_1$ **as**
select distinct $V$
from $L_2$
where $P_2^1$
select …
from $L_1$, $t_1$
where $P_1$ and $P_2^2$

**$P_2^1$ contém predicados em $P_2$ que não envolvem quaisquer variáveis de correlação**

**$P_2^2$ reintroduz predicados envolvendo variáveis de correlação, com relações renomeadas corretamente**

**V contém todos os atributos usados nos predicados com variáveis de correlação**

# Otimizando subconsultas aninhadas (cont.)

**Em nosso exemplo, a consulta aninhada original seria transformada para**

create table $t_1$ as
    select distinct ***nome-cliente***
    from ***depositante***
    select ***nome-cliente***
from ***tomador***, $t_1$
 where $t_1$.***nome-cliente*** = ***tomador.nome-cliente***

**O processo de substituir uma consulta aninhada por uma consulta com uma junção (possivelmente com uma relação temporária) é chamado de descorrelação.**

**A descorrelação é mais complicada quando**

- **a subconsulta aninhada usa agregação, ou**
- **quando o resultado da subconsulta aninhada é usado para testar igualdade, ou**
- **quando a condição vinculando a subconsulta aninhada à outra consulta é** not exists**,**