PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
ESPECIALIZAÇÃO EM ADMINISTRAÇÃO DE BANCO DE DADOS

PROCESSAMENTO E OTIMIZAÇÃO DE CONSULTAS EM GERENCIADORES DE BANCO DE DADOS RELACIONAIS

Curitiba
Julho/2008

CARLOS EDUARDO GORGES

PROCESSAMENTO E OTIMIZAÇÃO DE CONSULTAS EM GERENCIADORES DE BANCO DE DADOS RELACIONAIS

Projeto de conclusão do curso de Especialização em Administração de Banco de Dados da Pontifícia Universidade Católica do Paraná, Centro de Ciências Exatas e Tecnologia, sob a orientação do Professor Attilio Zanelatto Neto.

Curitiba
Julho/2008

Resumo

Não há dúvidas que o otimizador é um dos componentes principais de um SGBD. Sua principal tarefa é escolher uma estratégia para resolver uma consulta relacional utilizando o menor consumo computacional possível. Sua evolução baseia-se em retirar do usuário a totalidade dessa responsabilidade objetivando a mitigação de erros e melhor eficácia, tornando-o cada vez mais automatizado, complexo e importante. Desta forma, o objetivo deste trabalho é criar um material de referência sobre o processamento de consultas em SGBDs com foco na fase de otimização, gerando um comparativo entre os métodos de otimização abordados. O otimizador faz parte do conjunto de tarefas que o SGBD precisa realizar para que a consulta requisitada seja executada e o resultado seja devolvido para o usuário. Esse conjunto de tarefas é chamado de processamento de uma consulta e ela é dividida em várias etapas com suas respectivas responsabilidades. Os métodos de otimização são as formas como a problemática de otimização podem ser resolvidas no processamento de uma consulta. Os dois métodos mais comuns utilizados para a otimização de consulta em SGBDS são: o baseado em regras heurísticas, também conhecido como otimizador por regra, e o baseado em estimativa de custo, também conhecido como otimizador por custo. O resultado desses otimizadores é o plano de execução, que corresponde a uma seqüência de etapas empregadas no processo de execução de uma consulta.

Palavras-chave: SGBD, Otimizador, processamento de consultas SQL.

Lista de Figuras

Lista de siglas e abreviaturas

1. SGBD (Sistema de Gerenciamento de Bancos de Dados) – Sistema responsável pelo armazenamento, organização, gerenciamento das bases de dados e pela garantia da consistência dos mesmos;
2. SGBDR (Sistema de Gerenciamento de Banco de Dados Relacional) – SGBD onde os dados são representados em linhas e colunas, baseado no modelo relacional de dados;
3. SGBD Oracle – Produto desenvolvido pela empresa norte americana Oracle Corporation;
4. SQL – Structured query language (linguagem de consulta estruturada) é a linguagem de pesquisa declarativa utilizada nos SGBD atuais;
5. Predicados – valores aplicados nos filtros das consultas;

SUMÁRIO

1. Introdução

O objetivo dos otimizadores de consulta é claro: escolher uma estratégia para resolver uma consulta relacional utilizando o menor consumo computacional possível. Sua evolução baseia-se em retirar do usuário a totalidade dessa responsabilidade objetivando a mitigação de erros e melhor eficácia, tornando-o cada vez mais automatizado, complexo e importante em relação aos demais componentes de um SGBD.

Dada a importância dos otimizadores no cenário dos SGBDs, e ao fato de que, atualmente, existe uma grande quantidade de profissionais trabalhando nessa área sem o conhecimento adequado no assunto abordado. A proposta desse trabalho é criar um material de referência sobre o processamento de consultas em SGBDs com foco na fase de otimização, gerando comparativos dos métodos de otimização abordados.

2. Desenvolvimento

Quando se submete uma consulta SQL para um SGBD, se informam os resultados que se pretende obter, mas não necessariamente o melhor caminho para fazê-lo. O responsável por escolher a estratégia mais eficiente para o processamento dessas consultas é o componente dos SGBD conhecido como otimizador, que é um algoritmo (modelo matemático) que "transforma" a consulta com o objetivo de minimizar o tempo e o custo computacional da sua execução.

Suas principais decisões são:

- Qual algoritmo e abordagem de otimização aplicar, quando há mais de uma disponível;
- Se, e como, transformar os comandos da consulta;
- Decidir o método de acesso a cada tabela;
- Decidir a ordem de união das tabelas;
- Decidir o método de união das tabelas;
- Decidir como executar os operadores;

Essas decisões são baseadas em:

- Conjunto de regras;
- Estatísticas;
- Parâmetros;
- Modificadores (Hints);
- Informações no dicionário de dados;
- Predicados (filtros);

2.1. Processamento de uma consulta

O otimizador faz parte de um conjunto de tarefas que o SGBD precisa efetuar para que cada consulta requisitada pelo usuário seja executada e o resultado seja devolvido. Esse conjunto de tarefas é chamado de processamento de uma consulta e ela é dividida em várias etapas com suas respectivas responsabilidades.

2.1.1. Etapas do processamento

As etapas que compreendem o processamento de uma consulta estão no diagrama encontrado na figura 1 (LAHDENMAKI, 2005);

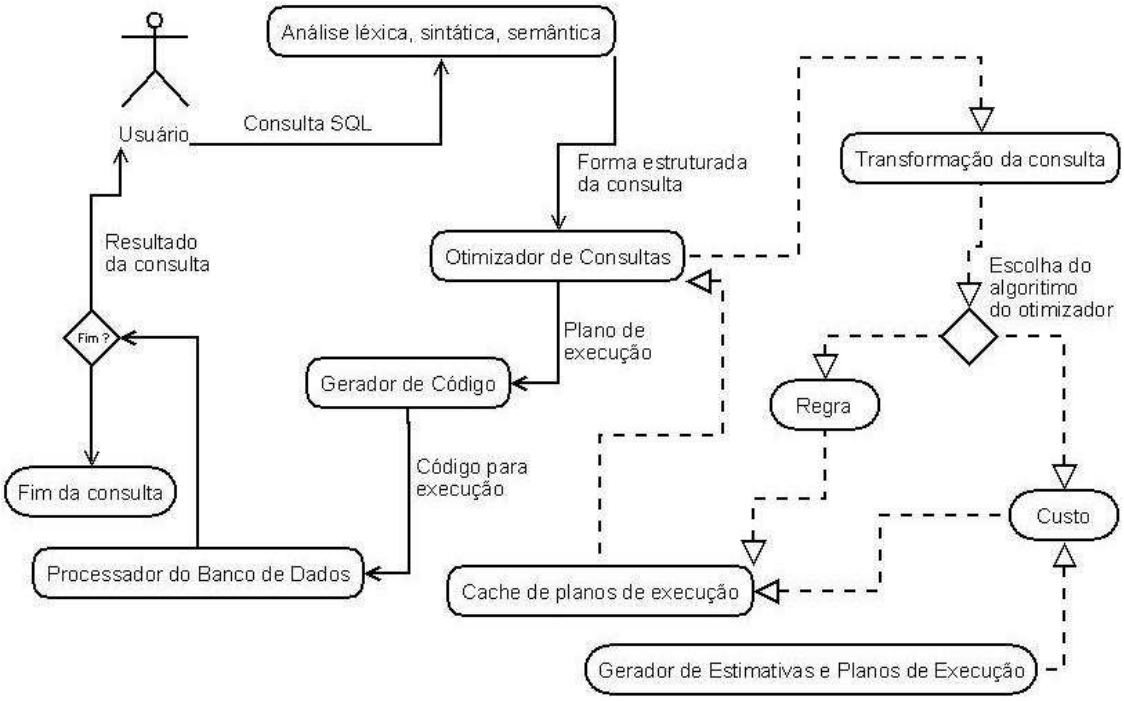


Figura 1 – Estrutura do processamento de consultas

2.1.1.1. Análise léxica, sintática e semântica (parse)

Esta é a primeira fase de processamento da consulta submetida pelo usuário. Sua responsabilidade é montar, a partir do texto da consulta SQL, uma representação interna da consulta em uma forma estruturada, chamada árvore de consulta, já com as validações necessárias efetuadas. Ela é dividida em:

- Análise léxica (alfabeto) e sintática (gramática):

Nestas fases, o texto da consulta é validado gramaticalmente e separado em uma estrutura de acordo com o alfabeto da linguagem SQL do SGBD em questão. Consultas mal formadas, geralmente, ocasionam erro nessas fases.

- Análise semântica (validação):

Nesta fase, os objetos encontrados na estrutura gerada são validados. Por exemplo, é verificado se as tabelas referenciadas na consulta existem e se o usuário tem permissão para acessa-lás.

2.1.1.2. Otimizador de consultas

A fase de otimização têm como objetivo escolher uma estratégia, entre as várias possíveis, para recuperação dos resultados a partir da árvore de consulta. Os dois algoritmos mais comuns usados para a otimização de consulta em SGBDS são: o baseado em regras heurísticas, também conhecido como otimizador por regra, e o baseado em estimativa de custo, também conhecido como otimizador por custo. A estratégia de execução também é chamada de plano de execução ou plano de acesso. Essa fase é dividida em:

- Transformação da consulta:

A transformação da consulta altera a árvore de consulta com o objetivo de gerar de uma consulta equivalente mais eficiente, gerando assim um plano de execução melhor na fase de otimização. As quatro principais técnicas de transformação são:

- Agrupamento das visões

Sabendo que as visões são tratadas como consultas separadas do bloco da consulta principal, esta fase busca reescrever a consulta unindo a consulta do corpo da visão com o bloco da consulta principal, possibilitando a otimização conjunta das consultas e disponibilizando mais uma opção para o otimizador.

- Recuperação de predicados

Para cada visão não agrupada, essa fase se encarregará de avaliar os predicados da consulta principal, e procurará aplicá-los diretamente na visão. Essa técnica melhora os sub-planos das visões, pois permitem, por exemplo, que esses predicados sejam usados para um acesso via índice ou como simples operação de filtro;

- Remoção de sub-consultas

Assim como as visões, as sub-consultas são tratadas separadamente da consulta principal. Esta fase busca unir as sub-consultas no bloco da consulta principal, transformando

esta consulta aninhada em uma junção da consulta principal, disponibilizando mais uma opção para o otimizador.

- Reescrita da consulta utilizando visões materializadas

Sabendo que uma visão materializada é similar a uma consulta armazenada em uma tabela, quando o transformador de consultas encontra uma consulta compatível com uma visão materializada, ele reescreve a consulta utilizando essa visão. Isto apresenta ganhos de desempenho, uma vez que parte do resultado já está previamente processado, dispensando a necessidade de executá-lo novamente.

- Escolha do algoritmo de otimização:

Alguns SGBDs dispõem de mais de um algoritmo e abordagem de otimização. É nesta fase que um dos algoritmos e abordagem é escolhido de acordo com as regras do SGBD usado.

- Execução do algoritmo e abordagem de otimização:

Nesta fase é definida, de acordo com o algoritmo e abordagem escolhidos, qual estratégia de processamento para a consulta que será usada. Sempre que essa fase é necessária, o processo é chamado de parse completo (hard parse) e seu resultado é o plano de execução escolhido.

- Cachê dos planos de execução:

A fase de otimização exige certa quantidade de recursos computacionais, principalmente quando o algoritmo utilizado é baseado em estimativa de custo. Desta forma, nesta última fase, o plano de execução é guardado para posterior reutilização.

2.1.1.3. Parse completo e parse parcial

Como a maioria dos produtos de SGBDs considera que a fase de otimização faz parte do parse (junto com a análise léxica, semântica e sintática), várias literaturas (GREEN, 2002)

(POWELL, 2003) definem o conceito de parse completo (hard parse) e parse parcial (soft parse). O parse parcial ocorre quando um plano de execução previamente gerado é reutilizado no processamento da consulta. O parse completo ocorre quando é necessária a chamada do algoritmo de otimização para a geração do plano de execução. O objetivo do reuso é diminuir o impacto dos recursos computacionais necessários para todo o processamento da consulta e cada produto contém uma solução para resolver este problema. No SGBD Oracle existe um conceito de variáveis bind, que funcionam como predicados dinâmicos, e variáveis literais, que são predicados estáticos. A definição do tipo do predicado é feita pelo usuário na chamada da consulta. Os planos previamente gerados são reutilizados quando a consulta submetida é equivalente a original, onde somente o valor dos predicados definidos como bind mudem. Os planos de execução guardados para reuso são gerados com os predicados usados da primeira execução da consulta, processo chamado “bind variable peeking”. No SGBD Oracle ainda existe outra forma para forçar o reuso de planos de execução: as variações “SIMILAR” e “FORCE” do parâmetro de configuração do compartilhamento de cursores (cursor_sharing). Ambas as opções buscam transformar automaticamente as consultas submetidas com predicados do tipo literal em consultas com predicados do tipo bind. A diferença entre o “SIMILAR” e o “FORCE” é que no primeiro caso há regras não documentadas para a definição de quais predicados devem ser convertidos, e no segundo caso ele sempre irá efetuar a conversão. Essa diferença é crucial para a performance da execução das consultas, pois um plano de execução gerado com certos predicados em muitos casos não é o ideal para outras combinações desses predicados.

2.1.1.4. Gerador de código e processador (execute/fetch)

O objetivo do gerador de código e processador é executar o plano de execução gerado nas fases anteriores do processamento da consulta e retornar, gradativamente, as linhas que compõem o resultado para o usuário.

2.1.2. Plano de execução

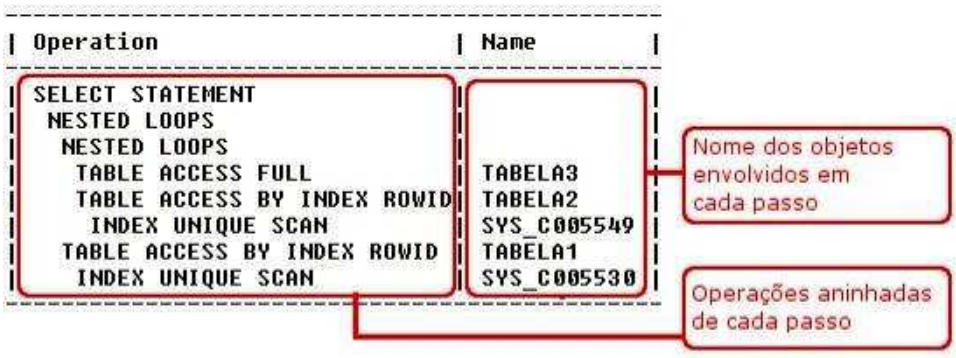
O plano de execução é a estratégia escolhida pelo algoritimo e abordagem de otimização adotados pelo SGBD para a execução da consulta. O plano de execuções é formado por um grupo estruturado e ordenado de instruções que informam ao SGBD, passo a passo, como a consulta deve ser executada.

2.1.2.1. Informações relevantes

De acordo com o SGBD e otimizador que gerou o plano, algumas informações relevantes são apresentadas. As figuras 2 e 3 demonstram a decomposição dessas informações no SGBD Oracle 10g. Na decomposição apresentada estão às colunas com as informações relevantes dentro do plano. Apesar de a decomposição estar apresentada em colunas, o plano é lido linha a linha, da operação aninhada com maior identação para a menor, na ordem de cima para baixo quando houver operações no mesmo nível.

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		1491K	52M		10407 (4)	00:02:05
* 1	HASH JOIN		1491K	52M	37M	10407 (4)	00:02:05
2	TABLE ACCESS FULL	TABELA3	1502K	20M		941 (6)	00:00:12
* 3	HASH JOIN		1491K	32M	30M	5053 (4)	00:01:01
4	TABLE ACCESS FULL	TABELA1	1498K	12M		695 (7)	00:00:09
5	TABLE ACCESS FULL	TABELA2	1491K	19M		941 (6)	00:00:12

Figura 2 – Informações do plano de execução gerado por CUSTO



Operation	Name
SELECT STATEMENT	
NESTED LOOPS	
NESTED LOOPS	
TABLE ACCESS FULL	TABELA3
TABLE ACCESS BY INDEX ROWID	TABELA2
INDEX UNIQUE SCAN	SYS_C005549
TABLE ACCESS BY INDEX ROWID	TABELA1
INDEX UNIQUE SCAN	SYS_C005530

Figura 3 – Informações do plano de execução gerado por REGRA

As informações mais importantes encontradas nos planos de execução são as seguintes:

- Modo do otimizador

Como o otimizador escolhe o algoritmo e abordagem de otimização, quando há mais de um disponível, o plano de execução será influenciado por esta descisão já que ele é o resultado dessa escolha. Dependendo do SGBD e da ferramenta de geração do plano de execução, o modo do otimizador é encontrado como uma informação auxiliar no plano. No

exemplo acima precisamos analisar o plano para verificar o modo do otimizador: no otimizador por custo as estimativas estão presentes.

- Operações aninhadas

As operações mostram, de forma aninhada, os métodos de acesso, ordem de união, métodos de união e operadores escolhidos pelo algoritmo do otimizador.

- Nome dos objetos envolvidos em cada passo

São as fontes de dados acessadas por todo o plano: podem ser índices, tabelas, partições, visões, acessos remotos a outros bancos e muitos outros. Cada fonte de dados é acessada a partir de uma operação, que no exemplo das figuras 2 e 3 é encontrada na mesma linha, na coluna “operações aninhandas”.

- Uso estimado de espaço temporário necessário

É a estimativa de uso de espaço temporário em meio físico (disco, storage, etc), para operações de agrupamento e ordenação, calculadas pelo otimizador por custo.

- Tempo estimado para execução de cada operação

É o tempo estimado pelo otimizador por custo para execução de cada operação.

- Custo (cost)

O custo estimado pelo otimizador por custo é uma unidade que procura demonstrar o quanto cada operação consumirá em recursos. A lógica de cálculo desse valor não é documentada e muda de acordo com as versões dos SGBDs.

- Uso de CPU estimado em cada operação

É a porcentagem de uso de CPU estimado pelo otimizador por custo.

- Número de linhas estimadas em cada passo (cardinalidade)

A cardinalidade representa o número de linhas em uma tabela ou o número de linhas distintas retornadas por um índice. A cardinalidade de uma consulta é o número de linhas estimada do resultado.

- Número de bytes estimados em cada passo (volume de dados em bytes)

O volume de dados é o tamanho estimado, em bytes, dos dados a serem manipulados em cada operação. O calculo é baseado na cardinalidade multiplicada pelo tamanho médio da linha da tabela coletada nas estatísticas.

2.1.2.2. Métodos de acesso

Os métodos de acesso são os modos com que os dados são obtidos de uma tabela em uma base de dados. Em geral, o acesso via índice deve ser usado para recuperação de uma quantidade pequena de dados de uma tabela enquanto a leitura completa é mais eficiente na recuperação de uma quantidade grande de dados. O acesso direto é a forma mais simples e rápida de acesso a uma linha. Esses três principais métodos estão descritos abaixo:

- Leitura total da tabela:

A consulta irá ler todas as linhas e filtrar o resultado de acordo com os predicados aplicados na consulta. Essa forma de leitura utiliza requisições multi-bloco em disco, que normalmente são mais rápidas que requisições de bloco simples.

- Índice b-tree

Os acessos por índices, que são estruturas auxiliares as tabelas, permite pesquisas rápidas por pequenas frações de uma tabela, como uma linha. Ele é formado por uma estrutura em forma de árvore, onde os valores da pesquisa ficam em seus galhos e os endereços das linhas na tabela ficam nas folhas. Após a pesquisa nos galhos pelo predicado da consulta, os endereços de acesso direto das linhas são retornados e usados para acessar a tabela. Esse tipo de acesso direto utiliza requisições por bloco simples em disco, que normalmente são lentas.
Existem várias variações de acessos via índice, as principais são as seguintes:

- Index-unique scan

  A pesquisa é efetuada em um índice que não contém valores repetidos,, permitindo que a consulta termine logo que a primeira ocorrência do valor seja achada. É usado, por exemplo, em consultas com predicados em colunas com valores únicos, como o CPF de uma pessoa.

- Index-range scan

  Esse tipo de acesso ocorre em consultas onde os predicados são intervalos de valores ou quando os campos do índice não contêm valores únicos. A pesquisa é efetuada com predicado de menor valor e, com a primeira folha resultado, a pesquisa continua na lista ordenada (duplamente ligada) até que o maior valor seja encontrado. É usado, por exemplo, na pesquisa de pessoas com idade entre 10 e 20 anos.

- Index-skip scan

  A pesquisa é efetuada nos galhos de um sub-nível do índice. Isto é possível efetuando várias sub-consultas, uma para cada galho dos níveis anteriores, atraz do valor pesquisado. É usado quando o predicado é aplicado em colunas que somente estão incluídas a partir da segunda posição de índices e a leitura completa da tabela é muito custosa.

- Index full scan

  A pesquisa é feita lendo toda a lista ordenada (duplamente ligada) do índice, permitindo um resultado já ordenado. Normalmente usado quando todos os valores de uma coluna presente no índice estão sendo requisitados ordenadamente.

- Index fast full scan

  A pesquisa é feita lendo todos os blocos do índice desordenadamente, como o índice está armazenado. Usado para evitar a leitura completa da tabela quando todos os campos envolvidos na consulta estão no índice e não são nulos.

- Acesso direto (ROWID)

O acesso direto a uma linha varia de acordo com o SGBD, mas em geral sua representação é uma união do identificador do arquivo, endereço do bloco e linha da tabela. Na maioria dos SGBDs esse endereço é tratado internamente, sem que o usuário tenha acesso a ele (como por exemplo, nas folhas dos índices). No SGBD Oracle é possível passar esse endereço, em um formato definido, como predicado de uma consulta.

2.1.2.3. Métodos de união

Sempre que mais de uma fonte de dados (tabela, índice, etc) é acessada em um bloco de consulta, o SGBD deve escolher a forma de uni-las para possibilitar o processamento conjunto do resultado. Os métodos mais comuns de união, que são escolhidos de acordo com a consulta e o algoritmo de otimização, são:

- Junção por iteração simples (nested loops join)

É o método mais comum e sua grande vantagem é retornar rapidamente algum resultado para o usuário, sendo lento para grandes volumes de dados se comparado com outros métodos de junção. Esse algoritmo define uma fonte de dados externa e uma fonte de dados interna, interagindo em todo o resultado da fonte de dados interna para cada linha da fonte de dados externa. A figura 4 demonstra a lógica do funcionamento desse algoritmo.

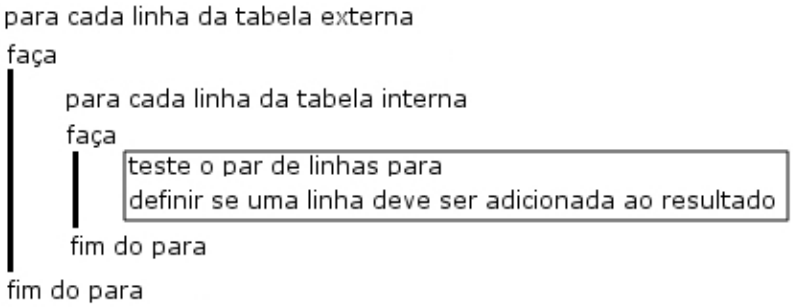


Figura 4 – Lógica da junção por iteração simples

- Junção por hashing (hash join)

Este é o método mais comum para processamento de grandes volumes de dados e só está disponível no otimizador por custo. Sua lógica escolhe a menor das duas fontes de dados envolvidas na união e monta uma tabela hash em memória que contém a representação dos valores distintos das chaves da junção encontrados na tabela menor. A partir da tabela hash, toda a fonte de dados maior será lida em busca dos valores em comum, retornando os que atendem a junção.

- Junção por classificação e intercalação (sort-merge join)

Este método também é eficiente para processamento de grandes volumes de dados e sua grande vantagem é retornar o resultado já ordenado. Em uma junção com esse algoritmo, às duas fontes de dados são ordenadas e percorridas inteiramente para a união.

- Junção por plano cartesiano (cartesian merge join)

Essa união ocorre quando não há uma condição de união entre uma ou mais fontes de dados, forçando a união de cada linha de uma fonte de dados com cada linha de outra. Junção por plano cartesiano normalmente é gerado por erros de escrita na consulta.

Além dos métodos já descritos, existe o método para junção externa (outer join), que retornam não só as linhas da junção, mas também as linhas que não atendem a chave da junção em uma das fontes de dados. Existem ainda outras derivações e métodos: semi join, anti join, junção paralela e algumas outras dependendo do SGBD.

2.1.2.4. Ordem de união

Quando mais de uma fonte de dados (tabela, índice, etc) é acessada em um bloco de consulta, assim como os métodos de união, a ordem em que fontes de dados serão unidas precisa ser definida. A ordem de união influenciará na quantidade de dados trabalhados e consequentemente na velocidade da execução da consulta. A forma como essa ordem é escolhida depende do algoritmo de otimização usado e será discutida nos capítulos referente aos métodos de otimização.

2.1.2.5. Operações específicas

As operações são rotinas encontradas em um SGBD que executam tarefas imprescindíveis na resolução dos planos de execução que serão usados para as consultas. No SGBD Oracle, as principais operações especificas são:

- IN-LIST ITERATOR

Iterage em uma fonte de dados filtrando os valores que estão em um predicado do tipo “IN”.

- COUNT

Operação que conta o número de linhas selecionadas de uma tabela.

- SORT

Operação que efetua a ordenação em uma fonte de dados. Existem várias operações e um método de união que usa esse operador: MIN, MAX, AVG, COUNT, STDDEV, ORDER BY, GROUP BY, DISTINCT, UNION, INTERSECT, MINUS e Sort-Merge Join;

- SEQUENCE

Efetua um acesso a um objeto do tipo SEQUENCE (sequência);

- UNION ALL

Operação que aceita dois conjuntos de linhas e retorna a união desses conjuntos. Note que a função “UNION” eliminanda duplicados a partir de uma operação extra de ordenação (SORT).

- VIEW

Operação que executa uma visão e retorna seu resultado para outra operação.

- FILTER

Operação que recebe um conjunto de linhas e cria um subconjunto de acordo com a regra de filtro. Permite ao SGBD filtrar resultados mesmo quando o método de acesso à fonte de dados não permita isso (leitura completa da tabela, por exemplo).

2.1.3. Métodos de otimização

Os métodos de otimização são as formas como a problemática de otimização descrita no capítulo de processamento da consulta é resolvida. Apesar de existirem vários otimizadores para casos específicos, os mais usados em SGBDs são o otimizador por regra e o otimizador por custo.

2.1.3.1. Otimização por regra

O otimizador por regra foi amplamente usado pelos SGBDs, mas atualmente foi descontinuado ou mantido por compatibilidade pela maioria dos produtos, pois seu funcionamento é considerado limitado comparado com a otimização por custo: há limitações de recursos, como união por hash e index-skip scan, e limitações na qualidade de suas decisões, pois o algoritmo não utiliza variáveis importantes como o tamanho e distribuição dos dados das tabelas e índices. Suas decisões são baseadas somente na sintaxe da consulta SQL, em regras estáticas e em poucas informações do dicionário de dados. Para a geração do plano de execução, os predicados são processados na ordem que aparecem nos parâmetros da clausula “where” do texto da consulta SQL e os métodos de acesso são escolhidos a partir de uma lista de regras definidas de acordo com o SGBD usado. Como normalmente é possível

acessar de mais de uma forma as fontes de dados, para cada regra há um peso (ranking) ou prioridade. Quando há mais de um método possível, o método com menor peso ou maior prioridade é escolhido. Como exemplo, a figura 5 contém as regras usadas no otimizador por regra do SGBD Oracle, que é baseado em peso (rank). Como é possível notar na figura 5, o acesso pelo endereço da linha ou por índice, quando disponíveis, são sempre as melhores escolhas, e a leitura total da tabela, a pior. Quando há duas formas de acesso com o mesmo peso, cada SGBD implementa uma forma de desempate: no SGBD Oracle o critério de desempate é o objeto com a data de criação mais atual. Como discutido anteriormente, quando há mais de uma fonte de dados envolvida na consulta os métodos de união a ordem da união devem ser definidas. No otimizador por regra o método de união também é escolhido com base na lista de regras. No SGBD Oracle, o otimizador irá sempre utilizar a junção por iteração simples (nested loops join) com uma exceção: caso o método de acesso de uma fonte de dados tenha peso 12 ou pior (maior) e a chave de união seja por igualdade (equijoin), o otimizador irá utilizar a junção por classificação e intercalação (sort-merge join). A ordem de união é definida a partir da ordem inversa das fontes de dados encontrada no “from” do texto da consulta.

Métodos de acesso e sua prioridade	
Peso 1	Pesquisa por linha única acessada pelo endereço (ROWID)
Peso 2	Pesquisa por linha única acessada por uma junção em cluster
Peso 3	Pesquisa por linha única acessada pela chave hash do cluster a partir da chave primária ou única
Peso 4	Pesquisa por linha única acessada por chave única ou primária indexada
Peso 5	Junção por cluster
Peso 6	Pesquisa por chave hash do cluster
Peso 7	Pesquisa por chave indexada do cluster
Peso 8	Pesquisa em índice composto
Peso 9	Pesquisa em índice com uma única coluna
Peso 10	Pesquisa por intervalo fechado em colunas indexadas
Peso 11	Pesquisa por intervalo aberto em colunas indexadas
Peso 12	Junção por classificação e intercalação (Sort-Merge Join)
Peso 13	Uso de operadores MAX ou MIN em colunas indexadas
Peso 14	Uso do operador ORDER BY (SORT) em uma coluna indexada
Peso 15	Leitura total da tabela

Figura 5 – Regras e prioridades do otimizador por regra no SGBD Oracle

2.1.3.2. Otimização por custo

O otimizador por custo foi criado com o intuito de resolver de forma mais acurada e dinâmica a problemática de otimização. Atualmente ele é o otimizador utilizado pelos SGBDs líderes de mercado. Seu funcionamento consiste em estimar sistematicamente o custo de estratégias de execução diferentes e escolher o plano de execução com o menor custo estimado. Suas decisões são baseadas no dicionário de dados, estatísticas, parâmetros e não são influenciadas pela sintaxe da consulta SQL. Como sua decisão depende de estatísticas, é necessária a atualização periódica dessas informações para o melhor funcionamento do algoritmo. Os dois componentes principais do otimizador por custo são: o gerador de estimativas e o gerador de planos de execução. A função desses dois componentes é gerar um conjunto de planos de execução, estimando o custo de cada um. O conjunto de planos de execução é formado examinando todos os possíveis métodos de acesso (uso de índice, leitura total da tabela, etc), algoritmos de união (sort-merge join, hash join, nestead loops, etc) e operadores (filter, sort, etc). Isto possibilita que o otimizador determine qual dos planos é o mais eficiente a partir de seu custo. O tamanho do conjunto de planos gerados normalmente é limitado por parâmetro de configuração para impedir que o próprio otimizador se torne um problema. A métrica de custo do otimizador é uma tentativa de calcular o custo computacional do processamento de um plano. Cada SGBD leva em conta certa quantidade de fatores para o cálculo desse valor e normalmente ele é baseado no número de operações de entrada/saída necessários, CPU e outros fatores determinados pelo dicionário de dados. Desta forma o método de acesso, o método de união, a ordem de união e os operadores são

escolhidos de acordo com o plano de execução com menor custo. É importante ressaltar que mesmo trabalhando com mais informações que outros algoritmos de otimização, o custo também é heurístico, existindo o risco da escolha de uma estratégia de execução que não seja a melhor.

2.1.3.2.1. Parâmetros

Os parâmetros do otimizador são configurações estáticas do SGBD que objetivam ajustar o comportamento do otimizador de acordo com a característica da aplicação. Os principais ajustes definem a tendência da escolha do otimizador a planos de execução com melhor desempenho no retorno do inicio do resultado da consulta ou no retorno do total do resultado da consulta. Os principais parâmetros do otimizador no SGBD Oracle são os seguintes:

- OPTIMIZER_MODE

Decide qual algoritmo e abordagem do otimizador será usado. Os valores possíveis são: RULE (Regra), ALL_ROWS (custo com abordagem para obter todas as linhas rapidamente), FIRST_ROWS (custo com abordagem para obter as primeiras linhas rapidamente) e CHOOSE (o otimizador escolhe qual utilizar acordo com a disponibilidade de estatísticas);

- OPTIMIZER_INDEX_COST_ADJ

Define a tendência de escolha do otimizador por acesso por índice. Esse parâmetro pode ser configurado de “1” até “10000”, onde “100” significa que o acesso por índice é igual aos outros métodos e 50 significa que o uso de índice custa à metade que outro método.

- OPTIMIZER_INDEX_CACHING

Define a tendência da escolha do otimizador pelo método de união “junção por iteração simples” (nested loops join) e o operador IN-LIST ITERATOR ao invés de junção por hash (hash join) ou junção por intercalação (sort-merge join). Seu valor é a porcentagem de blocos dos índices que o otimizador deve assumir que está em memória.

- DB_FILE_MULTIBLOCK_READ_COUNT

Este parâmetro determina o número máximo de blocos que serão lidos em uma única operação de leitura/gravação durante uma leitura seqüencial (multi-bloco). Quanto maior este valor, maior a tendência do otimizador escolher o método de acesso por leitura completa da tabela ou index fast full scan. A partir do oracle 9 e a adoção do “CPU Costing Model” esse valor é estimado quando as estatísticas de sistema são coletadas, ignorando o valor desse parâmetro e usando no lugar o valor encontrado no dicionário de dados.

2.1.3.2.2. Estatísticas

As estatísticas são informações coletadas em tempo de execução que ajudam o otimizador tomar melhores decisões.

As principais estatísticas coletadas são:

- Estatísticas de tabela: número de blocos, número de blocos vazios, número de chained ou migrated rows, tamanho médio da linha, data da coleta e tamanho da amostra.
- Estatísticas de índices b-tree: altura, número de blocos folha, número de chaves distintas, número médio de blocos folha por chave, número médio de blocos por chave, número de entradas (index entries), clustering factor, data da coleta e tamanho da amostra.
- Estatísticas de Colunas: número de valores distintos, menor valor, maior valor, data da coleta e tamanho da amostra;

Outra estatística importante nos casos de campos com valores com distribuição não uniforme são os histogramas, que permitem ao otimizador escolher planos de execução distintos de acordo com o valor do predicado.

2.1.3.2.3. Modificadores (hints)

Como o otimizador por custo é em boa parte, automático, a interação do usuário sobre como suas decisões são tomadas é limitada. No SGBD Oracle, os modificadores, que são diretivas de compilação, permitem que o usuário influencie a escolha do otimizador por custo.

2.1.4. Custo versus regra

Com a evolução do otimizador por custo, vários recursos foram sendo melhorados e adicionados. Atualmente o SGBD Oracle está na versão 11g e existem inúmeras vantagens para o uso do otimizador por custo, por exemplo:

- Tabelas e índices particionados: capacidade de dividir físicamente as informações contidas em tabelas e índices em porções menores a partir de regras definidas pelo usuário;
- Tabelas organizadas por índice: criação de tabelas que respeitam a ordem de um índice b-tree;
- Índices com chave reversa: índices onde os valores das chaves são invertidos, byte a byte;
- Paralelismo na execução: permite a execução paralela da consulta;
- Otimizador extensível: suporte a extensão das funcionalidades do otimizador pelo usuário;
- Reescrita da consulta para o uso de visões materializadas: funcionalidade discutida no capitulo 2.1.1.2;
- Método de união por hash (Hash joins): método de união discutido no capitulo 2.1.2.3;
- Índices bitmap e união por índices bitmap: novo tipo de índice baseado em mapas de bits;
- Método de acesso “Index skip scans”: método de acesso discutido no capitulo 2.1.2.2;

Para comprovar a evolução do otimizador, foram efetuados testes em alguns desses recursos. O banco de dados utilizado foi o Oracle Database 10g Enterprise Edition Release 10.2.0.1.0:

INDEX SKIP SCAN

Para o teste, foi usada a tabela no formado do diagrama E-R encontrado na figura 6.

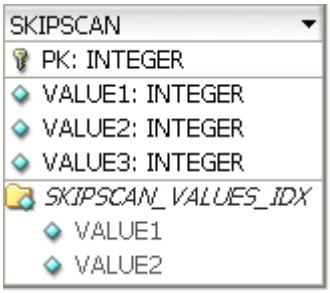


Figura 6 - Index skip scan: diagrama E-R da tabela usada para o teste

Nesta tabela foram carregadas 3.020.730 linhas, onde a coluna PK contém valor seqüencial de 0 a 3020729, a coluna VALUE1 contém valores de 1 a 100, a coluna VALUE2 contém valores de 1 a 10000 e a coluna VALUE3 contém valores de 1 a 50000. Todas as colunas VALUE contêm valores com distribuição uniforme.

No paradigma antigo, o índice SKIPSCAN_VALUES_IDX somente seria usado caso o filtro aplicado fosse em VALUE1 ou VALUE1 e VALUE2. O otimizador por custo permite, caso seja interessante, utilizar o índice mesmo nos casos onde o filtro seja aplicado nas colunas dos sub-níveis, onde antes a abordagem possível era somente a leitura completa da tabela, resultando em menor número de leituras necessárias para o processamento da consulta. Os seguintes testes foram efetuados, gerando os seguintes planos de execução:

- Filtro na coluna presente no sub-nível do índice e otimizador por regra:

SELECT * FROM SKIPSCAN where VALUE2 = 1;

```
------------------------------------
| Id  | Operation         | Name     |
------------------------------------
|   0 | SELECT STATEMENT  |          |
|*  1 |  TABLE ACCESS FULL| SKIPSCAN |
------------------------------------
```

- Filtro na coluna presente no sub-nível do índice e otimizador por custo:

SELECT * FROM SKIPSCAN where VALUE2 = 1;

```
-------------------------------------------------------------
| Id  | Operation                    | Name                 |
-------------------------------------------------------------
|   0 | SELECT STATEMENT             |                      |
|   1 |  TABLE ACCESS BY INDEX ROWID| SKIPSCAN             |
|*  2 |   INDEX SKIP SCAN            | SKIPSCAN_VALUES_IDX |
-------------------------------------------------------------
```

O resultado dos testes, com o número de blocos necessários para o processamento da consulta e tempo de execução encontra-se nos gráficos das figuras 7 e 8.

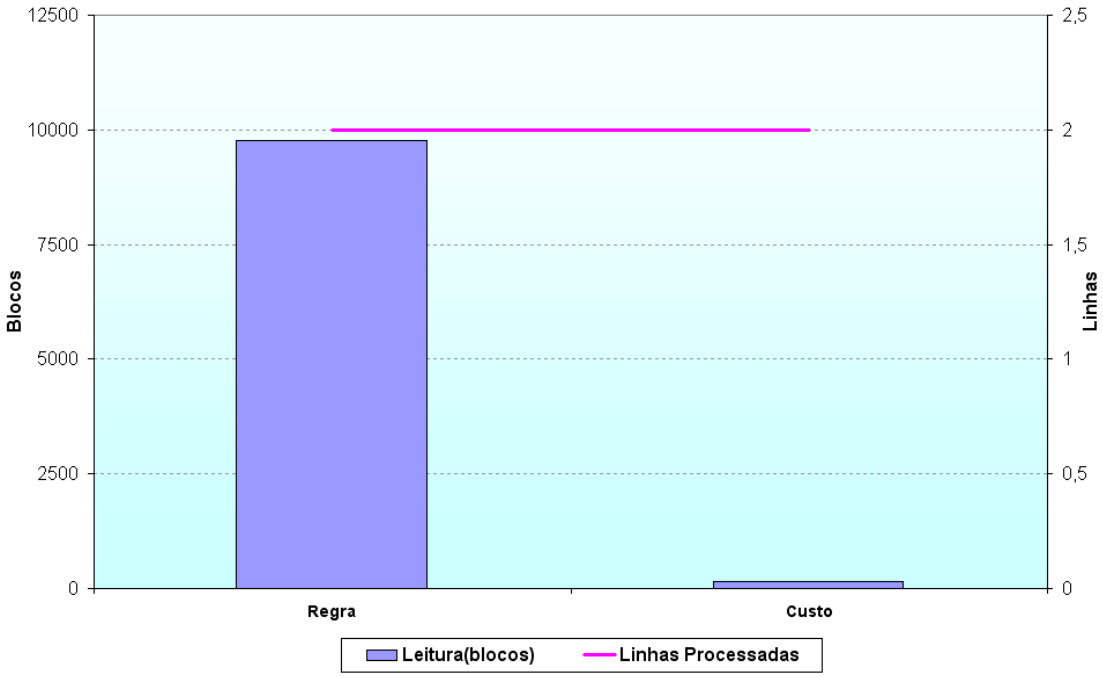


Figura 7 – Index skip scan: gráfico comparativo de leituras "Regra versus Custo"

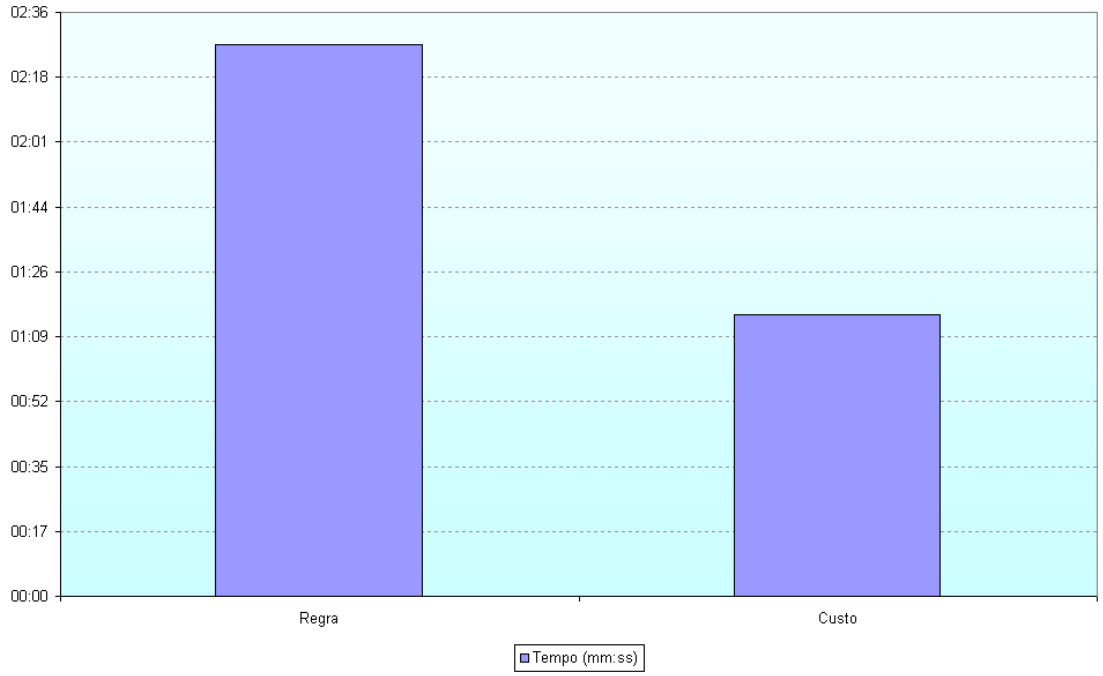


Figura 8 – Index skip scan: gráfico comparativo de tempo "Regra versus Custo"

Os testes comprovaram que o uso do otimizador por custo é mais eficiente que o otimizador por regra em casos de filtros aplicados em colunas posicionadas em sub-níveis de índices: em média, foram exigidos 98,4% menos leituras e 44,4% menos tempo para execução

da consulta. Essa diferença ocorre, pois o otimizador por custo baseado nas suas estatísticas previamente coletadas diferencia o caso que é menos custoso: acessar a tabela por índice via sub-níveis ou efetuar a leitura total da tabela. Nos casos onde existam índices com mais de uma coluna, onde cada coluna representa um nível, e consultas com filtros em colunas que não estão no primeiro nível do índice, o otimizador por custo permite que seja usado o índice varrendo os sub-níveis, sendo que o custo do uso desse índice é proporcional à quantidade de valores distintos nos níveis anteriores. No exemplo, como a coluna VALUE1 foi carregada somente com valores de 1 a 100, será necessário para percorrer todo o segundo nível do índice SKIPSCAN_VALUES_IDX efetuar 100 pesquisas distintas com valor do filtro aplicado na coluna VALUE2.

HASH JOIN

Para o teste, foi usada a tabela no formado do diagrama E-R encontrado na figura 9.

Figura 9 – Hash join: diagrama E-R das tabelas usadas para o teste

Na tabela HASHJOIN1 foram carregadas 1.511.010 linhas, onde a coluna PK contém valor seqüencial de 0 a 1.511.009 e a coluna VALUE contém valores de 1 a 5.000. Na tabela HASHJOIN2 foram carregadas 1.511.010 linhas, onde a coluna PK contém valor seqüencial de 0 a 1.511.009, a coluna VALUE contém valores de 1 a 5.000 e a coluna FK contém uma referência para cada linha da tabela HASHJOIN1, ou seja, valor seqüencial de 0 a 1.511.009. Os seguintes testes foram efetuados, gerando os seguintes planos de execução:

- União de duas tabelas médias e otimizador por regra:

SELECT * FROM HASHJOIN1 HJ1, HASHJOIN2 HJ2 where HJ2.FK = HJ1.PK;

```
-----------------------------------------------------
| Id  | Operation                     | Name        |
-----------------------------------------------------
|   0 | SELECT STATEMENT              |             |
|   1 |  NESTED LOOPS                 |             |
|   2 |   TABLE ACCESS FULL           | HASHJOIN2   |
|   3 |   TABLE ACCESS BY INDEX ROWID| HASHJOIN1   |
|*  4 |    INDEX UNIQUE SCAN          | SYS_C005505 |
-----------------------------------------------------
```

Obs: SYS_C005505 é o índice criado automaticamente para a chave primária - coluna PK da tabela HASHJOIN1.

- União de duas tabelas médias e otimizador por custo:

SELECT * FROM HASHJOIN1 HJ1, HASHJOIN2 HJ2 where HJ2.FK = HJ1.PK;

```
----------------------------------------
| Id  | Operation          | Name      |
----------------------------------------
|   0 | SELECT STATEMENT   |           |
|*  1 |  HASH JOIN         |           |
|   2 |   TABLE ACCESS FULL| HASHJOIN1 |
|   3 |   TABLE ACCESS FULL| HASHJOIN2 |
----------------------------------------
```

O resultado dos testes, com o número de blocos necessários para o processamento da consulta e tempo de execução encontra-se nos gráficos das figuras 10 e 11.

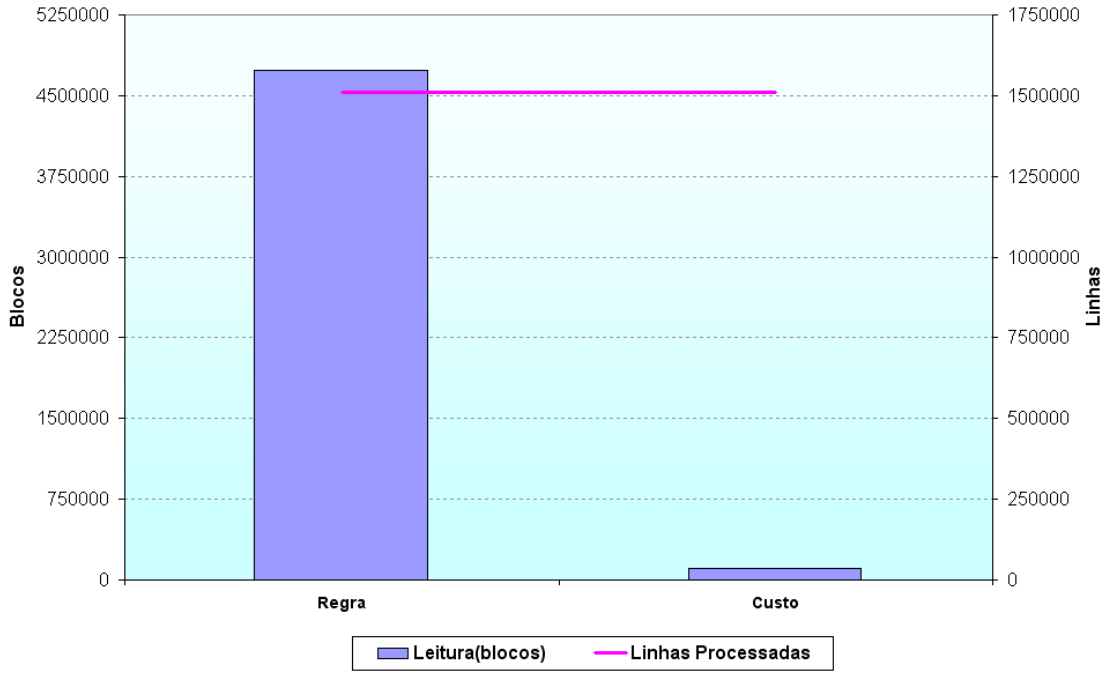


Figura 10 – Hash join: gráfico comparativo de leituras “Regra versus Custo”

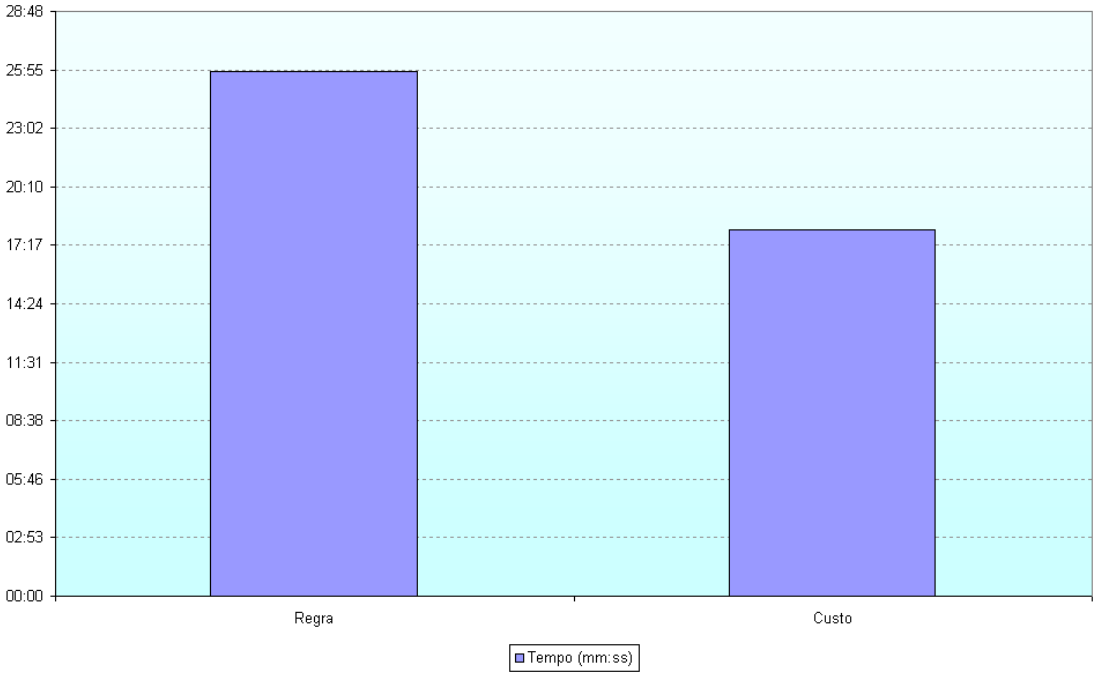


Figura 11 – Hash join: gráfico comparativo de tempo “Regra versus Custo”

Os testes comprovaram que o uso do otimizador por custo é mais eficiente que o otimizador por regra em casos de união de conjuntos de tamanho médio e grande: em média, foram exigidos 97,7% menos leituras e 30,1% menos tempo para execução da consulta. Essa diferença ocorre, pois o otimizador por custo baseado nas suas estatísticas previamente coletadas diferencia o caso que é menos custoso: unir a tabela via nested loops, hash join ou merge join. Normalmente o método nested loops é o melhor para união de pequenos conjuntos, mas para união de grandes conjuntos ele é ruim: para cada linha da tabela externa, a tabela ou índice interno inteiro terá que ser lido. Nesses casos o hash join é mais eficiente: os dois conjuntos são lidos integralmente somente uma vez. O merge join é melhor que o hash join em casos específicos, como em consultas onde há junção de conjuntos com ordenação pela chave da união, pois este método ordena os conjuntos para unir, não sendo necessário uma tarefa de ordenação posterior.

HISTOGRAMAS

Para o teste, foi usada a tabela no formato do diagrama E-R encontrado na figura 12.

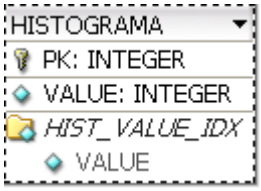


Figura 12 – Histograma: diagrama E-R da tabela usada para o teste

Nesta tabela foram carregadas 2.063.088 linhas, onde a coluna PK contém valor seqüencial de 0 a 2063087 e a coluna VALUE contém valores de 1 a 50 com a distribuição de valores demonstrada no gráfico na figura 13.

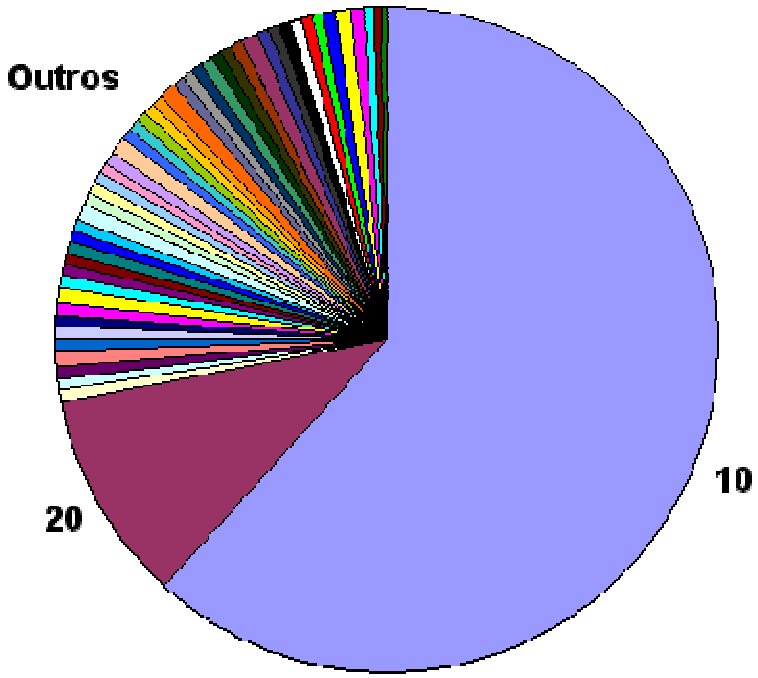


Figura 13 – Histograma: distribuição dos dados na coluna VALUE

Com a distribuição não uniforme dos dados comprovada acima, o uso do otimizador por custo e a coleta de histograma deve resultar em resultados significantes, para tal, os seguintes testes foram efetuados, gerando os seguintes planos de execução:

- Filtro com valor de baixa freqüência no campo da tabela e otimizador por regra:

SELECT * FROM HISTOGRAMA where VALUE = 30;

```
-----------------------------------------------------
| Id  | Operation                    | Name         |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |              |
|   1 |  TABLE ACCESS BY INDEX ROWID| HISTOGRAMA    |
|*  2 |   INDEX RANGE SCAN           | HIST_VALUE_IDX |
-----------------------------------------------------
```

- Filtro com valor de alta freqüência no campo da tabela e otimizador por regra:

SELECT * FROM HISTOGRAMA where VALUE = 10;

```
-----------------------------------------------------
| Id  | Operation                    | Name           |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |                |
|   1 |  TABLE ACCESS BY INDEX ROWID| HISTOGRAMA     |
|*  2 |   INDEX RANGE SCAN           | HIST_VALUE_IDX |
-----------------------------------------------------
```

- Filtro com valor de baixa freqüência no campo da tabela e otimizador por custo com histograma coletado para o campo VALUE:

SELECT * FROM HISTOGRAMA where VALUE = 30;

```
-----------------------------------------------------
| Id  | Operation                    | Name           |
-----------------------------------------------------
|   0 | SELECT STATEMENT             |                |
|   1 |  TABLE ACCESS BY INDEX ROWID| HISTOGRAMA     |
|*  2 |   INDEX RANGE SCAN           | HIST_VALUE_IDX |
-----------------------------------------------------
```

- Filtro com valor de alta freqüência no campo da tabela e otimizador por custo com histograma coletado para o campo VALUE:

SELECT * FROM HISTOGRAMA where VALUE = 10;

```
---------------------------------------
| Id  | Operation         | Name      |
---------------------------------------
|   0 | SELECT STATEMENT  |           |
|*  1 |  TABLE ACCESS FULL| HISTOGRAMA |
---------------------------------------
```

O resultado dos testes, com o número de blocos necessários para o processamento da consulta e tempo de execução encontra-se nos gráficos das figuras 14 e 15.

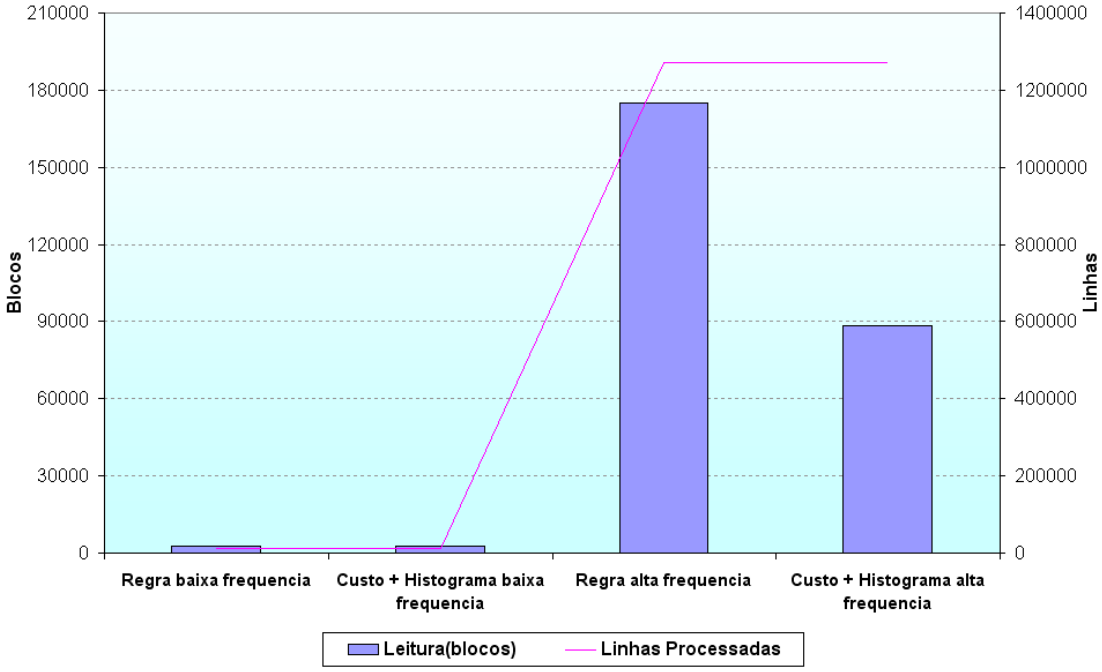


Figura 14 – Histograma: gráfico comparativo de leituras "Regra versus Custo"

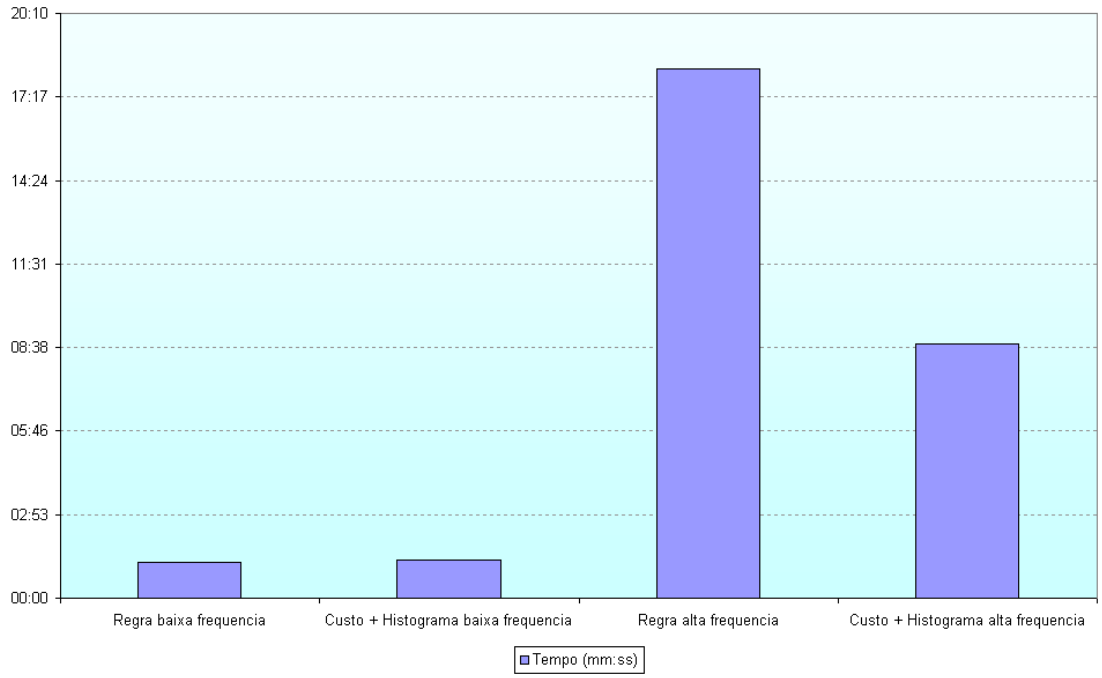


Figura 15 – Histograma: gráfico comparativo de tempo "Regra versus Custo"

Os testes comprovaram que o uso do otimizador por custo com histograma é mais eficiente que o otimizador por regra em casos de filtros aplicados em colunas com distribuição de valores não uniformes: em média, foram exigidos 44,56% menos leituras e 52% menos tempo para execução da consulta.

Essa diferença ocorre, pois o otimizador por custo baseado nas informações dos histogramas diferencia o caso que é menos custoso: acessar a tabela por índice ou efetuar a leitura total da tabela. Nos casos onde existam valores com uma quantidade significativa de freqüência os acessos por leituras simples na tabela gerados pela consulta no índice tornam-se tão custosos que é mais rápido ler a tabela na íntegra. No otimizador por regra o índice, abordagem menos eficiente neste caso, sempre será usado quando disponível.

3. Conclusão

A elaboração desse trabalho deixou evidente que o conhecimento do funcionamento interno do otimizador do SGBD utilizado é obrigatório e imprescindível para profissionais que desejem trabalhar com esses softwares. Com a evolução dos SGBDs a tendência é que vários pontos de configuração atualmente obrigatórios sejam automatizados e as alterações nos otimizadores para resolver problemas antes tratados pelos usuários é o ponto principal dessa evolução. Os experimentos aplicados ao SGBD Oracle, em forma de comparação entre os algoritmos de otimização (capítulo 2.1.4), evidenciaram que esse objetivo está sendo atingido.

4. Referências bibliográficas

LAHDENMAKI, Tapio. LEACH, Mike. Relational database index design and the optimizers: DB2, Oracle, SQL server. Wiley-Interscience, 2005.

GREEN, Connie Dialeris. et ali. Oracle9i Database Performance Tuning Guide and Reference, Release 2 (9.2). Oracle Corporation, 2002.

POWELL, Gavin. Oracle High Performance Tuning for 9i and 10g. Digital Press, 2003.

ORACLE MAGAZINE: Understanding Oracle SQL Optimization. Disponível em: <http://www.rampant-books.com/art_floss_oramag_sql_optimization.htm>. Acesso em 12/05/2008.