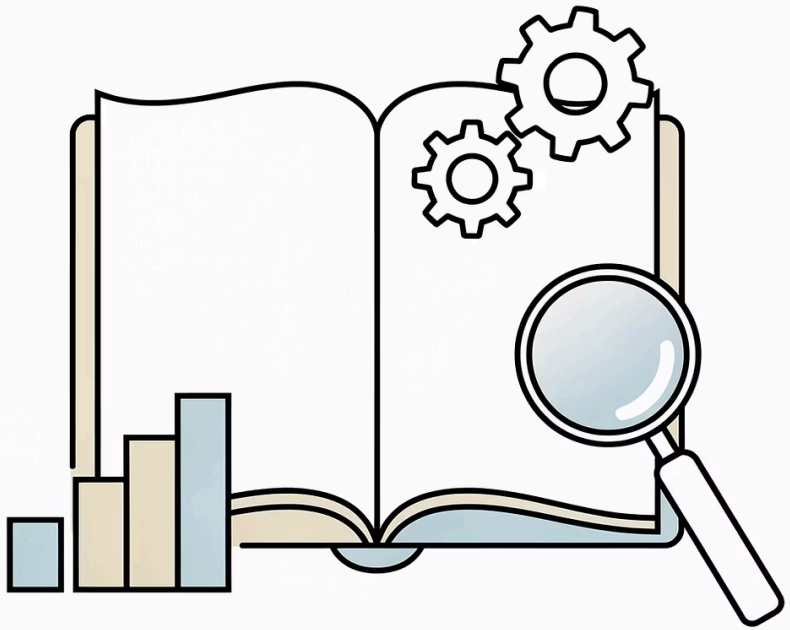


CEFET/RJ

Otimização de Consultas em Bancos de Dados

Uma exploração técnica das estratégias e técnicas para melhorar o desempenho de consultas em sistemas de gerenciamento de banco de dados relacionais.

Eduardo Ogasawara

eduardo.ogasawara@cefet-rj.br
https://eic.cefet-rj.br/~eogasawara

Introdução à Otimização de Consultas

A otimização de consultas é um processo fundamental em sistemas de gerenciamento de banco de dados (SGBDs) que visa melhorar o desempenho e a eficiência da execução de consultas SQL. O otimizador de consultas analisa diferentes estratégias de execução e seleciona aquela que minimiza o custo computacional.

Este processo envolve duas abordagens principais: a otimização lógica, que trabalha com transformações algébricas, e a otimização física, que considera os recursos reais do sistema. Juntas, essas técnicas garantem que as consultas sejam executadas da forma mais eficiente possível, reduzindo tempo de resposta e consumo de recursos.

Otimização Lógica e Otimização Física

Otimização Lógica

- Baseada em regras de equivalência da álgebra relacional
- Transforma expressões algébricas em formas equivalentes mais eficientes
- Opera independentemente de algoritmos específicos
- Não considera custos físicos de implementação
- Foco em reestruturação de consultas

Otimização Física

- Escolhe algoritmos concretos de avaliação
- Considera recursos físicos como índices disponíveis
- Avalia métodos de junção e ordenação
- Baseia decisões em custos estimados reais
- Adapta-se à infraestrutura do sistema

Estas duas abordagens funcionam como etapas complementares e sequenciais no processo completo de otimização. A otimização lógica prepara o terreno com transformações algébricas, enquanto a otimização física implementa as escolhas práticas baseadas nos recursos disponíveis e custos mensuráveis.

Árvores de Processamento de Consultas

As árvores de processamento de consultas são essenciais para visualizar e otimizar a execução de comandos em um sistema de banco de dados.

Definição

Representação formal da execução de consultas, onde cada nó é uma operação da álgebra relacional e cada folha uma relação base do banco de dados.

Estrutura

Folhas representam tabelas de entrada, nós internos correspondem a operações (seleção, projeção, junção). A raiz representa o resultado final.

Ordem de Execução

A avaliação ocorre de baixo para cima, combinando resultados intermediários. Árvores logicamente equivalentes podem ter custos computacionais distintos.

Papel na Otimização

Base para a otimização lógica, permitindo a aplicação de regras de equivalência da álgebra relacional para reorganizar operações e reduzir o custo de execução.

EXEMPLO PRÁTICO

Árvore de Processamento de Consulta

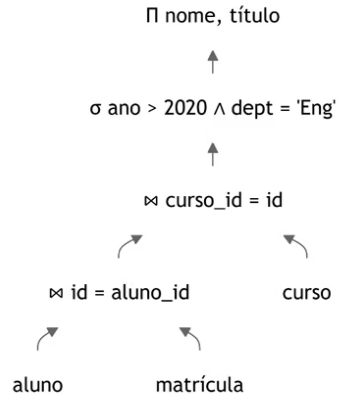
Vamos explorar como uma consulta SQL é transformada em uma árvore de processamento usando álgebra relacional, com base em um esquema acadêmico simples.

Esquema Relacional

- aluno(id, nome)
- matrícula(aluno_id, curso_id, ano)
- curso(id, título, dept)

Objetivo da Consulta

Obter os nomes dos alunos e os títulos dos cursos do departamento de Engenharia, iniciados após o ano de 2020.



Expressão em Álgebra Relacional

$$\pi_{nome,titulo}\left(\sigma_{ano>2020 \wedge dept='Eng'}\left(\left(aluno \bowtie_{id=aluno\_id} matricula\right) \bowtie_{curso\_id=id} curso\right)\right)$$

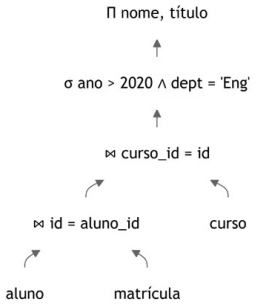
OTIMIZAÇÃO

Transformação da Árvore: Push-Down de Seleções

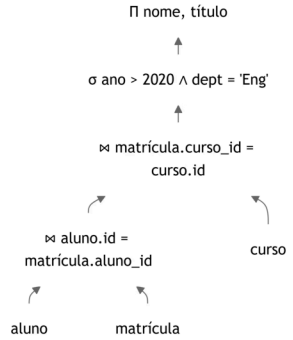
A otimização de consultas transforma a árvore inicial aplicando técnicas como push-down de seleções, que empurra os filtros para mais perto das relações base, reduzindo o tamanho dos resultados intermediários.

```
SELECT a.nome, c.título
FROM aluno AS a
JOIN matrícula AS m ON a.id = m.aluno_id
JOIN curso AS c ON m.curso_id = c.id
WHERE m.ano > 2020 AND c.dept = 'Eng';
```

Antes: Filtro aplicado tarde, após todas as junções serem executadas.



Depois: Predicados decompostos e empurrados para as relações apropriadas antes das junções.



Álgebra Relacional Otimizada: $\pi_{nome,título}\Big(\big(aluno \bowtie_{id=aluno\_id} \sigma_{ano>2020}(matrícula)\big) \bowtie_{curso\_id=id} \sigma_{dept='Eng'}(curso)\Big)$

Observação: A árvore otimizada é equivalente à original, mas tende a ser mais eficiente porque reduz o tamanho dos resultados intermediários antes das junções, melhorando significativamente o desempenho da consulta.

Otimização Baseada em Custo

A otimização baseada em custo é uma abordagem sofisticada que avalia numericamente diferentes planos de execução para selecionar o mais eficiente. Este processo sistemático combina teoria e prática para decisões fundamentadas.

01

Geração de Expressões Equivalentes

Aplicação de regras de equivalência da álgebra relacional para criar múltiplas versões logicamente equivalentes da consulta original.

02

Anotação de Expressões

Cada expressão equivalente é anotada com informações sobre algoritmos de acesso, métodos de junção e outras escolhas físicas, gerando planos de consulta alternativos.

03

Estimativa de Custos

Cálculo do custo estimado de cada plano alternativo utilizando estatísticas do catálogo e modelos de custo para operações de I/O, CPU e memória.

04

Seleção do Plano Ótimo

Escolha do plano com menor custo estimado para execução, equilibrando precisão das estimativas com tempo de otimização.

Estatísticas do Catálogo de Banco de Dados

O otimizador de consultas depende fortemente de estatísticas armazenadas no catálogo do sistema para tomar decisões informadas. Estas estatísticas fornecem uma visão quantitativa sobre os dados e sua distribuição, permitindo estimativas precisas de custo.

Cardinalidade das Relações

Número total de tuplas em cada relação do banco de dados. Esta métrica é fundamental para estimar o tamanho dos resultados intermediários e finais.

Blocos de Armazenamento

Quantidade de blocos físicos ocupados por cada relação no disco. Crucial para estimar custos de I/O, que frequentemente dominam o tempo de execução.

Valores Distintos

Número de valores únicos para cada atributo. Essencial para estimar a seletividade de predicados e a eficácia de índices.

Aplicações das estatísticas: O otimizador utiliza essas informações para estimar o custo de cada operação, comparar planos alternativos e fazer escolhas fundamentadas sobre ordem de junções, uso de índices e métodos de acesso. Estatísticas desatualizadas podem levar a decisões subótimas.

ÁLGEBRA RELACIONAL

Transformação de Expressões Relacionais

Equivalência em Conjuntos

Duas expressões da álgebra relacional são equivalentes se, para qualquer instância válida do banco de dados, ambas geram o mesmo conjunto de tuplas. A ordem das tuplas é considerada irrelevante nesta definição.

Equivalência em Multiconjuntos

Em SQL, que trabalha com multiconjuntos (bags), duas expressões são equivalentes se gerarem o mesmo multiconjunto de tuplas, preservando duplicatas. Esta é uma distinção importante para otimização prática.

> Uma regra de equivalência estabelece que expressões de duas formas distintas são equivalentes, permitindo substituir uma pela outra durante o processo de otimização sem alterar o resultado da consulta.

Estas regras formam a base teórica para todas as transformações realizadas pelo otimizador lógico, garantindo que as modificações preservem a semântica original da consulta enquanto potencialmente melhoram seu desempenho.

FUNDAMENTOS

Regras de Equivalência

As regras de equivalência da álgebra relacional são o alicerce da otimização lógica de consultas. Cada regra define uma transformação que preserva o resultado enquanto pode alterar significativamente o custo de execução.

Comutatividade

Operações como $\bowtie$ (junção) e $\cup$ (união) podem ter seus operandos trocados de ordem sem afetar o resultado. Útil para reordenar operações buscando eficiência.

Associatividade

Permite reagrupar operações em cadeia, mudando qual operação é executada primeiro. Fundamental para otimização de múltiplas $\bowtie$ (junções).

Distributividade

Algumas operações podem ser distribuídas sobre outras, como σ (seleção) sobre $\bowtie$ (junção), permitindo filtrar dados mais cedo no plano de execução.

Estas regras, quando aplicadas estrategicamente, permitem ao otimizador explorar um espaço de busca de planos alternativos, identificando transformações que reduzem o volume de dados processados em etapas intermediárias.

Regras de Equivalência: Propriedades Fundamentais

Comutatividade de Junção Natural

$$R \bowtie S \equiv S \bowtie R$$

A ordem dos operandos em uma junção natural pode ser invertida sem alterar o resultado, permitindo escolher a ordem mais eficiente baseada em estatísticas.

Comutatividade de Junção Theta

$$R \bowtie_\theta S \equiv S \bowtie_\theta R$$

Similar à junção natural, junções com predicados arbitrários também são comutativas, desde que o predicado seja ajustado adequadamente.

Associatividade de Junções

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

Fundamental para otimização de consultas com múltiplas junções, permite explorar diferentes ordens de execução que podem variar drasticamente em custo.

Regras de Equivalência: Seleção e Projeção

Decomposição de Seleções Conjuntivas

$$\sigma_{\theta_1 \wedge \theta_2}(R) \equiv \sigma_{\theta_1}(\sigma_{\theta_2}(R))$$

Uma seleção com condição AND pode ser decomposta em seleções sequenciais, permitindo aplicar índices ou reordenar predicados.

Comutatividade de Seleções

$$\sigma_{\theta_1}(\sigma_{\theta_2}(R)) \equiv \sigma_{\theta_2}(\sigma_{\theta_1}(R))$$

Seleções sucessivas podem ser aplicadas em qualquer ordem, útil para avaliar primeiro predicados mais seletivos.

Decomposição de Projeções

Apenas a última projeção em uma sequência é necessária, desde que contenha todos os atributos requeridos.

Permite eliminar projeções intermediárias redundantes.

Estas regras são especialmente valiosas para a técnica de *push-down* de seleções e projeções, reduzindo o volume de dados que trafega entre operadores.

Representação das Regras de Equivalência

PROPRIEDADES ALGÉBRICAS

Comutatividade

A propriedade comutativa permite trocar a ordem dos operandos sem alterar o resultado. Esta propriedade é válida para:

- Junção natural ($\bowtie$)
- Junção theta ($\bowtie_\theta$)
- União ($\cup$)
- Interseção ($\cap$)

Associatividade

A propriedade associativa permite reagrupar operações em cadeia, alterando qual par é avaliado primeiro. Crucial para:

- Múltiplas junções em consultas complexas
- Operações de conjunto encadeadas
- Otimização de ordem de execução

Estas propriedades matemáticas fundamentais da álgebra relacional fornecem a flexibilidade necessária para o otimizador explorar diferentes arranjos de operações, mantendo sempre a correção semântica da consulta original.

Push-Down de σ

Uma das técnicas mais poderosas de otimização é o *push-down* de operações de σ, movendo filtros para mais perto das relações-base. Isso reduz drasticamente o volume de dados processados nas etapas subsequentes.

1

σ após $\bowtie$

Forma inicial onde a seleção ocorre após a junção completar, processando um volume maior de dados intermediários.

2

Decomposição do Predicado

Análise do predicado de seleção para identificar componentes que referenciam apenas uma relação de cada vez.

3

σ Antes da $\bowtie$

Predicados que envolvem apenas atributos de uma relação são empurrados para antes da junção, filtrando dados antecipadamente.

> **Benefício chave:** Ao filtrar tuplas antes da $\bowtie$, reduzimos significativamente o produto $\times$ parcial que precisa ser computado, resultando em economia de I/O, memória e tempo de CPU.

Regras de Equivalência: Distribuição de Seleção

1 Seleção sobre Junção Natural

Quando o predicado de seleção envolve apenas atributos de uma das relações sendo unidas, a seleção pode ser aplicada antes da junção: $\sigma_\theta(R \bowtie S) \equiv \sigma_\theta(R) \bowtie S$ (se θ usa apenas atributos de R)

2 Seleção com Predicados Compostos

Quando o predicado é uma conjunção (AND) onde θ_1 envolve apenas R e θ_2 envolve apenas S: $\sigma_{\theta_1 \wedge \theta_2}(R \bowtie S) \equiv \sigma_{\theta_1}(R) \bowtie \sigma_{\theta_2}(S)$

3 Seleção sobre Junção Theta

Similar à junção natural, mas preservando o predicado de junção original: $\sigma_{\theta_1}(R \bowtie_{\theta_2} S) \equiv (\sigma_{\theta_1}(R)) \bowtie_{\theta_2} S$

Estas regras são fundamentais para a estratégia de otimização conhecida como *selection pushing* ou *push-down de seleção*, uma das transformações mais impactantes para o desempenho de consultas.

Regras de Equivalência: Projeção e Conjuntos

Distribuição de Projeção sobre Junção

A projeção pode ser distribuída sobre uma junção natural quando os atributos da junção estão incluídos nos atributos projetados:

$$\pi_{L_1 \cup L_2}(R \bowtie S) \equiv \pi_{L_1}(R) \bowtie \pi_{L_2}(S)$$

Onde L_1 e L_2 são listas de atributos de R e S respectivamente, e incluem os atributos de junção.

Operações de Conjunto

Seleção distribui sobre união, interseção e diferença:

- $\sigma_\theta(R \cup S) \equiv \sigma_\theta(R) \cup \sigma_\theta(S)$
- $\sigma_\theta(R \cap S) \equiv \sigma_\theta(R) \cap \sigma_\theta(S)$
- $\sigma_\theta(R - S) \equiv \sigma_\theta(R) - \sigma_\theta(S)$

Projeção também distribui sobre união: $\pi_L(R \cup S) \equiv \pi_L(R) \cup \pi_L(S)$

Estas regras permitem mover projeções e seleções por meio de operações de conjunto, possibilitando redução de dados em subconsultas e operações sobre visões.

Regras de Equivalência: Agregação e Agrupamento

Operações de agregação e agrupamento requerem tratamento especial nas transformações de consulta, pois alteram a estrutura dos dados de forma mais significativa que operações básicas.

1

Push-Down de Seleção com Agregação

Seleções que envolvem apenas atributos de agrupamento podem ser aplicadas antes da agregação: $\sigma_\theta(\Gamma_{G,F}(R)) \equiv \Gamma_{G,F}(\sigma_\theta(R))$ quando θ referencia apenas atributos em Γ.

2

Decomposição de Agregações

Algumas funções de agregação podem ser computadas em etapas. Por exemplo, COUNT e SUM podem ser parcialmente agregados e depois combinados, útil para processamento paralelo.

3

Agregação com Junção

Sob certas condições, agregações podem ser empurradas por meio de junções, especialmente quando a junção é muitos-para-um em relação ao agrupamento.

A cláusula HAVING em SQL aplica condições após a agregação, enquanto WHERE filtra antes. O otimizador pode transformar condições HAVING elegíveis em predicados WHERE para melhor desempenho.

Regras de Equivalência: Subconsultas e Ordenação

Descorrelação de Subconsultas

Subconsultas correlacionadas podem frequentemente ser transformadas em junções equivalentes, eliminando a necessidade de avaliar a subconsulta para cada tupla externa. Esta transformação, chamada de *decorrelation*, pode melhorar drasticamente o desempenho.

Exemplo: EXISTS e IN podem ser convertidos em semi-junções.

Eliminação de Subconsultas Redundantes

Subconsultas que retornam resultados constantes ou que são semanticamente equivalentes a expressões mais simples podem ser eliminadas ou simplificadas pelo otimizador.

Operações de Ordenação

ORDER BY pode ser postergado até o final do plano de execução, mas algumas vezes pode ser benéfico usar ordenação intermediária quando há índices ordenados disponíveis ou quando operações como junção sort-merge podem aproveitar dados já ordenados.

Exemplo de Transformação

Considere uma consulta sobre um banco de dados acadêmico que busca informações sobre professores e disciplinas de um departamento específico. Vejamos como o otimizador transforma o plano de execução.

Plano Inicial

$$\sigma_{dept=\text{'Computação'}}(\text{professor} \bowtie \text{ensina} \bowtie \text{disciplina})$$

Executa a junção completa de três tabelas antes de aplicar o filtro, processando um grande volume intermediário de dados desnecessários.

Plano Otimizado

$$(\sigma_{dept=\text{'Computação'}}(\text{professor})) \bowtie \text{ensina} \bowtie \text{disciplina}$$

Aplica o filtro antes da primeira junção (*push-down*), reduzindo drasticamente o número de tuplas envolvidas nas operações subsequentes.

Ganho de desempenho: Se o departamento de Computação tiver 10% dos professores, reduzimos o volume da primeira junção em 90%, economizando I/O, memória e tempo de CPU.

Exemplo com Múltiplas Transformações

Em consultas complexas, múltiplas regras de equivalência são aplicadas em sequência para alcançar o plano ótimo. Vamos analisar um exemplo com junções, seleções e projeções.

01

Consulta Original

$\pi_{\text{nome,título}}(\sigma_{\text{ano}>2020 \wedge \text{dept}=\text{"Eng"}}(\text{aluno} \bowtie \text{matrícula} \bowtie \text{curso}))$

Forma direta da consulta SQL: buscar nomes de alunos e títulos de cursos do departamento de Engenharia iniciados após 2020.

02

Decomposição de Seleção

$\pi_{\text{nome,título}}(\sigma_{\text{ano}>2020}(\sigma_{\text{dept}=\text{"Eng"}}(\text{aluno} \bowtie \text{matrícula} \bowtie \text{curso})))$

Separamos o predicado conjuntivo em duas seleções independentes.

03

Push-Down de Seleções

$\pi_{\text{nome,título}}(\text{aluno} \bowtie \sigma_{\text{ano}>2020}(\text{matrícula}) \bowtie \sigma_{\text{dept}=\text{"Eng"}}(\text{curso}))$

Movemos cada seleção para a relação apropriada, filtrando dados antes das junções.

04

Push-Down de Projeção

$\pi_{\text{nome,título}}(\pi_{\text{id,nome}}(\text{aluno}) \bowtie \pi_{\text{aluno\_id,curso\_id,ano}}(\text{matrícula}) \bowtie \pi_{\text{id,título,dept}}(\text{curso}))$

Projetamos apenas atributos necessários em cada relação, incluindo chaves de junção.

> **Resultado:** Um plano otimizado que processa muito menos dados em cada etapa, aproveitando seleções antecipadas e projeções que reduzem largura e altura das relações intermediárias. O custo de execução pode ser reduzido em ordens de magnitude.

OTIMIZAÇÃO DE CONSULTAS

Múltiplas Transformações em Árvores de Expressão

A otimização de consultas em bancos de dados relacionais frequentemente envolve múltiplas transformações sucessivas na árvore de expressão inicial. Esse processo iterativo busca encontrar uma representação equivalente mais eficiente da consulta original, aplicando regras de equivalência algébrica que preservam o resultado final mas alteram significativamente o custo de execução.

Cada transformação pode reorganizar operações, reordenar junções, ou reposicionar seleções e projeções na árvore. O objetivo é minimizar o custo total de processamento, considerando fatores como número de acessos a disco, tamanho de resultados intermediários e utilização de memória.

Exemplo de Operação de Projeção (π)

Antes da Otimização

A operação de projeção (π) pode aparecer em diferentes pontos da árvore de consulta. Quando posicionada tardiamente, processa mais atributos do que o necessário nas operações intermediárias.

$$\pi_{nome,salario}(\sigma_{departamento='TI'}(\text{funcionarios}))$$

Após a Otimização

Aplicando a regra de "push down" de projeção (π), eliminamos atributos desnecessários o mais cedo possível, reduzindo o volume de dados transferidos entre operações e melhorando o desempenho geral.

A projeção antecipada (π) reduz significativamente o tamanho dos resultados intermediários, especialmente em tabelas com muitos atributos.

Exemplo de Ordenação de Junção

A ordenação de junção (merge join) é uma técnica eficiente que aproveita dados previamente ordenados ou ordena explicitamente as relações de entrada antes de realizar a junção. Este método é particularmente vantajoso quando as relações já possuem índices ordenados ou quando o resultado final precisa estar ordenado.

01

Ordenação das Relações

Ambas as relações são ordenadas pelo atributo de junção, seja aproveitando índices existentes ou realizando ordenação externa.

02

Varredura Simultânea

Ponteiros avançam simultaneamente pelas duas relações ordenadas, comparando valores e identificando correspondências.

03

Geração do Resultado

Tuplas correspondentes são combinadas e adicionadas ao resultado, que já emerge naturalmente ordenado.

Exemplo de Ordenação de Junção (continuação)

Vantagens da Merge Join

- Complexidade linear O(n+m) após ordenação
- Resultado naturalmente ordenado
- Eficiente para grandes volumes
- Aproveitamento de índices existentes

Considerações de Desempenho

O custo total inclui a ordenação inicial das relações, que pode ser significativo. No entanto, se as relações já estão ordenadas ou se o resultado ordenado é necessário para operações subsequentes, a merge join torna-se altamente competitiva.

A escolha entre merge join, hash join ou nested loop depende das características específicas dos dados e do plano de execução completo.

Enumeração de Expressões Equivalentes

Os otimizadores de consulta utilizam um conjunto de regras de equivalência algébrica para gerar sistematicamente todas as expressões equivalentes à consulta original. Este processo é fundamental para explorar o espaço de busca e identificar o plano de execução mais eficiente.

1

Expressão Inicial

Consulta SQL convertida em árvore de álgebra relacional

2

Aplicação de Regras

Regras de equivalência geram novas expressões

3

Expansão Iterativa

Processo repetido até convergência

> Conceitualmente, gera-se todas as expressões equivalentes executando repetidamente as transformações até que nenhuma outra expressão seja encontrada: para cada expressão encontrada até aqui, use todas as regras de equivalência aplicáveis, e acrescente expressões recém geradas ao conjunto de expressões encontradas até aqui.

A técnica de enumeração exaustiva é muito dispendiosa em espaço e tempo, exigindo abordagens mais sofisticadas como programação dinâmica para viabilizar a otimização em consultas complexas.

Plano de Avaliação

Um plano de avaliação é uma especificação completa e detalhada de como uma consulta será executada. Ele define não apenas a estrutura lógica das operações, mas também os algoritmos específicos escolhidos para cada operação e a coordenação entre elas.

Algoritmos Específicos

Para cada operação (junção, seleção, projeção), o plano especifica o algoritmo exato: merge join, hash join, nested loop, etc.

Métodos de Acesso

Define como os dados serão acessados: varredura linear (scan), índice primário, índice secundário, ou estruturas especializadas.

Coordenação de Execução

Estabelece a ordem de execução, estratégias de pipeline, materialização de resultados intermediários e uso de memória.

Escolha dos Planos de Avaliação

A seleção de um plano de avaliação ótimo requer considerar a interação entre diferentes técnicas e algoritmos. A escolha do algoritmo de menor custo para cada operação independentemente pode não gerar o melhor algoritmo geral.

Exemplos de Interações

- **Merge Join:** Pode ser mais dispendiosa que hash join, mas oferece saída classificada que reduz custos em agregações externas
- **Nested Loop Join:** Oferece oportunidades valiosas para pipelining, eliminando necessidade de materialização
- **Índices:** Uso estratégico pode beneficiar múltiplas operações na sequência

Abordagens de Otimização

1. **Baseada em Custo:** Pesquisar todos os planos possíveis e escolher o de menor custo estimado
2. **Baseada em Heurística:** Usar regras práticas para escolher rapidamente um plano razoável

Sistemas modernos geralmente combinam ambas as abordagens para balancear qualidade e tempo de otimização.

Otimização Baseada em Custo

A otimização baseada em custo é uma abordagem sistemática que estima o custo de execução de diferentes planos alternativos e seleciona aquele com o menor custo esperado. Esta técnica requer modelos sofisticados de custo e estatísticas detalhadas sobre os dados.

Modelo de Custo

Funções matemáticas que estimam tempo de CPU, operações de E/S, uso de memória e transferência de dados para cada operação.

Estatísticas do Catálogo

Informações sobre cardinalidade, distribuição de valores, índices disponíveis e características físicas do armazenamento.

Comparação de Planos

Avaliação sistemática de alternativas, considerando todas as combinações viáveis de algoritmos e ordens de operação.

Programação Dinâmica na Otimização

A programação dinâmica é uma técnica fundamental para tornar viável a otimização baseada em custo, especialmente em consultas envolvendo múltiplas junções. Ela evita recomputação de subconsultas idênticas, armazenando e reutilizando soluções ótimas para subproblemas.

Subproblemas Menores

Começar com planos para relações individuais e pares de relações, encontrando o melhor plano para cada subconjunto.

Combinação Progressiva

Combinar soluções de subproblemas menores para construir soluções de problemas maiores, sempre mantendo apenas o melhor plano.

Solução Completa

Construir incrementalmente até obter o plano ótimo para a consulta completa, reutilizando resultados intermediários.

A programação dinâmica reduz a complexidade de exponencial para polinomial, tornando viável a otimização de consultas com 10-15 junções.

Algoritmo de Otimização da Ordem de Junção

Este algoritmo exemplifica a abordagem de programação dinâmica para encontrar o plano de junção de menor custo para um conjunto de relações S. Ele explora subconjuntos menores para construir soluções ótimas para problemas maiores, evitando recomputações.

```
procedure acharMelhorPlano(S)
   if (melhorplano[S].custo ≠ ∞)
      return melhorplano[S]
   // Se não estava calculado o melhorplano[S]; calcula-se agora
   for each subconjunto não vazio S_1 de S de modo que S_1 ≠ S
      P_1 = acharMelhorPlano(S_1)
      P_2 = acharMelhorPlano(S - S_1)
      A = melhor alg. para junção de resultados de P_1 e P_2
      custo = P_1.custo + P_2.custo + custo de A
      if custo < melhorplano[S].custo
         melhorplano[S].custo = custo
         melhorplano[S].plano =
            "executar P_1.plano;
            executar P_2.plano;
            juntar resultados de P_1 e P_2 usando A"
   return melhorplano[S]
```

Princípios do Algoritmo

- Construção bottom-up de planos
- Armazenamento de melhores planos para cada subconjunto
- Poda de planos subótimos
- Consideração de propriedades físicas interessantes

Complexidade e Escalabilidade

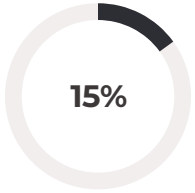
- Para n relações, existem (2n-2)!/(n-1)! possíveis ordens de junção esquerda profunda. A programação dinâmica reduz isso para O(n·2^n) subproblemas.
- Na prática, heurísticas adicionais e restrições no espaço de busca são aplicadas para consultas muito grandes, garantindo tempo de otimização aceitável mesmo sacrificando ligeiramente a optimalidade.

A ordem em que as junções são executadas tem impacto dramático no desempenho da consulta. O algoritmo de otimização explora sistematicamente diferentes ordenações, usando programação dinâmica para eficiência.

A recursão é a chave aqui: o algoritmo busca o melhor plano para subproblemas (subconjuntos de relações) e, a partir deles, constrói o plano para o problema completo, garantindo a otimalidade.

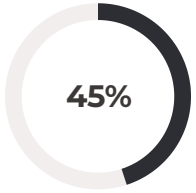
Custo da Otimização

O processo de otimização em si consome recursos computacionais significativos. É fundamental balancear o tempo gasto otimizando com os ganhos esperados na execução da consulta.



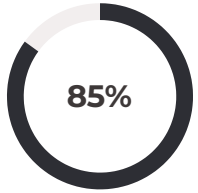
Consultas Simples

Tempo de otimização relativamente baixo, heurísticas frequentemente suficientes



Consultas Médias

Equilíbrio entre otimização baseada em custo e heurística



Consultas Complexas

Investimento em otimização é justificado pelos ganhos dramáticos na execução

Consultas executadas frequentemente (em aplicações OLTP) podem ser otimizadas uma vez e reutilizadas. Para consultas ad-hoc complexas (comum em OLAP), o tempo de otimização pode representar uma fração significativa do tempo total.

Estimativa de Cardinalidade de Resultados Intermediários

A precisão das estimativas de cardinalidade é crucial para a qualidade do plano de execução escolhido. O otimizador deve prever quantas tuplas resultarão de cada operação intermediária.

Cardinalidade Após Seleções

Estimada usando fatores de seletividade baseados em histogramas, valores distintos e distribuições de dados armazenadas no catálogo do sistema.

Cardinalidade Após Junções

Calculada considerando chaves estrangeiras, índices de correlação e assumindo independência ou dependência parcial entre atributos.

Impacto no Desempenho

Resultados intermediários menores reduzem drasticamente o custo de E/S e influenciam significativamente a escolha da ordem ideal de junção.

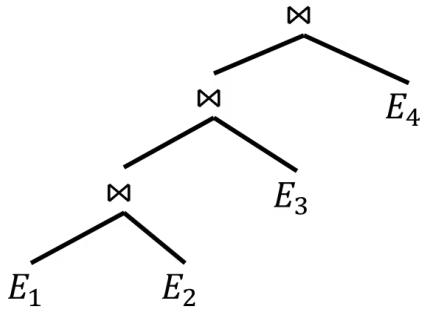
Estatísticas desatualizadas podem levar a estimativas incorretas e planos subótimos. Manutenção regular das estatísticas é essencial.

Árvores de Junção Esquerda Profunda

Estrutura Esquerda Profunda

Em árvores de junção esquerda profunda, a entrada do lado direito para cada junção é sempre uma relação base, nunca o resultado de uma junção intermediária.

Esta estrutura é uma forma restrita de árvore de consulta que simplifica significativamente o espaço de busca do otimizador.



Vantagens e Limitações

- **Pipelining:** Facilita execução em pipeline, pois o lado esquerdo pode alimentar continuamente a junção seguinte
- **Espaço Reduzido:** Menos planos a considerar durante otimização
- **Memória:** Apenas a relação direita precisa ser carregada inteiramente

Embora restrinja o espaço de busca, pesquisas mostram que o plano ótimo frequentemente está nesta classe, especialmente com nested loop joins.

Restrições no Espaço de Planos de Consulta

O número de planos de consulta possíveis cresce exponencialmente com o número de relações envolvidas. Para tornar a otimização viável, SGBDs aplicam restrições estratégicas que limitam o espaço de busca.

Restrição de Forma

Considerar apenas árvores esquerda profundas ou busihy trees, eliminando milhões de alternativas de formato.

Poda por Custo

Descartar planos parciais cujo custo já excede o melhor plano completo encontrado até o momento.

Compromisso

Balance entre qualidade do plano final e tempo gasto na otimização, essencial para aplicações práticas.

Essas restrições representam um compromisso pragmático: reduzem o custo de otimização substancialmente, podendo descartar o plano ótimo teórico, mas na prática produzem planos muito próximos do ótimo em tempo aceitável.

Representação das Regras de Equivalência

As regras de equivalência algébrica são formalizadas em notações que permitem sua aplicação sistemática durante a otimização. Uma das mais importantes é o "push down" de seleções.

1

Regra Original

$\sigma_{condição}(R \bowtie S)$ — Seleção aplicada após a junção completa

2

Transformação

Decomposição da condição em predicados que afetam apenas R ou apenas S

3

Regra Transformada

$(\sigma_{cond_R}(R)) \bowtie (\sigma_{cond_S}(S))$ — Seleções aplicadas antes da junção

> **Push Down Selection:** Aplicar seleções o mais cedo possível na árvore reduz drasticamente o tamanho dos resultados intermediários, diminuindo custos de E/S e processamento em todas as operações subsequentes.

Custo da Otimização

Compreender os fatores que influenciam o custo da otimização é essencial para projetar estratégias eficazes e balancear tempo de otimização com qualidade do plano resultante.

Estratégias de Contenção

- Timeout de otimização configurável
- Níveis de otimização (rápida, normal, exaustiva)
- Cache de planos para consultas repetidas

Benefícios do Investimento

Para consultas complexas executadas sobre grandes volumes, um bom plano pode ser ordens de magnitude mais rápido que um plano ingênuo.

Ordens Interessantes na Otimização Baseada em Custo

Uma "ordem interessante" é uma ordenação específica dos dados que pode beneficiar operações subsequentes na consulta. O otimizador deve considerar planos que produzem ordens interessantes, mesmo que tenham custo ligeiramente maior inicialmente.

1

Identificação

Detectar atributos de ORDER BY, GROUP BY e junções que se beneficiam de dados ordenados

2

Propagação

Rastrear quais operações preservam, destroem ou criam ordens interessantes

3

Avaliação

Comparar custo de produzir ordem interessante versus ordenação explícita posterior

4

Decisão

Escolher plano considerando benefícios globais da ordenação em toda a consulta

Por exemplo, usar merge join pode ser mais caro que hash join isoladamente, mas se produz saída ordenada necessária para GROUP BY subsequente, o custo total pode ser menor ao eliminar ordenação explícita.

Otimização Heurística

A otimização baseada exclusivamente em custo pode ser proibitivamente cara para consultas muito complexas. Heurísticas oferecem uma alternativa pragmática, aplicando regras que historicamente produzem bons resultados.

1

Substituir Produtos Cartesianos

Transformar operações de produto cartesiano ($\times$) seguidas de seleção (σ) em operações de junção explícitas ($\bowtie$), que são muito mais eficientes.

2

Seleção Antecipada

Realizar seleções (σ) o mais cedo possível na árvore de execução, reduzindo drasticamente o número de tuplas processadas.

3

Projeção Antecipada

Executar projeções (π) precocemente para eliminar atributos desnecessários e reduzir volume de dados intermediários.

4

Operações Restritivas Primeiro

Priorizar seleções (σ) e junções ($\bowtie$) mais restritivas (maior seletividade) antes de operações menos seletivas.

5

Identificar Pipelining

Reconhecer sub-árvores cujas operações podem ser canalizadas, executando-as em pipeline para evitar materialização.

Sistemas modernos frequentemente combinam heurísticas com otimização parcial baseada em custo: heurísticas reduzem rapidamente o espaço de busca, então otimização por custo refina escolhas críticas.

Referências

Elmasri & Navathe

Fundamentals of Database Systems

Pearson, 2016

Referência abrangente sobre fundamentos de sistemas de bancos de dados, cobrindo aspectos teóricos e práticos.

Korth, Sudarshan & Silberschatz

Database System Concepts

McGraw-Hill, 2019

Texto fundamental que serviu como base para a maioria dos exemplos apresentados nesta apresentação.

Özsu & Valduriez

Principles of Distributed Database Systems

Springer Nature, 2019

Obra especializada em sistemas de bancos de dados distribuídos, essencial para compreensão avançada.

Nota: Os conceitos e exemplos apresentados baseiam-se principalmente na literatura clássica de sistemas de bancos de dados, em especial *Database System Concepts* e *Fundamentals of Database Systems*.