UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ANA CRISTINA DE OLIVEIRA CIRINO CODATO

PROCESSAMENTO DE CONSULTAS EM BANCOS DE DADOS DE DIFERENTES TECNOLOGIAS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Murilo Silva de Camargo

Florianópolis, novembro de 2001

PROCESSAMENTO DE CONSULTAS EM BANCOS DE DADOS DE DIFERENTES TECNOLOGIAS

ANA CRISTINA DE OLIVEIRA CIRINO CODATO

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Fernando O. Gauttier
Coordenador do Curso

Banca Examinadora

Prof. Dr. Murilo Silva de Camargo
Professor Orientador

Prof. Dr. Vitório Bruno Mazzola

Prof. Dr. Rosvelter João Coelho da Costa

"O meu interesse está no futuro, pois é lá que eu vou passar o resto da minha vida". (Charles Howard)

DEDICATÓRIA

Ofereço este trabalho aos meus avós José de Oliveira e Odethe Alves de Oliveira, pois não mediram esforços para que eu pudesse alcançar essa importante conquista.

AGRADECIMENTOS

Minha gratidão a Deus pela força e coragem encontradas nele.
Aos meus pais e avós, que nos momentos de fraqueza souberam me impulsionar ao meu objetivo.
Ao meu marido e ao meu filho pela paciência e dedicação.
Ao Professor Murilo pela orientação e considerações às minhas dúvidas.
Aos meus amigos pelo apoio e crescimento mútuo.

SUMÁRIO

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

RESUMO

Esta dissertação tem como objetivo a análise do Processamento de Consultas em Bancos de Dados de diferentes tecnologias, assim como, realizar um comparativo entre elas. Foram abordadas as seguintes tecnologias de Banco de Dados: Centralizado, Distribuído, Orientado a Objeto, Objeto-Relacional, Móveis e Multimídia.

Cada tecnologia de Banco de Dados pesquisada nesta dissertação busca atender uma necessidade de mercado. Devido às particularidades das várias tecnologias do banco de dados, observa-se que o processamento de consultas possui aspectos relevantes que são tratados diferencialmente em cada abordagem.

No decorrer do trabalho foram relatadas algumas características, pontos relevantes e restrições de cada Banco de Dados, ressaltando sempre o processamento de consultas e suas etapas. Para concluir o trabalho foram realizados estudos comparativos entre as diversas aplicações abordadas, enfatizando seus principais aspectos.

PALAVRAS-CHAVE: Processamento de Consultas, Otimização, Banco de Dados Centralizado, Banco de Dados Distribuído, Banco de Dados Orientado a Objeto, Banco de Dados Objeto-Relacional, Banco de Dados Móveis e Banco de Dados Multimídia.

ABSTRACT

This dissertation has as objective the analysis of the Query Processing in databases of different technologies, as well as, to accomplish a comparative one among them. The following database technologies were approached: Centralized, Distributed, Object-Oriented, Relational-object, Mobile and Multimedia.

Each database technology researched in this dissertation looks for to assist a market need. Due to the particularities of the several technologies of the database, it is observed that the query processing possesses important aspects that are treated differently in each approach.

In elapsing of the work some they were told characteristics, important points and restrictions of each database, always standing out the query processing and its stages. To conclude the work comparative studies they were accomplished among the several approached applications, emphasizing its main aspects.

KEYWORDS: *Query Processing, Optimization, Centralized Database, Distributed Database, Object-Oriented Database, Relational-object Database, Mobile Database and Multimedia Database.*

CAPÍTULO I

INTRODUÇÃO

1.1 Apresentação

A nova geração de Banco de Dados busca dotar adaptabilidade à velocidade das transformações que ocorrem em todos os setores da vida computacional ou informatizada, conferindo uma crescente importância para a habilidade de armazenar, gerenciar e recuperar dados. O surgimento desta nova geração de Banco de Dados tem motivado e até provocado uma vontade maior de pesquisar e conhecer novas tecnologias.

O desafio do mundo computacional é poder armazenar, gerenciar e recuperar os dados, de forma precisa, completa, econômica, flexível, confiável, relevante, simples e em tempo verificável.

O processamento de consultas em banco de dados, seja qual for sua aplicação, também faz parte dessa vantajosa e crescente evolução. A cada aparecimento de uma nova tecnologia, adaptações devem ser feitas, como por exemplo, o surgimento da INTERNET.

O objetivo deste trabalho é realizar um estudo comparativo entre as diversas tecnologias de Bancos de Dados, explorando o processamento de consultas de cada uma delas. Os Bancos de Dados estudados foram: Bancos de Dados Centralizados, Bancos de Dados Distribuídos, Bancos de Dados Orientados a Objeto, Bancos de Dados Objeto-relacionais, Bancos de Dados Móveis e Banco de Dados Multimídia.

Neste estudo comparativo houve uma preocupação em apontar, entre as abordagens escolhidas, os pontos em comum e as particularidades do processamento de consultas de cada tecnologia.

1.2 Metodologia

A metodologia utilizada para o desenvolvimento desta dissertação foi através de pesquisas bibliográficas, análise do conteúdo de artigos científicos, relatórios técnicos, documentações de sistemas e levantamentos comparativos. As fontes principais de pesquisa são (Elmasri e Navathe, 2000), (Date, 2000), (Silberschatz et al., 1999), (Özsu e Valduiez,1999), (Ramakrishnan, 1998), (Khoshafian, 1994), *sites* da Internet, tutoriais, *workshop*, documentos oficiais de congressos, artigos e relatórios técnicos.

1.3 Organização do Trabalho

A organização do trabalho está dividida no formato a seguir. O segundo capítulo define o processamento de consultas em banco de dados, descrevendo detalhadamente todas as suas etapas, é neste capítulo que o Banco de Dados Centralizado é enfatizado. Em todos os capítulos subseqüentes há uma preocupação em abordar o processamento de consultas. O terceiro capítulo comenta sobre Banco de Dados Distribuído, ressaltando as características, as vantagens e as restrições dessa tecnologia. Mais adiante, no quarto capítulo, é apresentado Banco de Dados Orientado a Objeto onde o armazenamento de dados ganha uma atenção especial. Os aspectos de Banco de Dados Objeto-relacional estão descritos no quinto capítulo. Já Banco de Dados Móveis e suas aplicações apresentam-se no sexto capítulo. O sétimo capítulo trata de Banco de Dados Multimídia. O oitavo capítulo aponta as comparações entre as tecnologias abordadas. Finalmente, o nono capítulo apresenta o resumo da dissertação desenvolvida, as conclusões obtidas e as perspectivas futuras relacionadas ao assunto.

CAPÍTULO II

PROCESSAMENTO DE CONSULTAS EM BANCO DE DADOS

O processamento de consultas é a atividade responsável em extrair informações de um banco de dados. Quando uma consulta é submetida ao sistema, ela deve ser traduzida e transformada em uma seqüência de operações que possa ser efetivamente executada pelo sistema computacional. No entanto, essa tradução permite que o sistema obtenha uma seqüência de comandos otimizados que possibilitam o processamento da consulta de uma forma eficiente.

Este capítulo define e descreve as etapas do processamento de consultas em ambiente centralizado.

2. Banco de Dados Centralizado

Os sistemas de banco de dados centralizados têm como característica o processamento e o armazenamento dos dados num único servidor de banco de dados, isto é, num único sistema computacional, não interagindo com outros. A CPU e os dispositivos de controle podem trabalhar concorrentemente, competindo pelo acesso à memória [Silberschatz et. al., 1999]. O banco dados centralizado é dividido nos seguintes esquemas: a) esquema interno: verifica os métodos de acesso e a estrutura do banco de dados; b) esquema conceitual: prevê como os dados serão organizados

logicamente e garante a integridade dos mesmos; e c) esquema externo: especifica quais diretórios e/ou dados, num determinado grupo de usuários, poderão ser acessados.

Em sistema de banco de dados centralizado todos os passos do sistema, não necessariamente, têm que ser centralizado. Por exemplo, o acesso ao banco de dados, solicitado pelos usuários distribuídos ao longo dos sites da rede, não descaracteriza o banco de dados de ser centralizado.

Em banco de dados centralizado um único Sistema Gerenciador de Banco de Dados (SGBD) é utilizado. O Sistema Gerenciador de Banco de Dados Centralizado (SGBDC) tem como função: a) controlar, através de uma rede de computadores, os acessos realizados; b) gerenciar, através de transações, os processos enviados pelos usuários; c) gerenciar a recuperação de banco de dados no caso de falhas; d) gerenciar a memória utilizada por esses processos; e) fazer o armazenamento do dicionário de dados e banco de dados; f) processar comandos de linguagem de manipulação de dados (DML); g) controlar as transações; h) manter a integridade do sistema; e i) controlar a concorrência de acesso aos dados entre as múltiplas transações.

São várias as vantagens na utilização de um SGBDC, como o controle de segurança dos dados, a simplificação da integridade dos dados, a disponibilidade dos dados e seu gerenciamento. Além das vantagens, um SGBDC também possui algumas desvantagens, por exemplo, o tráfego de inúmeros processos, num determinado período de tempo, acarreta um gargalo de processos amortizando o tempo de resposta ao usuário.

2.1 Etapas do Processamento de Consultas

As etapas do processamento de consultas são ilustradas na figura 2.1.

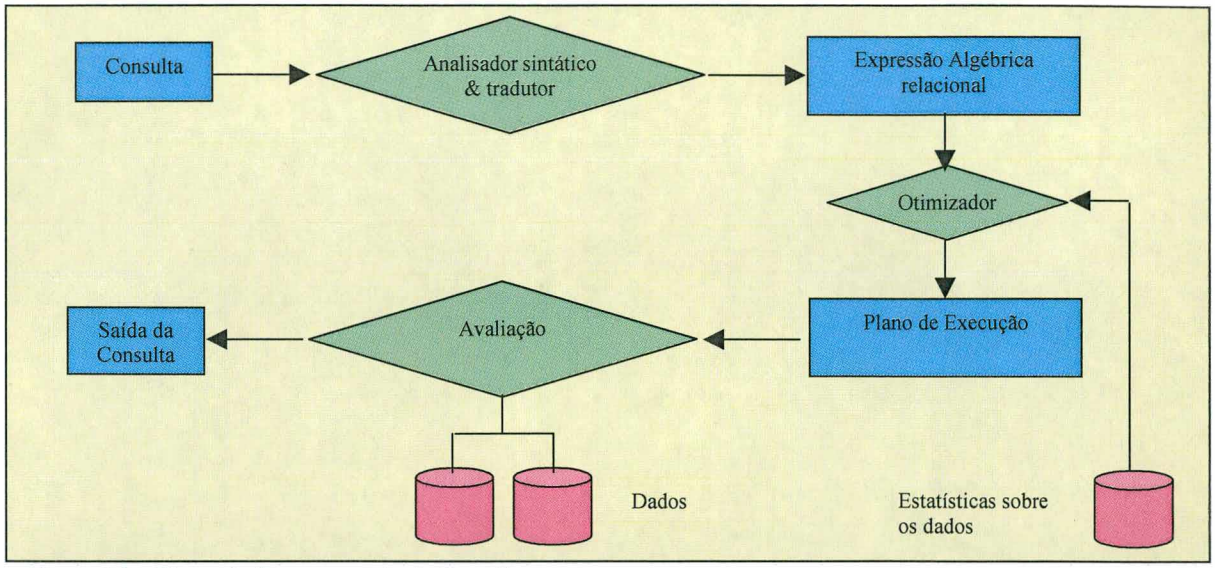

Figura 2.1 – Etapas do Processamento de Consultas
Fonte: SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. *Sistema de banco de dados.* São Paulo: Makron Books, 1999. 3. ed. 778 p.

O processamento de consultas é feito pelo SGBD, ele usa algumas técnicas para processar, otimizar e executar consultas de alto nível.

As linguagens de banco de dados relacionais permitem expressar consultas complexas de uma forma sucinta e simples [Özsu e Valduriez, 1999]. Na construção da resposta de uma consulta submetida, o usuário não especifica como os dados serão acessados, esta tarefa é função do processador de consulta, módulo específico do SGBD. Este processador dispensa o usuário da otimização da consulta.

As etapas do processamento de consulta, que serão descritas no decorrer desse capítulo, estão baseadas em Banco de Dados Centralizado. As próximas seções enfatizam as etapas do processamento de consultas, desde a tradução da linguagem não-procedural em linguagem procedural até a escolha do melhor plano de avaliação.

2.1.1 Tradução da linguagem não-procedural em linguagem procedural

Inicialmente deve-se conhecer e identificar as diferenças entre a linguagem não-procedural e a linguagem procedural. A linguagem não-procedural é aquela que dispensa a necessidade de indicar os procedimentos de como executar uma consulta, já a

linguagem procedural, todos os procedimentos para obter o resultado de uma consulta devem ser descritos passo a passo.

O cálculo relacional é uma linguagem não-procedural, isto é, permite descrever o jogo de respostas sem explicitar como eles deveriam ser computados [Ramakrishnan, 1998]. O cálculo relacional, por sua viabilidade, possui grande influência na evolução das linguagens de consultas comerciais, como por exemplo, a SQL.

Já numa consulta representada através da álgebra relacional, todos os procedimentos são descritos passo a passo, até chegar na resposta desejada, sempre baseando-se na ordem que os operadores são aplicados na consulta [Silberschatz et al., 1999]. Por este motivo, a álgebra relacional pode ser dita procedural. As consultas em álgebra relacional são compostas por uma coleção de operadores, onde a seleção (σ), projeção (π), união ($\cup$), produto cartesiano ($\times$) e diferença (-) são alguns dos operadores básicos. Uma propriedade fundamental da álgebra relacional é que todo operador aceita uma ou duas instâncias de relação como argumento e retorna uma instância de relação como resultado [Ramakrishnan, 1998]. A representação interna da consulta é normalmente criada na forma de grafos de consultas ou árvores de consultas. Esta representação interna é baseada na álgebra relacional, que é um formalismo matemático passível de otimização, e portanto mais útil ao SGBD do que a SQL.

Com base nas informações acima pode-se iniciar as explicações de como é realizada a transformação do cálculo relacional em álgebra relacional.

Primeiramente uma consulta SQL submetida ao sistema deve ser analisada léxica, sintática e semanticamente [Elmasri e Navathe, 2000]. O analisador léxico (*scanner*) identifica a linguagem símbolo (*token*), por exemplo, palavras chaves da linguagem SQL, identifica também os nomes dos atributos e das relações que constam na consulta. O analisador sintático (*parser*) confere a sintaxe da consulta para determinar se ela foi construída de acordo com as regras sintáticas (regras gramaticais) da linguagem de consulta. A consulta também deve ser validada (*validated*), isto é, deverão ser conferidos todos os atributos e relações, verificando se são válidos semanticamente.

Na linguagem relacional, como a SQL, a transformação mais importante é a qualificação da consulta. Na análise pode haver rejeição das consultas onde o processo é impossível ou desnecessário. A razão principal desta rejeição é quando uma consulta é

do tipo incorreta ou semanticamente incorreta. Quando descoberto um desses casos, a consulta é devolvida ao usuário com uma explicação, caso contrário, continua o processamento da consulta. Uma consulta é incorreta, se qualquer um de seus atributos ou nome da relação não forem definidos, ou ainda se estão sendo aplicadas operações para atributos do tipo errado. Existem técnicas que descobrem consultas incorretas. Uma consulta é semanticamente incorreta se os componentes não contribuem de qualquer forma à geração do resultado.

Depois que a consulta é analisada ela deve ser decomposta em uma coleção de blocos, onde cada bloco é traduzido em operadores algébricos e posteriormente otimizado [Elmasri e Navathe, 2000]. A decomposição da consulta tem como função transformar o cálculo relacional em álgebra relacional. Numa típica consulta relacional, o otimizador preocupa-se em otimizar um bloco de cada vez [Özsu e Valduriez, 1999]. A decomposição da consulta pode ser dividida em quatro passos: a) o cálculo da consulta tem que ser normalizado e satisfatório para a manipulação subseqüente. A normalização dá prioridade ao operador lógico; b) a consulta normalizada é analisada semanticamente de forma que as consultas incorretas sejam rejeitadas assim que possível; c) a consulta correta, ainda na forma do cálculo relacional, é simplificada; e d) o cálculo da consulta é reestruturado como uma consulta algébrica [Özsu e Valduriez, 1999].

Para rescrever uma consulta em álgebra relacional, são necessários dois passos: a) transformação direta da consulta de cálculo relacional em álgebra relacional; e b) reestruturação da consulta em álgebra relacional para melhorar o desempenho.

Todo bloco de consulta SQL pode ser expressado através da seguinte forma: a cláusula *SELECT* corresponde ao operador de projeção(π), a cláusula *WHERE* corresponde ao operador de seleção(σ), a cláusula *FROM* corresponde ao produto cartesiano das relações ($\times$) e as cláusulas restantes são traçadas aos operadores correspondentes de uma maneira direta [Ramakrishnan, 1998].

Um bloco de consulta é uma consulta SQL sem ser aninhada e contém expressões *SELECT-FROM-WHERE*. Cada bloco possui exatamente uma cláusula *SELECT*, uma cláusula *FROM* e no máximo uma cláusula *WHERE*. Toda consulta SQL pode ser decomposta em uma coleção de blocos de consultas não aninhados, caso

ocorram consultas aninhadas, serão identificadas em blocos separados. [Ramakrishnan, 1998]

Considere a seguinte consulta em SQL na relação EMPREGADO:

```
SELECT   LNOME, FNOME
FROM     EMPREGADO
WHERE    SALÁRIO > (SELECT   MAX (SALÁRIO)
                    FROM     EMPREGADO
                    WHERE    DNO = 5);
```

Esta consulta inclui uma subconsulta aninhada e conseqüentemente deve ser decomposta em dois blocos. Primeiramente, o bloco interno seria:

```
(SELECT   MAX (SALÁRIO)
 FROM     EMPREGADO
 WHERE    DNO = 5)
```

e o bloco externo seria:

```
SELECT   LNOME, FNOME
FROM     EMPREGADO
WHERE    SALÁRIO > c
```

onde **c** representa o resultado encontrado no bloco interno.

Com relação ao bloco interno, a tradução da consulta SQL (cálculo relacional) em algébrica relacional poderia ser a seguinte:

$\Im_{\text{MAX SALÁRIO}}(\sigma_{\text{DNO=5}}(\text{EMPREGADO}))$

Com relação ao bloco externo, a tradução poderia ser:

$\pi_{\text{LNOME, FNOME}}(\sigma_{\text{SALÁRIO>C}}(\text{EMPREGADO}))$.

Onde $\Im$ corresponde a função agregada (*aggregate functions*), σ a operação de seleção (*select operation*) e π a operação de projeção (*project operation*).

Depois que a consulta submetida ao sistema é transformada, ela deve ser otimizada, para que o melhor plano de execução seja escolhido. A próxima seção trata especificamente da otimização de consultas, comenta os aspectos existentes para que uma consulta seja otimizada e um melhor plano de avaliação seja escolhido.

2.1.2 Otimização

Otimização é o processo de selecionar o plano de avaliação mais eficiente de uma consulta. Otimizar nada mais é do que localizar uma estratégia que se aproxima do ótimo. Na otimização da consulta, vários planos equivalentes por consulta são gerados e a partir daí escolhe-se o menos oneroso dentre eles. Vale a pena para o sistema, gastar um certo tempo na seleção de uma estratégia boa ao processar determinada consulta, até mesmo se ela for executada uma única vez [Elmasri e Navathe, 2000].

O SGBD deve definir uma estratégia de execução para recuperar o resultado da consulta dos arquivos do banco de dados. Uma estratégia de execução é um plano para executar a consulta, acessar os dados e armazenar os resultados temporários. O otimizador de consultas (*query optimizer*) é um módulo do SGBD responsável pela definição de uma estratégia de execução e o gerador de código (*code generator*) é responsável por gerar o código executável para esta estratégia. Este código é então passado ao módulo de execução (*runtime*), responsável por executá-lo, de maneira compilada ou interpretada, para produzir o resultado da consulta. Caso o resultado corresponda a um erro de execução, uma mensagem de erro é gerada.

Nem sempre a otimização, encontra um plano de execução ótimo, em alguns casos uma estratégia boa é suficiente para executar uma consulta. Achar uma ótima estratégia é normalmente muito demorado mesmo executando uma simples consulta.

A aproximação da procura exaustiva é freqüentemente usada na otimização, sendo consideradas todas as possíveis estratégias de execução. Tenta-se achar uma solução ótima, não necessariamente a melhor, pois evita o alto custo de otimização, em termos de memória e consumo de tempo [Özsu e Valduriez, 1999]. A otimização visa minimizar o número de acessos à memória secundária (disco) e a quantidade de memória principal necessária para processar as consultas.

A otimização pode ser feita estaticamente antes de executar a consulta ou dinamicamente com a consulta executada [Özsu e Valduriez, 1999]. Otimização de consulta estática acaba no momento da compilação da consulta. Assim o custo de otimização pode ser amortizado em cima da execução de múltiplas consultas. Esta contagem de tempo é apropriada para o uso do método de procura exaustiva. A

vantagem principal da otimização de consulta estática é que os tamanhos atuais das relações temporárias estão disponíveis ao processador de consulta e minimizam a probabilidade de uma escolha ruim. Otimização de consulta dinâmica procede no momento da execução da consulta. A escolha da melhor operação pode estar baseada no conhecimento preciso dos resultados das operações executadas previamente [Özsu e Valduriez, 1999]. Otimização de consulta híbrida tenta prover as vantagens da otimização de consulta estática enquanto evita os assuntos gerados por estimativas inexatas.

A efetividade de otimização de consulta confia em estatísticas do banco de dados. Otimização de consulta dinâmica requer estatísticas para escolher quais operações deveriam ser feitas primeiro. A precisão das estatísticas é alcançada atualizando-as periodicamente.

Nas seções subseqüentes serão discutidas, as duas principais técnicas para implementar a otimização de consulta. A primeira técnica está baseada em regras de heurísticas, regras estas que trabalham eficientemente na maioria dos casos, mas não são garantidas em todos os possíveis casos, pois elas reordenam as operações tipicamente numa árvore de consulta. A segunda técnica envolve o cálculo de custo de estratégias de execução e escolhe o plano de execução com baixa estimativa de custo. As duas técnicas normalmente são combinadas em um otimizador de consultas. Uma desvantagem da otimização baseada no custo é o custo da própria otimização. Embora o custo do processamento da consulta possa ser reduzido por meios de otimização inteligentes, a otimização baseada em custo ainda é cara. Conseqüentemente, muitos sistemas usam a heurística para reduzir o número de alternativas que podem ser escolhidas em uma abordagem baseada no custo [Silberschatz et al., 1999].

2.1.2.1 Heurística

A heurística é uma das opções que possibilita a otimização de uma consulta, ela utiliza regras que modificam a representação interna da consulta e melhoram o desempenho esperado para a execução dessa consulta.

Quando ocorre o uso da heurística, primeiramente, o *parser* de uma consulta de alto-nível gera a representação interna inicial seguindo de acordo com as regras pré-estabelecidas de heurísticas. Resumidamente, as regras de heurística são: a) executar operações de seleção o mais cedo possível, para eliminar tuplas que não preenchem a condição de seleção; b) combinar seleções com um produto cartesiano anterior, de forma a produzir uma junção natural, que é bem mais econômica do que um produto cartesiano; c) aplicar operações de projeção de forma antecipada, isto é, transferir ou inserir projeções sempre que necessário; d) aplicar operações em grupo à medida que se percorre as tuplas, pois evita a geração de muitas tabelas temporárias; e) procurar expressões comuns em uma árvore, devendo computá-las uma única vez e guardá-las em uma tabela temporária [Elmasri e Navathe, 2000].

Depois que o *parser* gerou a representação interna inicial, um plano de execução de consulta é criado para executar grupos de operações baseados em caminhos de acesso disponíveis nos arquivos envolvidos na consulta.

A árvore de consulta representa a entrada de relações da consulta através de nós e também representa as operações da álgebra relacional com nós internos. A execução da árvore de consulta consiste em executar operações de nós internos sempre que seus operandos estão disponíveis e substitui aquele nó interno pela relação resultante da execução da operação. A execução termina quando os nós da raiz são executados e produzem uma relação resultante para a consulta. Por exemplo: para todo projeto localizado em '*Stafford*', deverá ser recuperado o número do projeto, o número de controle do departamento, o último nome do gerente, o endereço e a data de aniversário. Esta consulta corresponde à seguinte expressão da álgebra relacional:

$$\pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(((\sigma_{\text{PLOCATION = 'Stanfford'}}(^{\text{PROJECT}})) \; |\times|_{\text{D.NUM = DNUMBER}}(^{\text{DEPARTMENT}})) \; |\times|_{\text{MGRSSN = SSN}}(^{\text{EMPLOYEE}}))$$

que corresponde à seguinte consulta SQL:

```
Q2:  SELECT   P.PNUMBER, P.DNUM, E.LNAME, E. ADDRESS, E.BDATE
     FROM     PROJECT AS P, DEPARTMENT AS D, EMPLOYEE AS E
     WHERE    P.DNUM = D.DNUMBER AND D.MGRSSN = E.SSN AND
              P.PLOCATION = 'Stafford';
```

As figuras 2.2 (a) e 2.2 (b) ilustram duas árvores de consultas para uma consulta Q2. A figura 2.2 (a) representa a árvore de consulta que corresponde à expressão da

álgebra relacional de Q2 e a figura 2.2 (b) ilustra a árvore de consulta para consultas SQL de Q2. A figura 2.2 (c) mostra o grafo da consulta Q2.

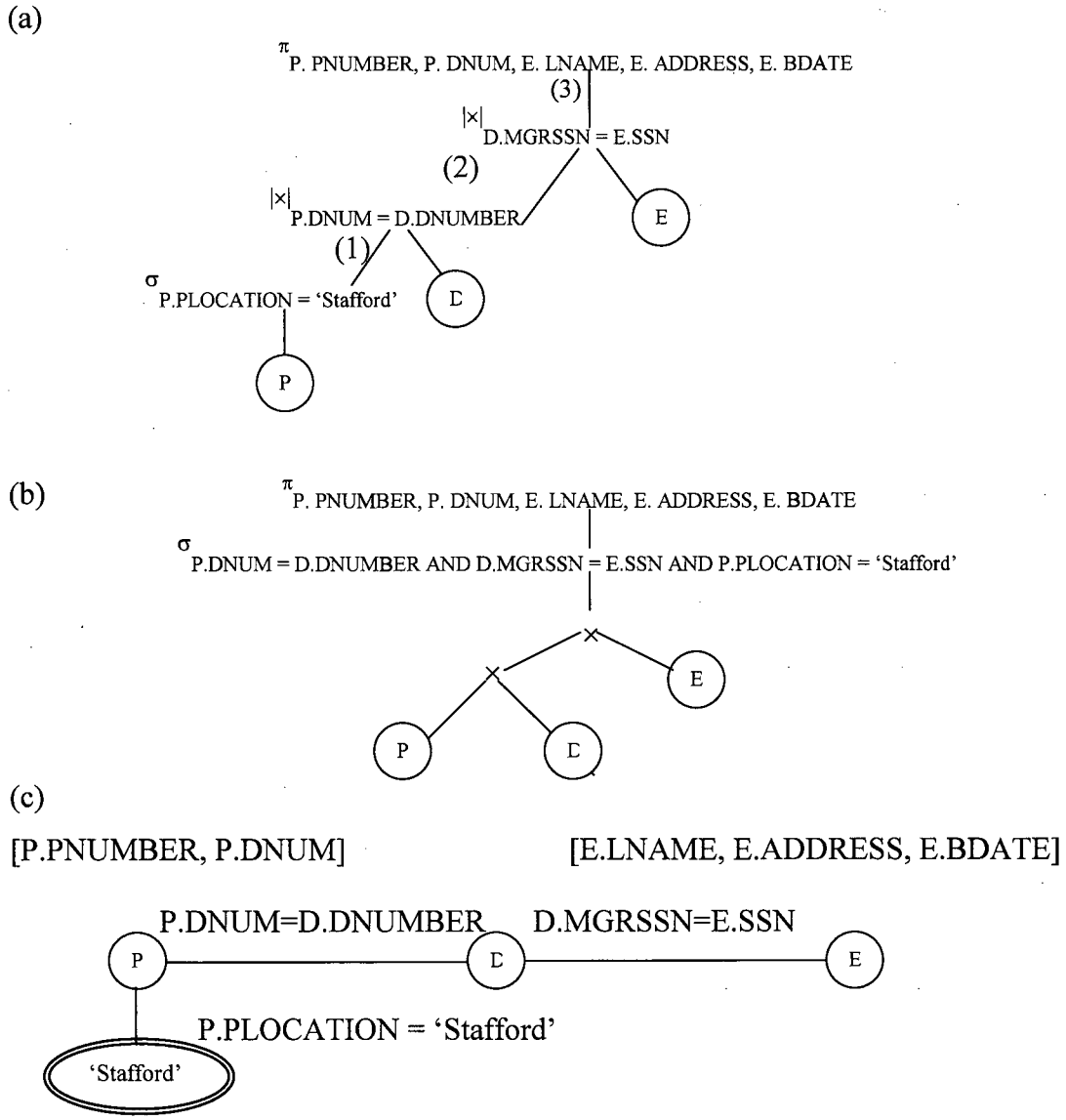

Figura 2.2 – Árvores e grafo de consulta. A figura (a) corresponde à árvore de consulta em álgebra relacional da consulta Q2. A figura (b) corresponde à árvore de consulta em SQL da consulta Q2. A figura (c) corresponde ao grafo da consulta Q2.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Na figura 2.2 (a) as três relações *PROJECT*, *DEPARTMENT* e *EMPLOYEE* são representadas através dos nós P, D e E, enquanto que as expressões das operações da álgebra relacional são representadas através de nós internos. Quando esta árvore de consulta é executada, o nó marcado como (1), tem que começar a executar antes do nó (2), porque algumas tuplas resultantes da operação (1) deverão estar disponíveis antes

de começar a executar a operação (2). Semelhantemente, o nó (2) tem que começar a executar e produzir resultados antes do nó (3), e assim por diante. Como pode-se perceber, a árvore de consulta representa uma ordem específica de operações para a execução de uma consulta. No grafo da consulta, figura 2.2 (c), as relações são representadas por nós que são exibidos com um único círculo. Os valores constantes, tipicamente da condição de seleção da consulta, é representado por nós constantes que são exibidos com dois círculos. As condições de seleção e junção são representadas pelas extremidades do gráfico. Finalmente, os atributos de cada relação são recuperados e exibidos entre parênteses sobre cada relação.

A representação do grafo da consulta não especifica uma ordem para indicar quais das operações serão executadas primeiro, há somente um único grafo que corresponde a determinada consulta. Embora algumas técnicas de otimização baseiam-se em grafos de consultas, geralmente aceita-se árvores de consultas porque, em prática, o otimizador de consulta precisa mostrar a ordem das operações de execução da consulta, desta maneira, em grafos de consulta, esta condição não acontece.

Geralmente, muitas expressões diferentes da álgebra relacional, e conseqüentemente muitas árvores de consultas diferentes, podem ser equivalentes, isto é, elas podem corresponder à mesma consulta. Neste caso, o *parser* da consulta gera uma árvore de consulta inicial padrão que corresponde a uma consulta SQL, sem fazer qualquer otimização. Por exemplo, para uma consulta com operações de seleção, projeção e junção, como Q2, a árvore inicial é ilustrada através da figura 2.2 (b), apresentada anteriormente.

O produto cartesiano das relações especificadas na cláusula *FROM* é aplicado primeiro, posteriormente são aplicadas as condições de seleção e junção da cláusula *WHERE*, em seguida aplica-se a projeção dos atributos da cláusula *SELECT*. Esta árvore de consulta inicial representa uma expressão da álgebra relacional que é muito ineficiente se executada diretamente, por causa das operações do produto cartesiano (×). Por exemplo, se as relações *PROJECT*, *DEPARTMENT* e *EMPLOYEE* tivessem tamanho de registro 100, 50 e 150 bytes e contivessem 100, 20 e 5000 tuplas, respectivamente, o resultado do produto cartesiano seria 10 milhões de tuplas com tamanho de registro de 300 bytes cada. Porém, a árvore de consulta da figura 2.2 (b) está em uma forma padrão simples que pode ser facilmente criada. A partir deste

momento é que começa o trabalho do otimizador de consulta, através da heurística, para transformar esta árvore de consulta inicial em uma árvore de consulta final com uma execução eficiente.

O otimizador tem que incluir regras de equivalência entre as expressões da álgebra relacional que podem ser aplicadas à árvore inicial. As regras da otimização da consulta por heurística, utilizam expressões de equivalência para transformar a árvore inicial em final, resultando em uma árvore de consulta otimizada.

Mais adiante será discutido informalmente como uma árvore de consulta pode ser transformada usando heurísticas, enfatizando a transformação das regras gerais e mostrando como elas podem ser usadas num otimizador de heurística.

Exemplificando a transformação de uma consulta, considere a seguinte consulta Q: " Ache os últimos nomes (*name*) dos empregados (*employee*) nascidos depois de 1957, que trabalham (*work*) num projeto (*project*) chamado '*Aquarius*'. Esta consulta pode ser especificada em SQL como segue:

```
Q:  SELECT   LNAME
    FROM     EMPLOYEE, WORKS_ON, PROJECT
    WHERE    PNAME = 'Aquarius' AND PNUMBER = PNO
             AND ESSN = SSN AND BDATE. '1957-12-31';
```

A árvore de consulta Q inicial é mostrada na figura 2.3 (a).

(a)

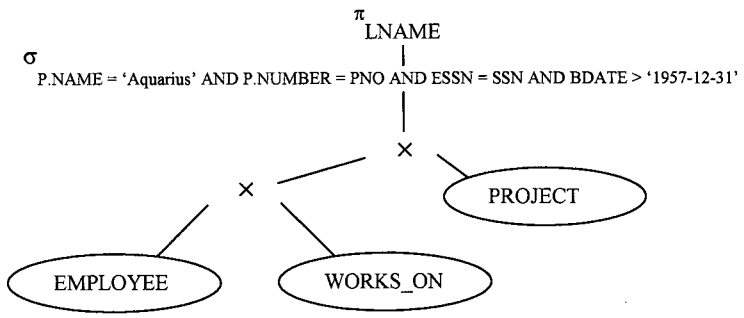

Figura 2.3 (a) – A árvore de consulta Q inicial .
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Executando diretamente esta árvore, primeiramente obtêm-se um arquivo bastante grande que contém o produto cartesiano dos arquivos *EMPLOYEE*, *WORKS_ON*, e *PROJECT*. Entretanto, esta consulta solicita somente um registro da

relação *PROJECT* (projeto *Aquarius*) e somente um registro da relação *EMPLOYEE* (data de nascimento 1957-12-31).

A figura 2.3 (b) exibe uma árvore de consulta melhorada. Inicialmente, nesta árvore, aplica-se as operações *SELECT* para reduzir o número de tuplas que aparecem no produto cartesiano.

(b)

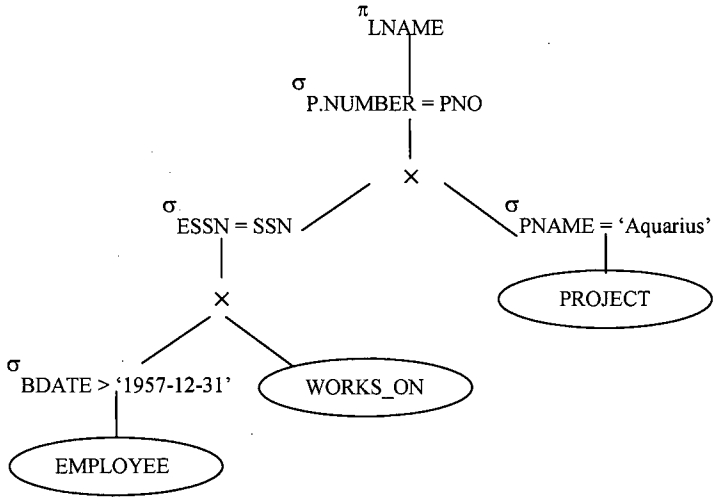

Figura 2.3 (b) – A árvore de consulta Q melhorada.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Para que uma melhora adicional possa ser obtida, é necessário trocar na árvore, as posições da relação *EMPLOYEE* e *PROJECT*, como mostrado na figura 2.3 (c).

(c)

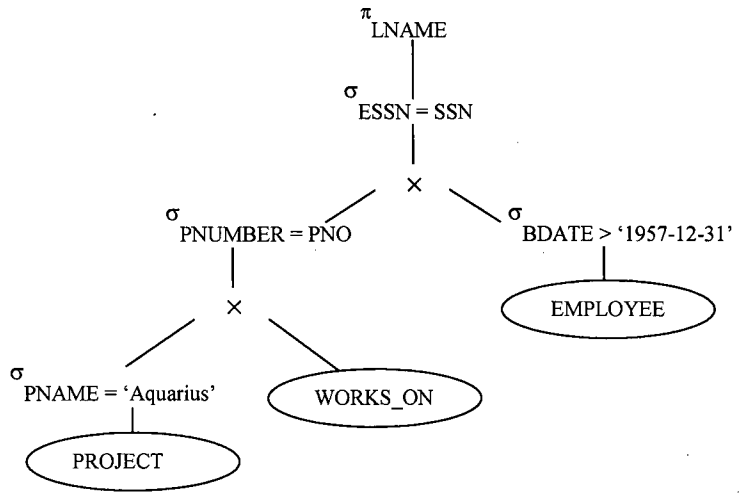

Figura 2.3 (c) – Passos da otimização da árvore de consulta Q.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Neste caso, a informação PNUMBER é um atributo chave da relação *PROJECT*, e conseqüentemente a operação de seleção da relação *PROJECT* recobrará um único

registro. Mais adiante pode-se melhorar a árvore de consulta, substituindo qualquer operação do produto cartesiano pela operação seguida de uma condição de junção, como exibido na figura 2.3 (d).

(d)

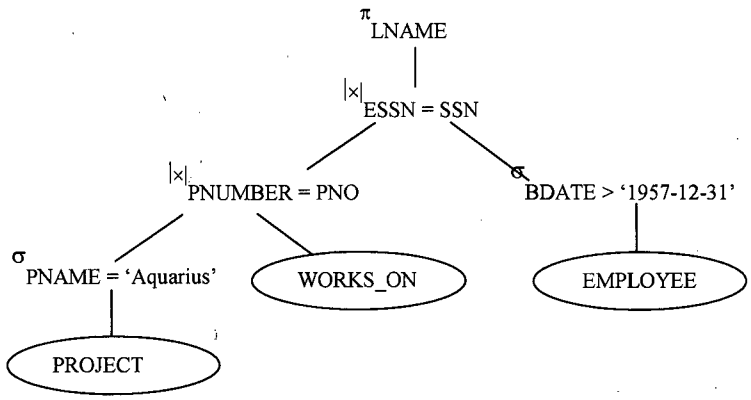

Figura 2.3 (d) – Passos do produto cartesiano e seleção com operações de junção.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

(e)

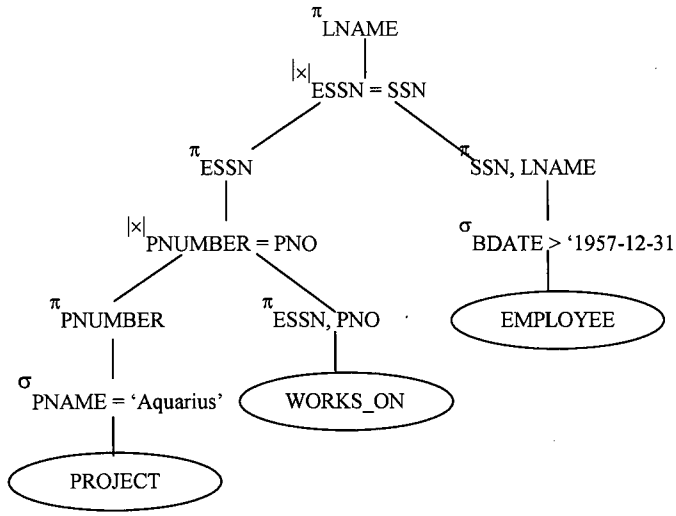

Figura 2.3 (e) – Passos para mover as operações de projeção da árvore de consulta.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Uma outra melhoria é manter somente os atributos que são necessários para as operações subseqüentes nas relações intermediárias, incluindo na árvore de consulta, assim que possível, operações de projeção (π), como na figura 2.3 (e), exibida anteriormente. Este processo reduz os atributos (colunas) das relações intermediárias, considerando que as operações de seleção (σ) reduzem o número de tuplas (registros).

Como o exemplo anterior demonstra, uma árvore de consulta pode ser transformada passo a passo em outra árvore de consulta com execução mais eficiente.

Porém, é necessário que se tenha absoluta certeza que os passos da transformação sempre conduzem a uma árvore equivalente.

Dada uma expressão da álgebra relacional, é função do otimizador de consulta propor um plano de avaliação da consulta que gere o mesmo resultado da expressão fornecida e que seja uma maneira menos onerosa de gerar o resultado (ou que, pelo menos, não seja muito mais cara que a maneira mais barata).

Para encontrar o plano de avaliação de consulta menos caro, o otimizador precisa gerar planos alternativos que produzam o mesmo resultado da expressão dada e escolher o plano menos caro [Silberschatz et al., 1999].

Para fazer isto, o otimizador de consulta têm que saber qual a regra de transformação que preserva esta equivalência. Portanto, a seguir, serão discutidas algumas destas regras de transformação. Preservar a equivalência significa que as relações geradas pelas duas expressões têm o mesmo conjunto de atributos e contêm o mesmo conjunto de tuplas, embora seus atributos possam estar ordenados de forma diferente.

Existem muitas regras para transformar as operações equivalentes da álgebra relacional. Neste momento, o que mais interessa é o significado das operações e as relações resultantes. Se duas relações têm ordens diferentes e possuem o mesmo jogo de atributos, ambas deverão representar a mesma informação, conseqüentemente considera-se que são equivalentes.

Algumas regras de transformação são significantes na otimização da consulta, como segue:

1. Cascata de seleção (σ): Uma operação de seleção conjuntiva pode ser rompida em uma seqüência de seleções individuais (significa uma sucessão):

   σ_{c1} e $_{c2}$ e ... e $_{cn}$ (R) $\equiv \sigma_{c1}(\sigma_{c2}(...(\sigma_{cn}(R))...))$

2. Comutatividade de seleção (σ): a operação de σ é comutativa:

   $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$

3. Cascata de projeção (π): Apenas as operações finais em uma seqüência de operações de projeção são necessárias, as demais podem ser omitidas.

   $\pi_{list1}(\pi_{list2}(...(\pi_{listn}(R))...)) \equiv \pi_{list1}(R)$

4. Comutando seleção (σ) com projeção (π): Se a condição de seleção c envolve somente os atributos A1, ..., An na lista de projeção, as duas operações podem ser comutadas: $\pi_{A1, A2, ..., An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1, A2, ..., An}(R))$
5. Comutatividade de junção ($|\times|$) e produto cartesiano ($\times$): a operação de $|\times|$ é comutativa, assim como a operação de $\times$: $R\ |\times|_c\ S \equiv S\ |\times|_c\ R \qquad R \times S \equiv S \times R$
6. Comutatividade de seleção (σ) com junção ($|\times|$) (ou produto cartesiano ($\times$)): Se todos os atributos da condição de seleção c envolvem somente os atributos de uma das relações que são unidas, correspondendo a R, as duas operações podem ser comutadas como segue: $\sigma_c(R \times S) \equiv (\sigma_c(R)) \times S$

   Alternativamente, se a condição de seleção c pode ser escrita como (c1 e c2), onde a condição c1 envolve somente os atributos de R e a condição c2 envolve somente os atributos de S, as operações são comutadas como segue:

   $\sigma_c(R \times S) \equiv (\sigma_{c1}(R)) \times (\sigma_{c2}(S))$

   As mesmas regras são aplicadas se a junção ($|\times|$) for substituída por uma operação de produto cartesiano ($\times$).
7. Comutando a projeção (π) com junção ($|\times|$) (ou produto cartesiano ($\times$)): Supondo que a lista de projeção é $L = \{A_1, ..., A_n, B_1, ... B_m\}$, onde $A_1, ..., A_n$ são atributos de R e $B_1, ..., B_m$ são atributos de S. Se a condição de junção c envolve somente atributos em L, as duas operações podem ser comutadas como segue:

   $\pi_L(R\ |\times|_c\ S) \equiv (\pi_{A1, ..., An}(R))\ |\times|_c\ (\pi_{B1, ..., Bm}(S))$

   Se a condição de junção c não contém atributos adicionais em L, estes devem ser somados à lista de projeção e será necessária uma operação de π final.
8. Comutatividade do conjunto de operações: os conjuntos das operações de união ($\cup$) e interseção ($\cap$) são comutativos.
9. Associatividade de junção ($|\times|$), produto cartesiano ($\times$), união ($\cup$) e interseção ($\cap$): Estas quatro operações são individualmente associativas, isto é, se θ representa qualquer uma destas quatro operações ao longo da expressão, tem-se:

   $(R\ \theta\ S)\ \theta\ T \equiv R\ \theta\ (S\ \theta\ T)$
10. Comutando seleção (σ) com conjunto de operações: A operação de σ comuta com $\cup$, $\cap$ e se θ representa qualquer uma destas três operações ao longo da expressão, tem-se: $\sigma_c(R\ \theta\ S) \equiv (\sigma_c(R))\ \theta\ (\sigma_c(S))$

11. A operação de projeção (π) comuta com união ($\cup$): $\pi_L (R \cup S) \equiv (\pi_L (R)) \cup (\pi_L (S))$
12. Convertendo uma seqüência de (σ, $\times$) em $|\times|$: Se a condição c de uma σ seguida de um $\times$ corresponde a uma condição de junção ($|\times|$), converta a seqüência (σ, $\times$) em $|\times|$, como segue: $(\sigma_c (R \times S)) \equiv (R |\times|_c S)$

Será apresentado logo abaixo, o esboço dos passos de um algoritmo que utiliza algumas das regras de transformação, descritas anteriormente, onde transforma uma árvore de consulta inicial em uma árvore de consulta otimizada, sendo que a execução é eficiente na maioria dos casos.

a) Utilizando a regra 1: neste caso, são interrompidas quaisquer operações de seleção com condições conjuntivas numa cascata de operações de seleção. Deste modo, é permitido um maior grau de liberdade, movendo as operações de seleção para baixo das diferentes ramificações da árvore.
b) Utilizando as regras 2, 4, 6 e 10: trata da comutatividade da seleção com outras operações. Move-se cada operação de seleção para o extremo da árvore de consulta assim como é permitido pelos atributos envolvidos na condição de seleção.
c) Utilizando as regras 5 e 9: estas regras correspondem à comutatividade e associatividade das operações binárias, reorganizando os nós da árvore e usando os seguintes critérios: i) coloca-se as relações do nó juntamente com as operações de seleção mais restritivas, sendo assim, serão executados na representação da árvore de consulta. A seleção mais restritiva pode ser definida como algo que produz uma relação com um número menor de tuplas ou com o tamanho menor. As operações de seleção mais restritivas, também possuem seletividade menor, isto é, maior praticidade, porque as estimativas de seletividade estão freqüentemente disponíveis no catálogo do SGBD; ii) é indispensável ter certeza que o nó ordenado não causa operações de produto cartesiano. Por exemplo, se as duas relações com a seleção mais restritiva não têm uma condição de junção direta entre elas, pode-se então mudar a ordem dos nós, evitando produtos de cartesianos.
d) Utilizando a regra 12: esta regra combina uma operação de produto cartesiano com uma operação de seleção subseqüente a uma operação de junção na árvore, isto se representar uma condição de junção.
e) Utilizando as regras 3, 4, 7 e 11: trata do cascateamento da projeção e a comutatividade da projeção com outras operações. Na utilização destas regras,

recomenda-se mover as listas de atributos de projeção, situados mais abaixo na árvore, criando novas operações de projeções necessárias. Somente os atributos resultantes da consulta em operações subseqüentes da árvore deveriam ser mantidos depois de cada operação de projeção.

f) Identificar as subárvores que representam os grupos de operações que podem ser executados por um único algoritmo.

No exemplo anterior, a figura 2.3 (b) exibe a árvore da figura 2.3 (a) depois de aplicado os passos 1 e 2 do algoritmo. A figura 2.3 (c) mostra a árvore depois do passo 3, a figura 2.3 (d) depois do passo 4 e a figura 2.3 (e) depois do passo 5. No passo 6, pode-se agrupar as operações da subárvore, cuja raiz é a operação π_{ESS}. Pode-se também agrupar as operações restantes numa outra subárvore onde as tuplas resultantes do primeiro algoritmo substituem a subárvore cuja raiz é a operação π_{ESS}, porque os meios iniciais de agrupamento desta subárvore é executado primeiro.

Depois de otimizada a árvore de consulta, é gerado um plano de execução para uma expressão de álgebra relacional. Este plano inclui informações sobre os métodos de acesso disponíveis para cada relação, como também os algoritmos a serem usados para computar os operadores relacionais representados na árvore. Como um exemplo simples, considere a seguinte consulta Q1, onde a expressão da álgebra relacional correspondente é:

$\pi_{FNAME, LNAME, ADDRESS}$ ($\sigma_{DNAME = 'RESEARCH'}$(DEPARTMENT)

$|\times|_{DNUMBER = DNO}$ EMPLOYEE)

A árvore de consulta é mostrada na figura 2.4.

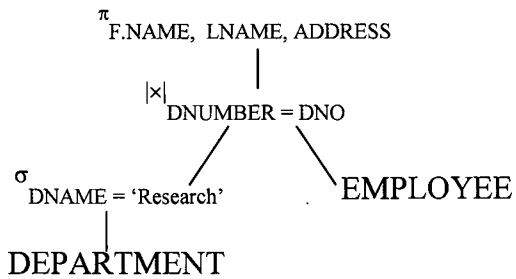

Figura 2.4 – Árvore de consulta para a consulta Q1.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Para converter esta árvore em um plano de execução o otimizador poderia escolher alguns algoritmos existentes. A aproximação escolhida para executar a consulta pode especificar uma avaliação materializada ou uma avaliação de *pipelined.*

Com a avaliação materializada, o resultado de uma operação é armazenado como uma relação temporária, ou seja, o resultado é materializado fisicamente. Por exemplo, a operação de junção pode ser computada e o resultado final armazenado como uma relação temporária, neste caso é lida como entrada do algoritmo que computa a operação de projeção, produzindo o resultado da consulta.

Por outro lado, com a avaliação de *pipelined*, as tuplas resultantes de uma operação são produzidas e remetidas diretamente à próxima operação na sucessão da consulta. Por exemplo, como as tuplas selecionadas do *DEPARTMENT* são produzidas pela operação de seleção, elas ficam situadas num *buffer*. O algoritmo da operação de junção consumiria as tuplas do *buffer* e essas tuplas que resultaram da operação de junção são *pipelined* ao algoritmo de operação de projeção. A vantagem de *pipelining* é que as poupanças de custo não são obrigadas a escrever o resultado intermediário para o disco e nem realizar a leitura da coluna, para a próxima operação.

2.1.2.2 Estimativas de Custo

Um otimizador de consulta não depende somente das regras de heurísticas para otimizar uma consulta, pois ele pode também calcular e comparar os custos da execução da referida consulta e testar as diferentes estratégias de execução, escolhendo a que possuir menor estimativa de custo.

Um otimizador baseado no custo gera uma faixa de planos de avaliação a partir de uma determinada consulta, usando as regras de equivalência, e escolhe aquele de menor custo.

Para aproximar-se da estratégia de execução com menor estimativa de custo, são requeridas estimativas precisas, de forma que as diferentes estratégias para a aproximação sejam comparadas. Portanto, é necessário limitar o número de estratégias de execução a serem consideradas, pois se gasta muito tempo na produção de

estimativas de custo para as várias possibilidades de estratégias de execução. Conseqüentemente, esta aproximação torna-se mais satisfatória em consultas compiladas, onde a escolha da estratégia de execução é feita em tempo de compilação e fica armazenada no sistema. Para consultas interpretadas, onde todo o processo acontece no *runtime*, a otimização completa pode reduzir a velocidade do tempo de resposta. A otimização mais elaborada é indicada para consultas compiladas, considerando que uma otimização parcial, menos demorada, trabalha melhor em consultas interpretadas.

A otimização baseada em custo, usa técnicas de otimização tradicionais que procuram o espaço de solução para um determinado problema, o que deverá minimizar o custo. As funções de custo usadas na otimização da consulta são estimadas, não são exatas, assim, a otimização pode selecionar uma estratégia de execução de consulta que se aproxima do ótimo.

A otimização baseada em estimativas de custo usa variáveis estatísticas, a fim de estimar o tamanho do resultado e do custo para várias operações e algoritmos. Esta técnica de otimização utiliza alguns componentes ao executar uma consulta. Os componentes são listados logo abaixo:

- Custo de acesso para o armazenamento secundário: Este é o custo de procura, onde ocorre o processo de leitura e escrita dos blocos de dados que residem no armazenamento secundário, principalmente em disco. O custo de procura busca os registros em um arquivo, que depende do tipo das estruturas de acesso naquele arquivo, tal como: ordenação, separação e indexação primária ou secundária. Além do que, os fatores são semelhantes aos blocos de arquivo que são alocados no mesmo cilindro de disco ou são difundidos no disco que afetam o custo de acesso. Além de acessar os arquivos do banco de dados, arquivos temporários podem ser criados se a tabela resultante de uma consulta não couber inteira na memória. Este custo depende do tipo de estrutura de acesso existente para os arquivos envolvidos em determinada consulta.
- Custo de armazenamento: Este é o custo do armazenamento de qualquer arquivo temporário gerado por uma estratégia de execução de uma consulta.
- Custo de computação: Este é o custo para executar operações nos *buffers* da memória, durante a execução da consulta. Tais operações incluem a procura e a ordem dos registros, fundindo estes registros numa junção e executando as

computações em valores de campo. Refere-se ao custo relativo às operações realizadas em memória para seleção, ordenação, combinação e manipulação matemática nos registros das tabelas envolvidas numa consulta.

- Custo do uso da memória: Este é o custo que pertence ao número de *buffers* de memória que são necessários durante a execução da consulta.
- Custo de comunicação: Este é o custo relativo ao envio da consulta e do tempo resultante, do local do banco de dados para o local ou término onde a consulta foi originada. Este custo refere-se à movimentação da consulta e de seus resultados entre a máquina geradora da consulta e o servidor do banco de dados.

Em grandes bancos de dados o principal enfoque é minimizar o acesso válido ao armazenamento secundário (discos). Simples funções de custo ignoram outros fatores e comparam diferentes estratégias de execução de consulta, em termos de número de transferência de bloco entre disco e memória principal. Para banco de dados pequenos, onde a maioria dos dados manipulados pelas consultas cabem na memória, o principal enfoque é diminuir o tempo de computação.

É difícil incluir todos os componentes de custo em uma função, por causa da dificuldade de nomear os pesos satisfatórios aos componentes de custo. Isso por que algumas funções somente consideram um único fator, o acesso ao disco.

Para formular funções de custo é necessário calcular os custos de várias estratégias de execução. É interessante manter pistas de qualquer informação necessária para as funções de custo. Esta informação pode ser armazenada no catálogo do SGBD onde tem acesso ao otimizador de consulta. Para que as funções de custo sejam formuladas, inicialmente, é necessário conhecer o tamanho de cada arquivo. Em um arquivo, onde os registros são todos do mesmo tipo, é indispensável conter: o número de registros (tuplas) (r), a média do tamanho do registro (R) e o número de blocos (b) ou suas estimativas finais. O fator bloqueador (bfr) para o arquivo pode ser requisitado. Há também necessidade de manter pistas do método de acesso primário e os atributos de acesso para cada arquivo. Os registros dos arquivos podem ser ordenados por um atributo, com ou sem índice primário ou agrupado, podendo ainda, separá-lo em um atributo chave. É mantida a informação sobre os índices secundários e atributos indexados. O número de níveis (x) de cada índice com vários níveis (primário, secundário ou agrupado) necessita de funções de custo que calculam o número de

acesso ao bloco que acontece durante a execução da consulta. Em algumas funções de custo ocorrem o bloqueio do número de índice de nível primário (bII).

Outro parâmetro importante é o número de valores distintos (d) de um atributo e sua seletividade (sl), que é a fração de registros que satisfazem uma condição de igualdade no atributo. Isto permite estimação de cardinalidade de seleção (s = sl * r) de um atributo, que é o número comum de registros que satisfarão uma condição de seleção de igualdade naquele atributo. Para um atributo chave, d = r, sl = 1/r e s = 1. Para um atributo que não é chave, supõe-se que d seja distribuído em valores distintos e uniformes entre os registros, então calcula-se sl = (1/d) e assim s = (r/d).

A informação do número de níveis de índice não muda freqüentemente, por este motivo é fácil mantê-la. Porém, outra informação pode mudar freqüentemente, por exemplo, o número de registros r em um arquivo, muda toda vez que o registro é inserido ou apagado.

A seguir serão listados operadores e seus algoritmos, objetivando a obtenção do custo da otimização e conseqüentemente à escolha de um plano de avaliação.

Operação de Seleção:

No processamento de consultas, a varredura de arquivos é o operador de mais baixo nível para ter acesso aos dados. Varreduras de arquivos são algoritmos de procura, que localizam e recuperam os registros que estão de acordo com uma condição de seleção. Em sistemas relacionais, a varredura de arquivos permite ler uma relação inteira, nos casos em que a relação é armazenada num arquivo único e dedicado [Silberschatz et al., 1999].

Serão apresentados dois algoritmos básicos de varredura, para implementar a operação de seleção em uma relação, cujas tuplas são armazenadas juntas num único arquivo.

- **A1** (busca linear), este algoritmo satisfaz a condição de seleção em cada bloco varrido e todos os registros são testados. A varredura termina quando pelo menos metade dos blocos são varridos e o registro seja encontrado. Ainda que possa ser ineficiente, pode ser aplicado em qualquer arquivo, independentemente da ordem do mesmo ou da disponibilidade de índices.

- **A2** (busca binária), este algoritmo usa a busca binária para localizar os registros de uma seleção em um arquivo ordenado, satisfazendo esta seleção. A busca binária é realizada nos blocos do arquivo. As estimativas de custo para a busca binária baseiam-se na hipótese de que todos os blocos de uma relação são armazenados no disco de forma contígua, caso contrário o custo deve ser somado às estimativas.

Para melhor compreensão, é necessário observar o exemplo abaixo, supondo a seguinte informação estatística sobre a relação *conta*:

fconta	=	20	(20 tuplas de conta cabem em um bloco)
V(nome_agência, conta)	=	50	(ou seja, há 50 agências diferentes)
V(saldo, conta)	=	500	(ou seja, há 500 valores diferentes de saldo)
nconta	=	10.000	(ou seja, a relação conta possui 10.000 tuplas)

Considerando a consulta:

σ nome_agência = "*Perryridge*" (conta) ⟶ expressão algébrica transformada em consulta

```
SELECT    nome_agência
FROM      conta
WHERE     nome_agência = "Perryridge"
```

A relação tem 10.000 tuplas, sendo que cada bloco tem 20 tuplas e o número de blocos é bconta = 500, então uma varredura de arquivo simples em *conta* gasta 500 acessos de blocos.

A relação *conta* está ordenada por nome_agência, V(nome_agência, conta) = 50, conseqüentemente (10.000/50 = 200), isto é, 200 tuplas da relação *conta* pertencem à agência *Perryridge*. Essas tuplas caberiam em 200/20 = 10 blocos. Portanto, 20 é o número de tuplas por bloco.

Uma busca binária para encontrar o primeiro registro levaria [log2 (500)] = 9 acessos de bloco. Assim o custo total seria 9 + 10 – 1 = 18 acessos de bloco.

a) Seleções usando índices

As estruturas de índices são caminhos de acesso e os dados podem ser localizados e acessados pelos caminhos de acesso fornecidos pelas estruturas de índices. Quando um índice permite que os registros de um arquivo sejam lidos numa ordem que corresponde à ordem física no arquivo, este índice é chamado de índice primário. Caso contrário, chama-se índice secundário. A varredura de índices consiste em algoritmos de busca que usam esses índices, portanto:

- **A3** (índice primário, igualdade na chave): para recuperar um único registro que satisfaz a condição de igualdade, pode-se usar a comparação de igualdade em atributo-chave com índice primário.
- **A4** (índice primário, igualdade em atributo que não é chave): em registros múltiplos onde usam-se o índice primário e a condição de seleção, especifica-se uma comparação de igualdade em um atributo.
- **A5** (índice secundário, igualdade): onde a especificação for uma igualdade, pode ser usado índice secundário. Essa estratégia consegue recuperar um único registro se o campo de indexação for uma chave. Se o campo de indexação não for uma chave, podem ser recuperados os registros múltiplos. Depois de verificado que o índice é secundário, é concluído o local do pior caso, pois cada registro que cumpre a condição reside em um bloco diferente. Com essa técnica é possível recuperar um único registro se o campo de indexação for uma chave.

b) Seleções que envolvem comparações

Em algoritmos com operações de seleções que envolvem comparações, quando não se tem informações adicionais sobre a comparação a ser feita, conclui-se que pelo menos a metade dos registros satisfará a condição de comparação, portanto tem-se nr/2 tuplas. Uma estimativa mais precisa pode ser feita quando tem-se um valor mais real usado na comparação *(v)*.

O valor mais baixo (min (A,r)) e o valor mais alto (max (A,r)) podem ser usados no catálogo para o atributo. Assim, contando com a suposição de que os valores são igualmente distribuídos, pode-se estimar que o número de registros que cumpra a condição A <= v será 0, se v < min(A,r) e para os casos restantes:

$$N_r \cdot \frac{v - \min(A, r)}{\max(A, r) - \min(A, r)}$$

- **A6** (índice primário, comparação): quando a condição de seleção for uma comparação, pode-se usar um índice primário ordenado. Para recuperação de tuplas é permitido a utilização do índice primário com a seguinte condição:

  A > v ou A >= v

- **Para A >= v**: é feito a busca do valor *v* no índice, para que seja encontrada a primeira tupla no arquivo que tem o valor de A = v. Uma busca no arquivo *a* partindo daquela tupla até o final do arquivo, devolve todas as tuplas que cumpra a condição.
- **Para A > v:** a busca é iniciada na primeira tupla que satisfaz A > v.
- **Para A < v ou A <= v:** não é necessário uma procura de índice. Para A < v, é feita uma varredura simples a partir do início do arquivo até a primeira tupla com o atributo A = v. No caso A <= v, a técnica é parecida, no entanto, a varredura continua até a primeira tupla com atributo A > v. Nos dois casos, não é necessário índice.

$$E_{A4} = HT_i + \frac{b_r}{f_r}$$

Se o valor verdadeiro estiver disponível quando a estimativa de custo for realizada, uma estimativa mais coerente pode ser feita.

$$E_{A4} = HT_i + \left\lceil \frac{c}{f_r} \right\rceil$$

- **A7** (índice secundário, comparação): Para guiar a restauração em condições de comparação, envolvendo <, <=, >= ou >, pode-se usar um índice ordenado secundário. É realizada uma busca a partir do menor valor até *v* nos blocos de índice de mais baixo nível (para < e <=) ou a partir de *v* até o valor máximo (para > e >=). Nessas comparações, onde pelo menos metade dos registros cumprem a condição, é realizado o acesso na metade dos blocos de índice de mais baixo nível, e através do índice, é acessado a metade dos registros do arquivo. E ainda, um outro caminho deve ser percorrido no índice, desde a raiz do bloco para o primeiro bloco folha a ser usado. Além das comparações de desigualdade de índices *clustering,* se encontrar o valor real a ser usado na comparação na hora da estimativa de custo, pode-se obter uma estimativa mais correta.

$$E_{A7} = HT_i + \frac{LB_i + n_r}{2 \quad 2}$$

HT_i = número de níveis no índice *i*
E_{A7} = custo estimado do algoritmo A7
LB_i = nº de blocos de índice de nível mais baixo no índice *i*
n = número de tuplas da relação r

c) Implementação de seleções complexas

Todas as condições de seleção consideradas acima são simples e da forma *A op B*, onde *op* corresponde à operação de igualdade ou de comparação. Neste momento, serão considerados predicados de seleção mais complexos, como:

- ***Conjunção:*** supondo que as condições sejam independentes entre si, a chance que uma tupla cumpra todas as condições é simplesmente o produto de todas as probabilidades.
- ***Disjunção:*** é realizada pela união de todos os registros que cumprem as condições individuais simples *0i.*
- ***Negação:*** o resultado de uma seleção é as tuplas de *r* que não estão na seleção.

A8 (seleção de conjunção usando um índice): Antes de tudo, verifica-se se um caminho de acesso está liberado para um atributo em uma das condições simples. Se existir um caminho, pode-se recuperar os registros que cumprem aquela condição com um dos algoritmos de seleção A2 até A7. Para completar a operação, é feito um teste no *buffer* da memória, para verificar se cada um dos registros recuperados cumprem ou não a condição simples restante. Para que se determine em que ordem as condições simples em uma seleção conjuntiva devem ser testadas, a seletividade é fundamental.

A9 (seleção de conjunção usando índice composto): Pode-se procurar diretamente no índice se a seleção atribui uma condição de igualdade em dois ou mais atributos e se ainda existe um índice composto nesses campos de atributos combinados. O tipo de índice determina qual algoritmo, A3, A4 ou A5 vai ser utilizado.

A10 (seleção de conjunção por meio da interseção de identificadores): O uso de ponteiros de registro ou identificadores de registro é outra alternativa para implementar as operações de seleção conjuntiva. Este tipo de algoritmo necessita de índices com registros de ponteiros nos campos envolvidos nas condições individuais. É realizada uma busca em cada índice com a finalidade de se encontrar ponteiros para tuplas que satisfaçam uma condição individual. O conjunto de ponteiros para tuplas que satisfazem a condição conjuntiva é a interseção de todos os ponteiros.

- **A11** (seleção de disjunção por meio da união de identificadores): É feita uma busca em cada índice para tuplas que cumpram a condição individual, se caminhos de acesso estiverem livres em todas as condições de uma seleção de disjunção. O agrupamento de todos os ponteiros recuperados fornece o conjunto de ponteiros para todas as tuplas que cumprem a condição de disjunção. Entretanto, executa-se uma varredura linear da relação, para encontrar as tuplas que cumprem a condição. Assim, se existir uma condição de disjunção, nesse caso o método de acesso mais eficiente é a varredura linear, com a conjunção de disjunção sendo testada em cada tupla durante a busca.

Operação de Junção:

O objetivo desta etapa é obter o tamanho do resultado de uma junção e calcular a junção de relações, analisando seus respectivos custos. Para efeito de cálculo das estimativas dos tamanhos das junções, será utilizado como exemplo a relação depositante X cliente.

Quando a relação depositante X cliente não possui atributos em comum, utiliza-se a estimativa para produto cartesiano. Já quando a relação depositante X cliente é chave para depositante, o número de tuplas é igual ao número de tuplas em depositante. No caso da relação depositante X cliente não for chave para depositante ou para cliente, utiliza-se a seguinte resolução, supondo que: V (nome_cliente, depositante) = 2500; V (nome_cliente, cliente) = 10000; $N_{depositante} = 5000$; $N_{cliente} = 10000$, então: 5000 * 10000 / 2500 = 20000 e 5000 * 10000 / 10000 = 5000.

Levando em consideração a menor, a estimativa é de 5000 tuplas. Portanto, as considerações gerais são: V = valores diferentes de clientes / depositantes e N = relação de clientes / depositantes.

a) Junção por laço aninhado

A junção por laço aninhado consiste em um algoritmo que examina todos os pares de tuplas em uma relação depositante X cliente, por exemplo. Nesse algoritmo, para cada tupla em depositante o algoritmo varre todas as tuplas em cliente. Esse

algoritmo é muito caro em termos de custo de processamento, pois o número de acesso a blocos da relação é de: $N_{depositante} * b_{cliente} + b_{depositante}$
onde b corresponde ao número de acessos a blocos, é calculado dividindo-se n por f, isto é, tuplas por bloco.
$b_{cliente} = 400$ e $b_{depositante} = 100$
5000 * 400 + 100 = 2000100 (pior caso)
100 + 400 = 500 (melhor caso)

b) Junção por laço aninhado de blocos

Na junção por laço aninhado de blocos, as informações são processadas numa base de blocos, ou seja, nessa junção o algoritmo faz o empilhamento de dois blocos, sendo um dentro do outro em um laço *for*. A geração dos pares de tuplas é feita pelo emparelhamento dos mesmos dentro de cada par de bloco. O custo dessa junção é baseado na seguinte fórmula: cliente X depositante
$b_{depositante} * b_{cliente} + b_{depositante}$ onde,
$b_{cliente} = 400$ $b_{depositante} = 100$
Número de acessos a blocos = 100 * 400 + 100 = 40100 (pior caso)
Número de acessos a blocos = 100 + 400 = 500 (melhor caso)

c) Junção por laço aninhado indexado

Neste tipo de junção, utilizam-se índices fixos ou temporários para a varredura de arquivos. Dada uma tupla tr em r, são procuradas através do índice, as tuplas em s que satisfazem a junção com tr. Por exemplo: dando a relação depositante X cliente, em depositante existe uma tupla de nome_cliente = "Ale". As tuplas que farão a junção em cliente serão as que tiverem nome_cliente = "Ale".

Para calcular o custo de processamento desta junção, considera-se que cliente possui um índice árvore B+ primário no atributo nome_cliente, e contêm 20 entradas em cada nó.

Nesse caso, o cliente possui 10.000 tuplas, a altura da árvore é 4, necessitando de mais um acesso.

$N_{depositante} = 5000$

+ 5000 x 5 = 25100 acessos a blocos.

d) Merge - junção

Consiste em um algoritmo, que utiliza-se de ponteiros que movem-se pelas tuplas de uma relação.

Considera-se uma relação tr x ts onde tr e ts são tuplas que possuem os mesmos valores para atributos de junção. Nessa relação, inicialmente os ponteiros apontam para a primeira tupla da primeira relação. Caso ocorra tuplas em uma relação com mesmos valores de atributos na junção, os mesmos são lidos em Ss.

O número de acessos a blocos é igual a Br + Bs. Uma vez que as tuplas da junção estão em ordem consecutiva, as tuplas e os blocos de tuplas são lidos somente uma vez. HT*i* = número de níveis no índice *i*. Por exemplo: depositante X cliente; Nome_cliente = atrib. junção; e 0 + 100 = 500 acessos a blocos

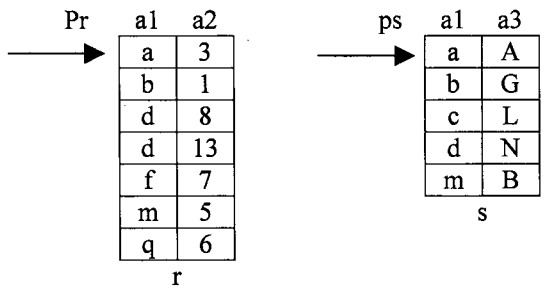

Figura 2.5 – Relações classificadas para merge-junção.
Fonte: SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. *Sistema de banco de dados.* São Paulo: Makron Books, 1999. 3. ed. 778 p.

e) *Hash* - junção

Nesse algoritmo, uma função *hash* h, faz o particionamento das tuplas de ambas as relações.

h – função hash que faz o mapeamento dos valores

Hr0, Hr1, ..., Hrn são partições das tuplas de r.

Hs0, Hs1, ..., Hsn são partições das tuplas de s.

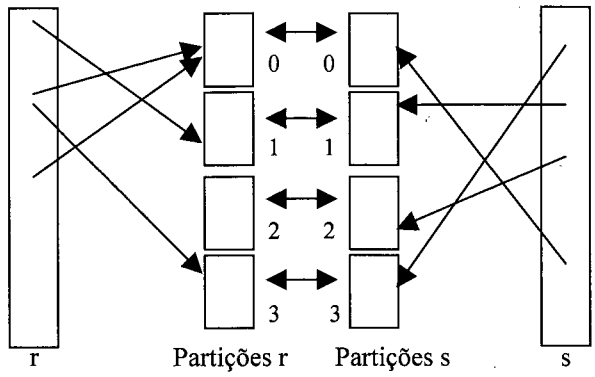

Figura 2.6 – Partições *hash* das relações.
Fonte: SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. *Sistema de banco de dados.* São Paulo: Makron Books, 1999. 3. ed. 778 p.

Por exemplo: d = tupla em depositante; c = tupla em cliente; h (c) = h (d), é necessário testar c e d para observar se os atributos são iguais; h (c)≠ h (d), então c e d tem valores diferentes para nome_cliente.

A fórmula utilizada no cálculo do custo do *hash* – junção é 3(br+bs)+2*max acessos a blocos. Por exemplo: cliente X depositante; tamanho de memória = 20 blocos Depositante é alocado em 5 partições, com tamanho de 20 blocos cada; cliente é alocado em 5 partições, com tamanho de 80 blocos cada; 3 (100 + 400) = 1500 transferências de blocos.

f) *Hash* – junção híbrido

Utilizado normalmente com grandes memórias e quando a relação de construção não cabe na memória.

Por exemplo: cliente X depositante

tamanho de memória = 25 blocos

Depositante é alocado em 5 partições, com tamanho de 20 blocos cada; cliente é alocado em 5 partições, com tamanho de 80 blocos cada; 3 (80 + 320) + 20 + 80 = 1300 transferências de blocos.

g) Junções complexas

As junções complexas podem ser conjuntiva ou disjuntiva. A conjuntiva r X $_{01}$ ^ $_{02}$ ^ ..... ^ $_{0n}$ S, nesse caso as junções são calculadas de maneira individual da seguinte

forma: r X_{01}S, r X_{02}S e assim por diante. A disjuntiva r $X_{01 \wedge 02 \wedge \ldots\ldots \wedge 0n}$S, o cálculo é feito da seguinte forma: (r X_{01}S) U (r X_{02}S) e assim por diante.

2.1.2.3 Classificação

A classificação de dados é essencial e tem um papel muito importante em sistemas de banco de dados. Primeiramente, as consultas SQL podem especificar que o resultado seja apresentado em ordem. Várias das operações relacionais, como as junções, podem ser implementadas eficazmente se as relações de entrada forem primeiramente classificadas. A classificação pode ser usada construindo um índice na chave classificação, e daí, usar aquele índice para ler a relação na ordem de classificação, mas este tipo de processo só ordena a relação logicamente, por meio de um índice, ao invés de ordená-lo fisicamente. Assim, a leitura das tuplas pode encaminhar a um acesso de disco para cada tupla. Por esse motivo, pode-se desejar ordenar as tuplas fisicamente. Quando a classificação de relações não cabe na memória é chamada de classificação externa. O algoritmo de *sort-merge* externo é a técnica mais comum usada.

Primeiramente, neste estágio, ocorre a execução das várias classificações temporárias.

i = 0

repeat

ler M blocos da relação, ou o que falta da relação;
a que for menor, ordenar a parte da relação que está na memória;
escrever os dados que estão em ordem no arquivo temporário R;
i = i + 1;

until o fim da relação

No segundo estágio – é feito o *merge* nos arquivos temporários. Supondo que N seja menor que M, de tal modo, que possa armazenar um *frame* de página para cada temporário, sendo assim, tem-se espaço para manter uma página de resultado.

Dos N arquivos de Ri, deve-se ler um bloco de cada um para uma página de *buffer* na memória;

repeat

escolha a primeira tupla entre todas as páginas do *buffer*;

escrever a tupla no resultado e apagar da página de *buffer*;

if qualquer temporário R*i*, da página de *buffer*, está vazia ***and not*** fim de arquivo (R*i*)

then o próximo bloco de R*i* deve ser lido na página de *buffer*;

until todas as páginas de *buffer* que estiverem vazias

A relação classificada é o resultado da fase de merge. Para se reduzir o número de operações de gravação no disco, o arquivo de resultado fica no *buffer*. *Merge* de n-vias é uma generalização do *merge* de duas vias, usado pelo algoritmo-padrão de *sort-merge* em memória. Podem existir M ou mais temporários gerados na primeira fase, isto é, se existirem relações muito maiores que a memória. Assim, não se consegue alocar um *frame* de página, para cada temporário durante a fase de *merge*. Então, são feitas várias operações de *merge* em múltiplos passos. Tendo memória suficientemente para M – 1 páginas de *buffer* de entrada, cada *merge* terá M – 1 temporários como entrada. Observe o exemplo abaixo:

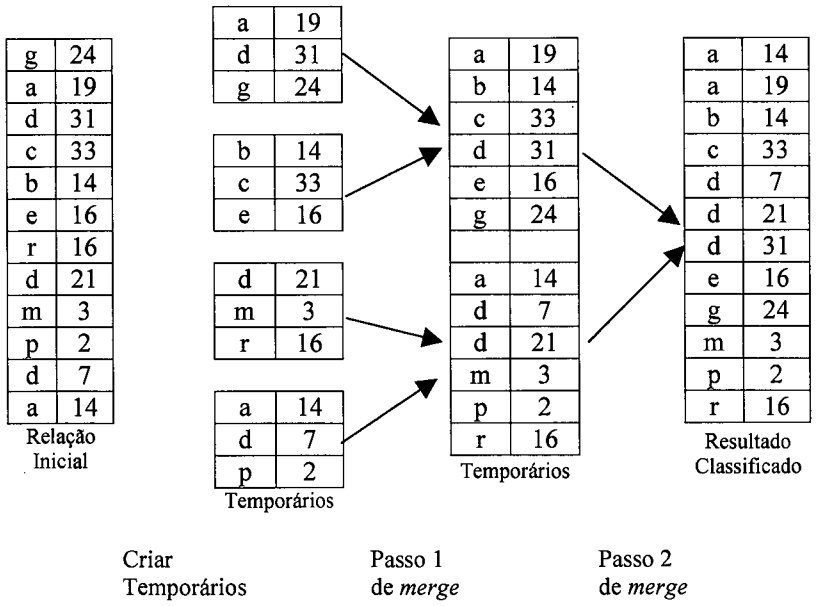

Figura 2.7 – Classificação externa usando *sort-merge*.
Fonte: SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. *Sistema de banco de dados.* São Paulo: Makron Books, 1999. 3. ed. 778 p.

Conforme comentários anteriores, conclui-se que uma consulta pode conter múltiplas relações e junções ordenadas. As regras de transformação algébrica incluem uma regra de comutatividade e a regra de associatividade para a operação de junção. Com estas regras, muitas expressões de junção equivalentes podem ser produzidas. Como resultado, o número de árvores de consulta alternativas crescem muito rapidamente, como também, o número de junções num aumento da consulta.

Geralmente, uma consulta com n relações de junções, terão n-1 operações de junção e conseqüentemente podem ter um grande número de diferentes ordens de junção. A figura 2.8 ilustra duas árvores de consulta com aprofundamento à esquerda (junção).

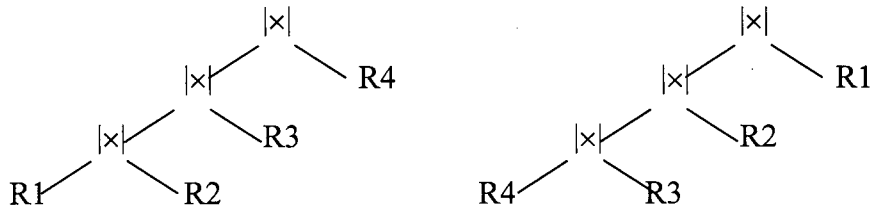

Figura 2.8 – Árvore de consulta com aprofundamento à esquerda.
Fonte: ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

Estimando o custo de todas as possíveis árvores de junção, para uma consulta com um grande número de junções, requererá uma quantia significativa de tempo para otimizar a consulta. Conseqüentemente, é necessário fazer alguma poda das possíveis árvores de consulta. Tipicamente, os otimizadores de consulta limitam a estrutura de uma árvore de consulta com junção, usando árvores com aprofundamento à esquerda ou à direita. Uma árvore com aprofundamento à esquerda é uma árvore binária, onde o filho à direita de cada nó sempre é uma relação base. O otimizador escolheria a árvore com aprofundamento à esquerda, com o menor custo calculado. Árvores com aprofundamento à esquerda, o filho à direita, é considerado relação interna quando executa uma junção por laço aninhado. Uma vantagem de árvores com aprofundamento à esquerda ou à direita, é que elas são favoráveis para *pipelining*. A seção 2.3.2.4 comenta as características de *pipelining* e materialização.

Como por exemplo, considerando a primeira árvore com aprofundamento à esquerda da figura 2.8 e assumindo o algoritmo de junção. Neste caso, uma página de disco de tuplas da relação externa é usada para verificar a relação interna, para emparelhar as tuplas. Como um bloco resultante das tuplas, é produzido pela junção de

R1 e R2, poderia ser usado para investigar R3. No mesmo caso, como uma página resultante de tuplas é produzida pela junção, poderia ser usada para investigar R4.

Outra vantagem de árvores com aprofundamento à esquerda ou aprofundamento à direita, é que elas podem ter uma relação base como introdução a cada junção, que permitirá ao otimizador, utilizar qualquer caminho de acesso àquela relação, podendo então ser útil executar a junção.

Se a materialização é usada ao invés de *pipelining*, os resultados da junção poderiam ser materializados e armazenados como relações temporárias. A idéia chave do ponto de vista do otimizador, com respeito à junção ordenada, é encontrar algo que reduzirá o tamanho dos resultados temporários, desde os resultados temporários de *pipelined* ou materializado, usado por operadores subseqüentes e conseqüentemente afetar o custo de execução desses operadores.

2.1.2.4 Avaliação de Expressões

Na avaliação das expressões, duas maneiras são consideradas: materialização e *pipelining*.

a) Materialização

No processo de avaliação de uma expressão, é avaliada uma operação por vez em sua respectiva ordem. O resultado de cada operação é materializado em uma relação temporária, para ser usado na próxima operação. Uma desvantagem desse método de avaliação é a necessidade de executar relações temporárias, que na maioria das vezes são escritas no disco.

Para analisar uma expressão por materialização, deve-se começar com as operações de nível mais baixo. Os dados de entrada para esse nível estão no banco de dados. Em seguida, o resultado dessa operação é armazenado em relações temporárias, que servirão de entradas para as operações em nível acima da árvore. Para esse nível, pode também existir relações de entradas que estão armazenadas no banco de dados. A

junção dos resultados dessas relações originará uma nova relação temporária. Repetindo esse processo e seguindo a ordem da expressão, eventualmente avalia-se a expressão na raiz da árvore, dando o resultado final da expressão. Esse processo é chamado de avaliação materializada, onde os resultados das operações intermediárias são criados e são usados para avaliar as operações do próximo nível. O custo desse método de avaliação de expressões é calculado pela soma dos custos de todas as operações, assim como o custo de escrita no disco das relações temporárias.

b) *Pipelining*

A avaliação de expressões por *pipelining* tem como característica a redução de arquivos temporários, cujos resultados de uma operação, são passados para a próxima operação no *pipelining*. Esse método elimina o custo de leitura e escrita de relações temporárias, pois evita o armazenamento de uma relação temporária.

Pode-se implementar um *pipelining*, criando uma única e complexa operação que constituem as operações de um *pipelining*. Embora essa prática seja usada na maioria das situações, geralmente é desejável usar o código para as operações individuais na elaboração de um *pipelining*, para que cada operação seja elaborada, como um processo separado ou encaixado no sistema, que recebe como entrada um fluxo de tuplas e gera como resultado outro fluxo de tuplas. Para cada par de operações adjacentes no *pipelining*, um *buffer* é criado para guardar as tuplas que são passadas de uma operação para a próxima. Os *pipelines* podem ser executados de dois modos:

- Dirigido por demanda – o sistema faz repetidas solicitações de tuplas à operação no topo do *pipeline*. Cada vez que uma operação recebe uma solicitação de tuplas, ela calcula a próxima tupla a ser retornada e, então, retorna a tupla. Se a entrada da operação não estiver no *pipeline*, as próximas tuplas a serem retornadas, podem ser calculadas a partir das relações de entrada, sendo mantido o que foi retornado até então.
- Dirigido pelo produtor – as operações não esperam solicitações para produzirem tuplas, mas, em vez disso, geram tuplas "ansiosamente". Cada operação no nível mais baixo do *pipeline* gera tuplas de resultado continuamente e as põe em seu resultado até o *buffer* ficar cheio. Uma operação em qualquer outro nível do

pipeline gera tuplas de resultado, quando ela obtém tuplas de entrada das operações dos níveis mais baixos no *pipeline*, até que seu *buffer* de saída esteja cheio. Uma vez que a operação usa uma tupla de uma entrada no *pipeline*, ela a remove de seu *buffer* de entrada. Nos dois casos, uma vez que o *buffer* esteja cheio, a operação espera até que as tuplas sejam removidas do *buffer* pela operação pai. Nesse momento, a operação gera mais tuplas até que o *buffer* esteja cheio novamente. Esse processo é repetido por uma operação, até que todas as tuplas de resultados tenham sido geradas.

2.1.2.5 Eliminação de duplicidade

Consiste na eliminação de tuplas idênticas. No caso do *sort-merge* externo são efetuadas remoções de duplicatas na criação do registro temporário antes que o mesmo seja escrito em disco. As duplicatas restantes são removidas durante a operação de *merge*.

Já no *hash*-junção o particionamento é feito baseando-se em uma função *hash*. Neste caso, lê-se a partição e constrói-se um índice *hash* na memória principal. Uma tupla somente é inserida nesse índice e essas tuplas são escritas como resultado.

2.1.3 Escolha do Plano de Avaliação

Gerar expressões faz parte do processo de otimização de consultas, cuja cada operação numa expressão pode ser implementada através de diferentes algoritmos. A função do plano de avaliação é definir qual algoritmo deve ser usado para cada operação e coordenar a execução dessas operações. Há vários algoritmos diferentes que podem ser usados para cada operação relacional, gerando um bom número de planos de avaliação alternativos.

Para possibilitar a escolha de um plano de avaliação para determinada consulta, é necessário indicar um algoritmo cujo resultado seja mais barato e assim avaliar cada operação. É indiferente a ordem das operações, podendo ser escolhida qualquer uma delas, desde que executem as operações das camadas mais baixas da árvore antes das operações das camadas mais altas.

Entretanto, escolher um algoritmo barato para cada operação de forma independente, não é necessariamente uma boa saída. Embora um algoritmo *merge*-junção, em um certo nível, possa ser mais cara que uma *hash*-junção, ela consegue prover um resultado classificado, que torna mais barata a avaliação de uma operação posterior (como a eliminação de duplicatas, a interseção ou outra *merge*-junção). De maneira similar, uma junção por laço aninhado com indexação pode proporcionar oportunidades para colocar os resultados em um *pipeline* para a próxima operação e assim, ela seria útil, mesmo se não fosse a forma mais barata de executar uma junção [Silberschatz et al., 1999].

Para obter a melhor escolha do algoritmo global, é conveniente considerar até mesmo os algoritmos que não são melhores para as operações individuais. Portanto, vale considerar expressões alternativas para uma consulta e também considerar algoritmos alternativos para cada operação na expressão. É indicado usar regras como as de equivalência, para definir quais são os possíveis algoritmos para cada operação, inclusive para definir a escolha de se colocar ou não em *pipeline* com outra expressão. Aconselha-se também usar essas regras, para avaliar sistematicamente todos os planos de avaliação das consultas para uma dada expressão.

Em um plano de avaliação, pode-se utilizar as técnicas de estimativas de custo. Embora que, o problema de escolha do melhor plano de avaliação para uma consulta ainda permanece. Existem duas abordagens para o plano de avaliação: a primeira procura todos os planos e escolhe o melhor com base no custo. A segunda usa a heurística para escolher o plano que se aproxima do ótimo. Em prática, os otimizadores de consultas incorporam elementos das duas abordagens, até obter um resultado satisfatório.

CAPÍTULO III

BANCO DE DADOS DISTRIBUÍDO

Em sistemas de banco de dados distribuídos os dados são armazenados em diversos lugares. Eles transmitem informações através de diversos meios de comunicação, como redes de alta velocidade e linhas telefônicas. Este capítulo aponta as principais características de Banco de Dados Distribuído e descreve as etapas do processamento de consultas.

3. Definição

Banco de dados distribuído consiste em uma coleção de nós, cada qual podendo participar na execução de transações que fazem acesso a dados em um ou diversos nós. Em um Sistema de Banco de Dados Distribuído, o banco de dados é armazenado em diversos computadores [Silberschatz et al., 1999].

Os computadores de um Sistema de Banco de Dados Distribuído comunicam-se com outros, por intermédio de vários meios de comunicação. Eles compartilham memória principal ou discos. Os computadores em um Sistema Distribuído podem variar, quanto ao tamanho e funções, desde estações de trabalhos até sistemas de grande porte.

3.1 Sistema Gerenciador de Banco de Dados Distribuídos (SGBDD)

O Sistema Gerenciador de Banco de Dados Distribuído (SGBDD) consiste num conjunto de programas desenvolvidos para manipular um conjunto de dados interelacionados, que são normalmente informações sobre assuntos em comum dispersos em lugares diferentes. O SGBDD relaciona informações entre múltiplas localidades interligadas através de um tipo qualquer de rede de comunicação, onde os usuários possuem mútuo acesso às diversas bases de dados. O SGBDD pode ser visualizado como um grafo, isto é, como um conjunto de nós e bordas, onde cada nó representa uma unidade de processamento central, podendo englobar vários tipos de processadores, como: microcomputadores, estações de trabalho (*work stations*) e sistemas de computadores em geral, formando um banco de dados local que estará, ao mesmo tempo, vinculado através de *links* de comunicação a outros inúmeros nós, com possíveis ramificações. Os *links* de comunicação são representados pelas bordas, formando desta maneira, um relacionamento entre os dados distribuídos, propiciando a cada nó a realização de transações locais, bem como, a participação em transações globais.

Existem várias formas de conectar os nós em um sistema que representam os possíveis meios de transmissão entre os nós e como estão organizados, determinando os caminhos físicos e possíveis trajetos entre os pares de estação conectados a esta rede. Para visualizar mais detalhadamente estas estruturas, será exibido na figura 3.1, o funcionamento da comunicação de dados através de redes de comunicação.

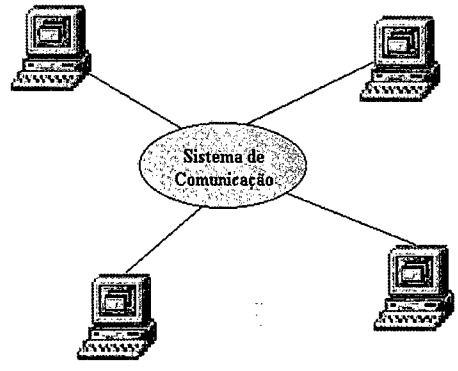

Figura 3.1 – Redes de Computadores

Uma rede de computadores é uma conjunção entre duas tecnologias, a comunicação e o processamento de informações. Estas tecnologias possibilitaram a definição de novas formas de comunicação, com uma maior eficiência dos sistemas computacionais. A rede é formada por um conjunto de módulos processadores, capazes de trocar informações e compartilhar recursos interligados. Estas redes podem ser classificadas como: redes locais (*Local Area Networks – LANs*), redes metropolitanas (*Metropolitan Area Networks – MANs*) e redes geograficamente distribuídas (*Wide Area Networks – WANs*).

3.1.1 Armazenamento, Replicação e Fragmentação dos Dados

Em Banco de Dados Distribuído, com relação ao armazenamento dos dados, um conjunto de tabelas qualquer pode ser manuseado por inúmeros usuários simultaneamente, podendo cada conjunto ser aplicado em diferentes formas de armazenamento.

Para que todos os usuários tenham acesso às informações simultaneamente é necessário o uso de replicações. Visando atender um dos princípios do SGBDD, que é disponibilizar um mesmo arquivo simultaneamente a todos os usuários, o sistema mantém várias cópias idênticas de uma relação original, onde cada cópia é armazenada em diferentes nós (*host*). Em caso de maior necessidade, poderá ser realizada uma replicação total, na qual as cópias são armazenadas em todos os nós do sistema. Outra forma de armazenamento dos dados seria disponibilizar apenas uma cópia de uma relação específica para todos os processamentos, alternado-os entre a original e a cópia. O uso da replicação oferece algumas vantagens, como também algumas restrições.

Caracterizam-se como vantagens: a) disponibilidade: caso ocorra falha em um dos nós onde a relação está armazenada, esta pode ser encontrada em um outro nó, podendo assim continuar o processamento das consultas envolvendo a mesma, apesar da falha; e b) aumento do paralelismo: a replicação dos dados provê maior paralelismo para o sistema, quando se refere a acessos mais direcionados à leitura da relação, permitindo o processamento de consulta simultânea de vários nós a uma mesma relação,

minimizando o movimento dos dados entre os nós.

O aumento de sobrecarga em atualizações é um ponto que torna a replicação desvantajosa, pois existe a necessidade de manter todas as réplicas atualizadas, para que não haja inconsistência de dados, resultando assim numa sobrecarga de processamento. Portanto, a replicação aumenta o desempenho e a disponibilidade nas operações de leitura, gerando, entretanto, uma sobrecarga de processamento devido às transações de atualização.

Outro aspecto importante a ser estudado é com relação à fragmentação dos dados. A fragmentação trata da criação de uma espécie de sub-relação de uma relação original e visa rapidez no acesso aos arquivos utilizados com maior freqüência pelos usuários. Cada relação pode ser particionada em vários fragmentos, que possuirão suas respectivas cópias mantidas pelo sistema em diferentes nós. Tal fragmentação é efetuada através da combinação de operações de restrição e projeção, preservando-se no caso da projeção, a chave primária da relação original. Se uma relação específica for fragmentada, cada fragmento deverá conter informações suficientes para a reconstrução da relação original. Essa reconstrução pode ser realizada através de uma operação união, ou seja, uma espécie de junção dos vários fragmentos. A fragmentação pode ser classificada como horizontal, vertical ou híbrida.

As técnicas de replicação e fragmentação de dados podem ser aplicadas sucessivamente em uma mesma relação, isto é, a relação pode ser replicada, fragmentada, e cada fragmento pode ser replicado e fragmentado ao mesmo tempo.

3.1.2 Transparência

Com o intuito de minimizar a preocupação dos usuários em relação à forma de armazenamento dos dados em um sistema, o mesmo deve prover a transparência de rede, como um de seus princípios básicos. Alguns tipos de transparência devem ser providos.

A principal necessidade de um SGBDD é prover a transparência de localização aos seus usuários, garantindo que os mesmos não precisem saber onde estão

armazenados os dados solicitados ou onde tais dados serão inclusos, devendo manusear o banco de dados como se este estivesse em sua própria máquina. Esta transparência de localidade é claramente vantajosa, visto que facilita a transição dos dados entre as localidades, sem a necessidade de reajustar os programas, garantindo uma independência física dos dados

O processo de fragmentação de arquivos deve prover transparência total quando da utilização do sistema, visto que ao usuário não compete ou interessa saber em que posição se inicia ou encerra o banco de dados em que consta a informação desejada.

É importante considerar também que, independente do arquivo de um banco de dados estar fragmentado ou não, o sistema deve garantir a replicação dos dados aos usuários, bem como, a transparência deste processo, para que não haja incompatibilidade de informações ou duplicidade em uma base de dados, considerando que, por se tratar de um sistema multiusuário, uma falha de atualização em qualquer uma destas réplicas, levaria o acesso às informações irreais, podendo causar grandes perdas financeiras, entre outras.

A replicação dos dados ou dos fragmentos dos dados no sistema é constante para atender a necessidade de cada usuário que o utiliza, lembrando que a transparência de localização de fragmentação e de réplica dos arquivos deve, na visão do usuário, assemelhar-se ao SGBD Centralizado, isto é, como se o mesmo estivesse trabalhando em uma máquina única.

3.1.3 Vantagens e Desvantagens de um SGBDD

A vantagem de utilizar banco de dados distribuído está na possibilidade de uma organização geograficamente dispersa. Este banco de dados é capaz de administrar as informações de forma transparente aos usuários, pois possui autonomia local, interrupção das operações, independência de hardware, entre outras vantagens.

Normalmente um SGBDD é implantado em uma organização, visando modificar a cultura de armazenamento dos dados na mesma e a utilização de suas melhorias operacionais. Uma das melhorias é a capacidade em coordenar inúmeras máquinas

interligadas e sua facilidade em agregar ou excluir novas localidades ao sistema, quando este se apresente em fase de crescimento. Este processo é muito mais prático do que em um SGBDC, pois este requereria em muitos casos a mudança completa do mesmo. Pelo fato do SGBDD apresentar ligações independentes umas das outras, trata-se, certamente, de um sistema mais confiável que o SGBDC, pois caso haja algum problema em uma das localidades, não afetará o sistema como um todo, podendo, no máximo, funcionar em ritmo menor que o habitual.

Referindo-se ao conceito de utilização de réplicas de arquivos comprova-se uma melhoria considerável para o gerenciamento do sistema, considerando que sempre haverá acesso a um banco de dados que possua ao menos uma cópia disponível.

O SGBDD propicia uma facilidade quanto ao acesso local dos dados, reduzindo os custos com a comunicação e o tempo de acesso às informações. Considerando também sua flexibilidade em remanejar os dados, quando há alterações nos parâmetros existentes, podendo mover, reproduzir ou excluir um banco de dados e/ou suas réplicas facilmente, mantendo a integridade das informações.

Outro fator que auxilia para o alcance da eficiência do sistema é o paralelismo na comunicação entre as redes de localidades múltiplas, que fornecem uma comunicação aperfeiçoada de dados entre os locais, agilizando o processo e melhorando o tempo de resposta em certas situações.

O sistema de banco de dados distribuído também possui suas limitações e desvantagens, como por exemplo: a) a velocidade na obtenção das informações não pode ser comparada com a velocidade de acesso a discos locais, pois os fatores adversos de comunicação exercem alguma influência; b) pode acontecer o aumento do *overhead* no processamento; c) o custo de desenvolvimento do *software* pode aumentar; d) aumento de *deadlocks*; e e) possui complexidade adicional para assegurar a própria coordenação entre os nós.

Devido à grande lentidão durante a transmissão de dados, principalmente, em redes de longa distância, o SGBDD tem como objetivo principal minimizar o número e o volume de mensagens processadas. Esses objetivos por sua vez levanta problemas em algumas áreas subsidiárias, nas quais destacam-se: processamento de consulta, propagação de atualização, concorrência, recuperação e gerenciamento de catálogo.

3.2 Processamento de Consultas em Banco de Dados Distribuído

Resumidamente, o processamento de consulta distribuída funciona da seguinte forma: a) a consulta expressa em cálculo relacional deve ser decomposta em uma seqüência de operações relacionais, chamada de consulta algébrica; b) o dado acessado deve ser localizado assim que as operações sobre relações são traduzidas, conduzindo-o num determinado local (fragmentos); e c) a consulta algébrica a partir de fragmentos, deve ser estendida com operações de comunicação e otimizada com referência a uma função de custo para ser minimizada. Esta função de custo minimizada refere-se ao cálculo de recursos como disco de I/O, CPU e redes de comunicação [Özsu e Valduriez, 1999]. A figura 3.2 ilustra a divisão em camadas do processamento de consultas distribuídas. Essas camadas tratam de problemas bem definidos que são: decomposição de consultas, localização dos dados, otimização global e otimização local.

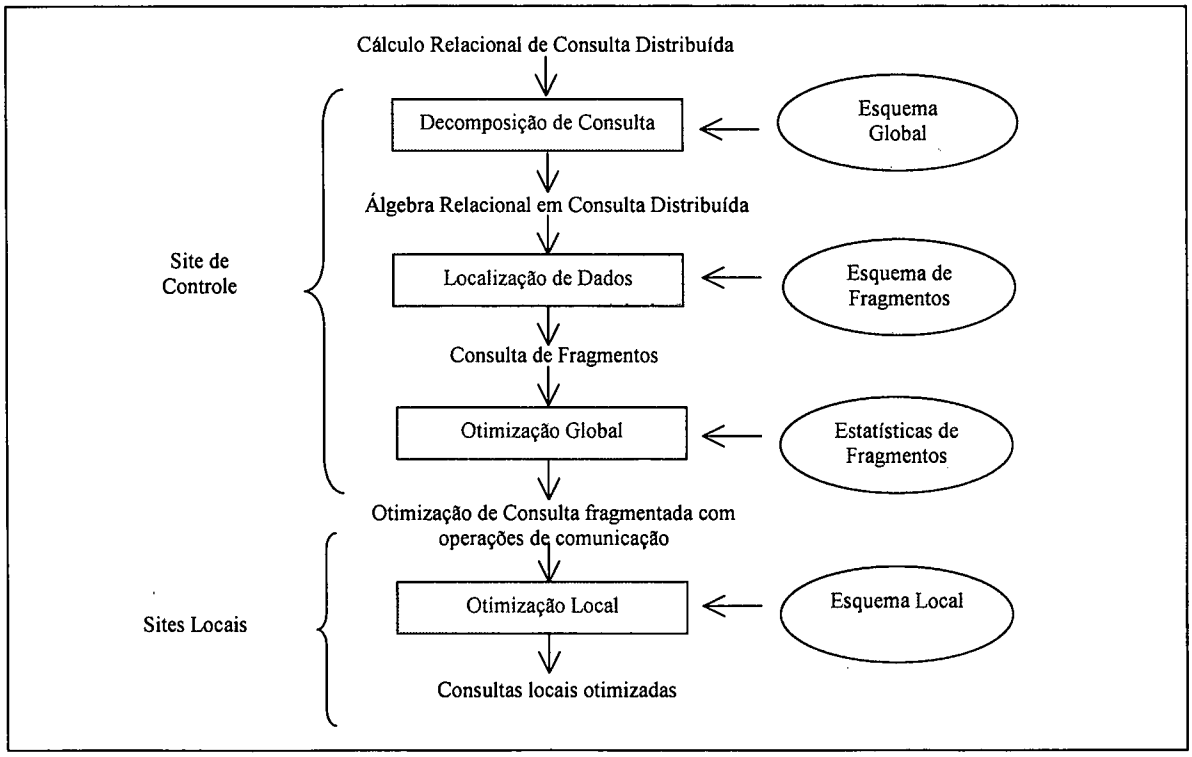

Figura 3.2 – Camadas do Processamento de Consultas
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

O processamento de consulta em banco de dados distribuído (BDD), tem uma complexidade maior que em banco de dados centralizados, devido ao grande número de parâmetros a serem considerados. Pelo fato das consultas distribuídas serem

implementadas através de fragmentos tornam-se mais complexas, porque os parâmetros afetam diretamente o desempenho das consultas. Como dito anteriormente, as relações envolvidas numa consulta distribuída podem ser fragmentadas ou replicadas, influenciando diretamente no custo das despesas de comunicação e, além disso, devido ao maior número de *sites* para acessar, o tempo de resposta pode tornar-se muito alto.

O processamento de consultas em banco de dados distribuído é da maior importância, uma vez que a definição dos fragmentos é baseada no crescimento da localidade referente e na execução paralela para consultas mais importantes [Özsu e Valduriez, 1999]. Em consultas distribuídas, a álgebra relacional não é suficiente para expressar estratégias de execução, por isso, é necessário fazer uma complementação com operações de troca de dados. É necessário selecionar os melhores locais para processar os dados, dificultando assim o processamento.

As próximas seções descrevem as etapas de processamento de consultas em Banco de Dados Distribuídos.

3.2.1 Decomposição da Consulta

A decomposição da consulta é a mesma para sistemas centralizados e distribuídos e sua função é transformar o cálculo relacional numa consulta de álgebra relacional. [Özsu e Valduriez, 1999]. Nesta camada, o resultado da consulta algébrica é "bom" e as piores execuções serão evitadas. A consulta fragmentada final geralmente está longe de ser uma ótima solução de execução, porque as informações relativas a fragmentos não são exploradas neste momento. A decomposição de consulta pode ser vista em quatro passos: a) o cálculo da consulta é reescrito de uma forma normalizada e satisfatória para a manipulação subseqüente, a normalização dá prioridade ao operador lógico; b) a consulta normalizada é analisada semanticamente, de forma que as consultas incorretas, sejam rejeitadas assim que possível; c) a consulta correta (ainda em cálculo relacional) é simplificada; e d) o cálculo de consulta é reestruturado como uma consulta algébrica [Özsu e Valduriez, 1999]. Os passos da decomposição de consulta são ilustrados na figura 3.3.

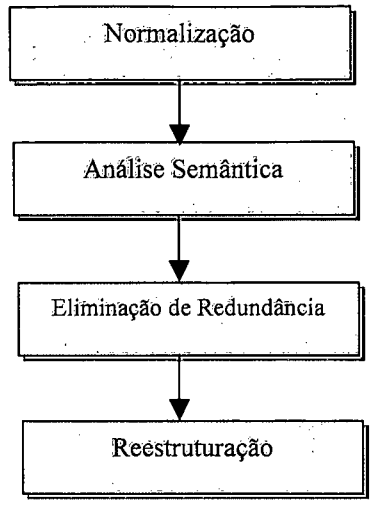

Figura 3.3 – Passos da Decomposição da Consulta

a) Normalização

Na fase de normalização, a consulta de entrada pode ser arbitrariamente complexa, dependendo das facillidades providas pela linguagem. O principal objetivo da normalização é transformar a consulta numa forma que facilite o processo que será realizado num próximo passo.

Na linguagem relacional, como a SQL, a transformação mais importante é a qualificação da consulta (cláusula *where*), podendo ser arbitrariamente um predicado complexo quantificador-livre precedido por todo quantificador necessário ($\forall$ ou $\exists$).

Há duas possíveis formas para o predicado, um dando precedência para o *and* ($\wedge$) e o outro para o *OR* ($\vee$). A forma conjuntiva normal é uma conjunção (predicado $\wedge$) de disjunções (predicado$\vee$), onde p_{ij} é um predicado simples. Observe o exemplo: $(p_{11} \vee p_{12} \vee ... \vee p_{1n}) \wedge ... \wedge (p_{m1} \vee p_{m2} \vee ... \vee p_{mn})$. A qualificação da forma disjuntiva normal seria: $(p_{11} \wedge p_{12} \wedge ... \wedge p_{1n}) \vee ... \vee (p_{m1} \wedge p_{m2} \wedge ... \wedge p_{mn})$.

Para realizar a transformação do predicado quantificador-livre é necessário utilizar as regras de equivalência para operações lógicas ($\wedge$, $\vee$, e $\neg$), sendo elas:

1. $p_1 \wedge p_2 \Leftrightarrow p_2 \wedge p_1$
2. $p_1 \vee p_2 \Leftrightarrow p_2 \vee p_1$
3. $p_1 \wedge (p_2 \wedge p_3) \Leftrightarrow (p_1 \wedge p_2) \wedge p_3$
4. $p_1 \vee (p_2 \vee p_3) \Leftrightarrow (p_1 \vee p_2) \vee p_3$

5. $p_1 \wedge (p_2 \vee p_3) \Leftrightarrow (p_1 \wedge p_2) \vee (p_1 \wedge p_3)$
6. $p_1 \vee (p_2 \wedge p_3) \Leftrightarrow (p_1 \vee p_2) \wedge (p_1 \vee p_3)$
7. $\neg(p_1 \wedge p_2) \Leftrightarrow \neg p_1 \vee \neg p_2$
8. $\neg(p_1 \vee p_2) \Leftrightarrow \neg p_1 \wedge \neg p_2$
9. $\neg(\neg p) \Leftrightarrow p$

Na forma disjuntiva normal, a consulta pode ser processada como um conjuntivo de subconsultas independentes unidas com operações de união, esses operadores correspondem às disjunções. Porém, esta forma pode conduzir à junção replicada e a predicados de seleção, conforme exemplo seguinte, justamente porque os predicados são freqüentemente unidos com outros predicados através do *and*. Utilizando-se a regra 5 acima mencionada, assume-se que p_1 é uma junção ou predicado de seleção, portanto o resultado seria a replicação de p_1.

A forma conjuntiva normal é mais prática, pois as qualificações de consultas incluem com maior freqüência os predicados *and* do que *or*. Porém, conduzir a replicação do predicado para consultas envolvidas em muitas disjunções e algumas conjunções, é um caso raro. Por exemplo, considere a seguinte consulta no banco de dados de engenharia. "Ache os nomes dos empregados que têm trabalhado no projeto P_1 durante 12 ou 24 meses". A consulta em SQL seria:

SELECT ENAME
FROM EMP, ASG
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = "P1"
AND DUR = 12 **OR** DUR = 24

A qualificação na forma conjuntiva normal seria:

EMP.ENO = ASG.ENO $\wedge$ ASG.PNO = "P1" $\wedge$ (DUR = 12 $\vee$ DUR = 24)

A qualificação na forma disjuntiva normal seria:

(EMP.ENO = ASG.ENO $\wedge$ ASG.PNO = "P1" $\wedge$ DUR = 12) $\vee$
(EMP.ENO = ASG.ENO $\wedge$ ASG.PNO = "P1" $\wedge$ DUR = 24

Na qualificação da forma disjuntiva normal, trabalhando as duas conjunções independentemente, há a possibilidade de existir um trabalho redundante, caso não sejam eliminados as subexpressões comuns.

b) Análise Semântica

A análise semântica habilita rejeição de consultas normalizadas onde o processo é impossível ou desnecessário. A razão principal para ocorrer rejeição é quando uma consulta é do tipo incorreta ou semanticamente incorreta. Quando descoberto um desses casos, a consulta é devolvida ao usuário com uma explicação, caso contrário, o processamento da consulta continua.

Uma consulta é do tipo incorreta, se qualquer de seu atributo ou nome da relação não forem definidos no *Schema Global*, ou ainda se estão sendo aplicadas operações para atributos do tipo errado. Uma consulta é semanticamente incorreta se os componentes não contribuem de qualquer forma à geração do resultado.

Existem técnicas que descobrem consultas incorretas. As técnicas utilizadas para detectar consultas do tipo incorretas são semelhantes a um programa que confere a digitação de determinada linguagem. Considere a seguinte consulta: "Encontre os nomes dos programadores responsáveis pelos projetos CAD/CAM por mais de três anos, e o nome do seu gerente". A consulta expressa em SQL:

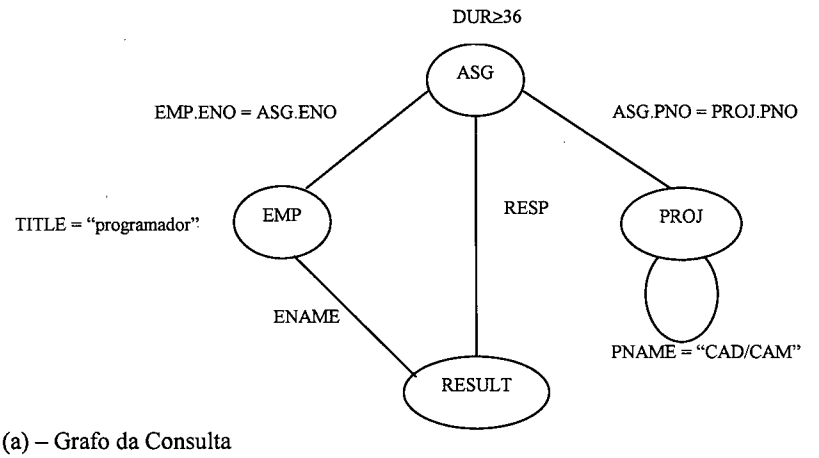

Figura 3.4 (a) – Grafo da Consulta
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

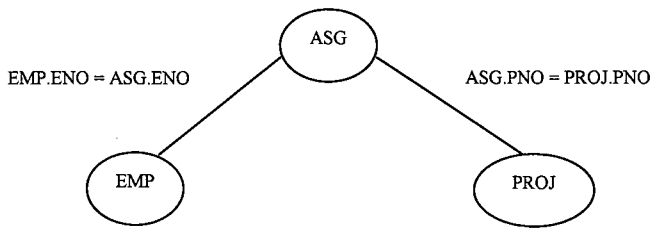

Figura 3.4 (b) – Grafo da Junção
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

SELECT ENAME, RESP
FROM EMP, ASG, PROJ
WHERE EMP.ENO = ASG.ENO
AND ASG.PNO = PROJ.PNO
AND PNAME = "CAD/CAM"
AND DUR ≥ 36
AND TITLE = "programador"

O grafo da consulta para a consulta descrita acima é ilustrado na figura 3.4 (a). A figura 3.4 (b) mostra o grafo resultante da junção do grafo da figura 3.4 (a).

c) Eliminação de Redundância

Uma consulta na visão do usuário pode ser enriquecida com vários predicados, para alcançar uma correspondência de visão-relação, assegurando a integridade semântica e segurança. A consulta enriquecida pode conter predicados redundantes. Esta redundância pode ser eliminada, simplificando a qualificação com regras de idempotência, sendo elas:

1. $p \wedge p \Leftrightarrow p$
2. $p \vee p \Leftrightarrow p$
3. $p \wedge$ verdadeiro $\Leftrightarrow p$
4. $p \vee$ falso $\Leftrightarrow p$
5. $p \wedge$ falso $\Leftrightarrow$ falso
6. $p \vee$ verdadeiro $\Leftrightarrow$ verdadeiro
7. $p \wedge \neg p \Leftrightarrow$ falso
8. $p \vee \neg p \Leftrightarrow$ verdadeiro
9. $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$
10. $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$

Considere o exemplo a seguir. A consulta na linguagem SQL

SELECT TITLE
FROM EMP
WHERE (NOT (TITLE = "programmer")
AND (TITLE = "programmer"

OR TITLE = "Elect. Eng.")

AND NOT (TITLE = "Elect. Eng."))

OR ENAME = "J. Doe"

pode ser simplificada usando as regras acima mencionadas, tornando-se:

SELECT TITLE

FROM EMP

WHERE ENAME = "J. Doe"

Na simplificação da consulta, p_1 corresponde a <TITLE = "Programmer">, p_2 corresponde a <TITLE = "Elect. Eng."> e p_3 corresponde a <ENAME = "J. Doe">. A qualificação da consulta seria: $(\neg p_1 \wedge (p_1 \vee p_2) \wedge \neg p_2) \vee p_3$.

A forma disjuntiva normal para a qualificação é obtida aplicando a regra 5, definida na normalização $(\neg p_1 \wedge ((p_1 \wedge \neg p_2) \vee (p_2 \wedge \neg p_2))) \vee p_3$ e a regra 3, também definida na normalização $(\neg p_1 \wedge p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge p_2 \wedge \neg p_2) \vee p_3$.

Aplicando a regra 7 definida na eliminação da redundância, obtêm-se: $(falso \wedge \neg p_2) \vee (\neg p_1 \wedge falso) \vee p_3$. Aplicando a mesma regra, obtêm-se: $falso \vee falso \vee p_3$ que é equivalente a p_3 através da regra 4.

d) Reestruturação

A última etapa da decomposição da consulta reescreve a consulta em álgebra relacional. Essa etapa é dividida em dois passos: a) transformação direta da consulta expressa em cálculo relacional para a álgebra relacional; e b) reestruturação da consulta já expressa em álgebra relacional para melhorar o desempenho. A transformação da tupla de uma consulta em cálculo relacional, em uma árvore de operador, pode ser conseguida facilmente, como segue:

a) uma folha diferente é criada para cada variável de tupla diferente (relação). Em SQL, as folhas estão disponíveis na cláusula *FROM.*

b) o nodo raiz é criado como uma operação *PROJECT* envolvendo os atributos resultantes. Estes são encontrados na cláusula *SELECT* da SQL.

c) a qualificação (cláusula *WHERE* da SQL) é traduzida na sucessão apropriada de operações relacionais (*select, join, union* e etc) seguindo das folhas para a raiz.

A sucessão pode ser dada diretamente pela ordem de aparecimento dos

predicados e operadores. Considere como exemplo, a consulta a seguir: "Encontre os nomes dos empregados diferentes de J. Doe, que trabalhou no projeto CAD/CAM durante um ou dois anos". A expressão em SQL é:

SELECT ENAME
FROM PROJ, ASG, EMP
WHERE ASG.ENO = EMP.ENO
AND ASG.PNO = PROJ.PNO
AND ENAME ≠ ''J.Doe''
AND PROJ.PNAME = ''CAD/CAM''
AND (DUR = 12 **OR** DUR = 24)

Observe a árvore da figura 3.5, os predicados foram transformados em ordem de aparecimento como operação de junção e seleção. Aplicando as regras de transformação, muitas árvores diferentes equivalentes podem ser encontradas.

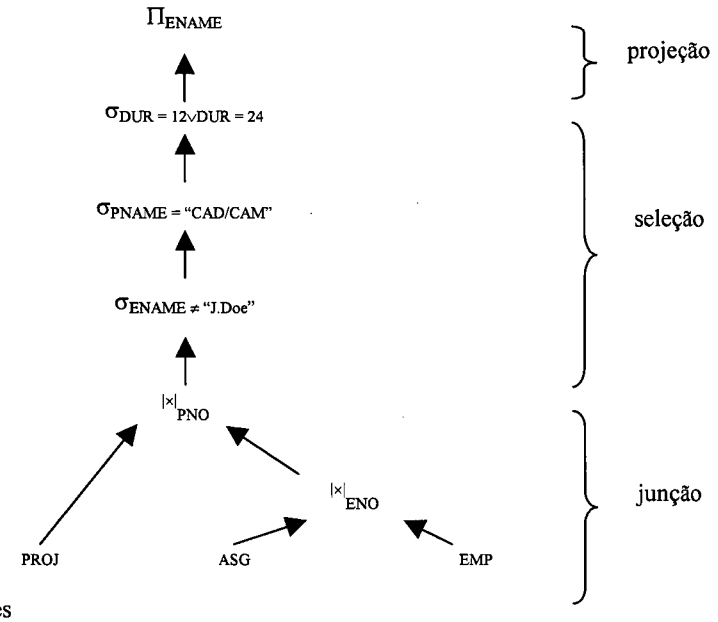

Figura 3.5 – Árvore de operadores
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

3.2.2 Localização de Dados Distribuídos

Na camada anterior são apresentadas técnicas gerais para a decomposição e reestruturação de consultas expressas em cálculo relacional. Estas técnicas globais são

aplicadas tanto para SGBD centralizados como distribuídos, não levando em conta a distribuição dos dados. A distribuição dos dados é tratada na camada de localização.

A camada de localização traduz a consulta algébrica em relações globais em uma consulta algébrica em forma de fragmentos físicos. A localização faz uso de informações armazenadas no esquema de fragmento [Özsu e Valduriez, 1999]. É na camada de localização, que são determinados quais fragmentos estão envolvidos na consulta e logo em seguida transforma a consulta distribuída em uma consulta fragmentada.

Uma relação global pode ser reconstruída aplicando regras de reconstrução ou fragmentação inversa, derivando um programa da álgebra relacional cujos operadores são os fragmentos, chamando então programa de localização [Özsu e Valduriez, 1999]. Um modo simples para localizar uma consulta distribuída é gerar uma consulta, onde cada relação global é substituída por seu programa de localização. A partir daí a consulta passa a ser chamada de consulta genérica. Geralmente, esta aproximação é ineficiente, porque ainda podem ser feitas reestruturações importantes e simplificações na consulta genérica. Cada tipo de fragmentação possui técnicas de redução que geram consultas otimizadas. Existem alguns tipos de redução de fragmentação e cada um deles tem uma função específica.

a) Redução para Fragmentação Horizontal Primária

A função da fragmentação horizontal primária é distribuir uma relação baseada em predicados de seleção. A redução das consultas com relações horizontalmente fragmentadas consiste principalmente em determinar, depois de reestruturar as subárvores, aquelas que produzirão relações vazias. A fragmentação horizontal pode ser explorada, para simplificar operações de seleção e união.

O programa de localização utilizado para uma relação horizontalmente fragmentada é o de união de fragmentos. A junção de relações horizontalmente fragmentadas pode ser simplificadas quando as relações de junção são fragmentadas de acordo com o atributo da junção. A simplificação consiste em distribuir junção sob uniões, eliminando assim, as junções inúteis. Nem sempre a consulta reduzida é melhor que a consulta genérica. O pior caso acontece, quando cada fragmento de uma relação

deve ser unido com cada fragmento da outra relação. Isto é equivalente ao produto cartesiano dos dois jogos de fragmentos, cada jogo corresponde a uma relação. A consulta reduzida é melhor quando o número de junção é pequeno.

b) Redução para Fragmentação Vertical

A finalidade da redução da fragmentação vertical é distribuir uma relação baseada em atributos de projeção. A junção é o operador usado para reconstruir a fragmentação vertical e o programa de localização para uma relação verticalmente fragmentada, consiste na junção de fragmentos no atributo comum. Semelhante à fragmentação horizontal, consultas em fragmentos verticais podem ser reduzidas, determinando as relações intermediárias inúteis e removendo as subárvores produzidas.

c) Redução para Fragmentação Derivada

Como visto anteriormente, a operação de junção é a operação mais importante. Visto que sua utilização é mais freqüente e cara, essa operação pode ser otimizada usando a fragmentação horizontal primária, quando as relações unidas são fragmentadas de acordo com os atributos da junção. Neste caso a junção de duas relações é implementada como uma união de junção parcial. Porém, este método impede uma das relações de ser fragmentada em um atributo diferente usado para seleção. A fragmentação horizontal derivada é uma outra maneira de distribuir duas relações, de forma que o processo em comum da seleção e junção é melhorado.

Se uma relação R está sujeita a fragmentação horizontal derivada, devido à relação S, os fragmentos de R e S que têm os mesmos valores de atributo na junção, são localizados no mesmo local. Além do que, S pode ser fragmentado de acordo com um predicado de seleção. Desde que as tuplas de R são colocadas de acordo com as tuplas de S, a fragmentação derivada somente deveria ser utilizada para relações um-para-muitos, onde uma tupla de S pode emparelhar com n tuplas de R, porém uma tupla de R com exatamente uma tupla de S [Özsu e Valduriez, 1999].

A fragmentação derivada poderia ser usada por muitas relações, desde que as tuplas de S sejam reproduzidas.

d) Redução para Fragmentação Híbrida

A fragmentação híbrida é obtida a partir da combinação das funções das fragmentações horizontais, verticais e derivadas, sendo que o seu objetivo é manter, eficazmente, consultas que envolvem projeção, seleção, e junção. A otimização de uma operação ou uma combinação de operações, sempre termina às custas de outras operações. Por exemplo, fragmentação híbrida baseada em seleção - projeção somente fará seleção, ou somente projeção, tornando-a menos eficiente que com a fragmentação horizontal ou a fragmentação vertical. O programa de localização, para a relação com fragmentação híbrida, usa uniões e junções de fragmentos. As consultas fragmentadas híbridas podem ser reduzidas, combinando respectivamente as regras horizontal primária, vertical e fragmentação horizontal derivada [Özsu e Valduriez, 1999].

3.2.3 Otimização de Consultas em Banco de Dados Distribuído

O módulo do software que executa a otimização de consulta é composto por três componentes: espaço de procura, modelo de custo e estratégia de procura [Özsu e Valduriez, 1999]. O espaço de procura de uma consulta submetida consiste de todas as alternativas de planos de execução da consulta. Os planos de execução da consulta são resumidos tipicamente por meio de operadores em árvores, que definem a ordem na qual as operações são executadas. Para uma determinada consulta, o espaço de procura pode ser definido assim como o jogo de operadores equivalentes em árvores, que podem ser produzidos usando regras de transformação.

Para uma consulta complexa, envolvendo muitas relações e muitos operadores, o número de operadores equivalentes em árvores pode ser muito alto. O otimizador de consulta restringe o tamanho do espaço de procura. A primeira restrição é usar heurística. A heurística mais comum usa a execução da seleção e projeção quando tem acesso às relações básicas. Outra heurística comum é evitar produtos cartesianos que não são requeridos pela consulta. Existe uma restrição importante que diz respeito à forma da junção de árvore.

As estratégias de procura mais usadas pelo otimizador de consulta são programação dinâmica e determinística. Estratégias determinísticas procedem construindo planos, a partir de relações básicas, unindo uma relação a cada passo, obtendo planos completos. A programação dinâmica é exaustiva e assegura que o "melhor" de todos os planos seja achado. Estratégias Randômicas procura a ótima solução ao redor de alguns pontos particulares. Eles não garantem que a melhor solução seja obtida, mas evita o alto custo de otimização, em termos de memória e consumo de tempo.

O modelo de custo de um otimizador inclui funções que indicam o custo de operadores, estatísticas de base de dados e fórmulas para avaliar os tamanhos dos resultados intermediários.

Uma boa medida de consumo de recurso é o custo total que acontecerá no processamento de uma consulta. O custo total equivale à soma de todas as vezes que acontecerá o processamento das operações em vários locais e comunicações *intersites*.

Uma outra medida a ser considerada é o tempo de resposta, medida esta, que é igual ao tempo decorrido na execução da consulta, isto é, o tempo para a conclusão da consulta. Ainda com relação a essa medida, considera-se que as operações podem ser executadas em paralelo e em locais diferentes, sendo assim, o tempo de resposta da consulta pode ser menor que seu custo total.

Em sistemas de Banco de Dados Distribuídos, com relação à medida do custo total, deve-se considerar: a) CPU, incorrida quando executa as operações dos dados em memória principal; b) I/O, corresponde ao tempo necessário para que o disco introduza as operações de *input* e *output*; e c) custo de comunicação, corresponde ao tempo necessário para trocar os dados entre os locais que participam da execução da consulta.

O custo de comunicação é o fator considerado como mais importante em banco de dados distribuído. Na otimização das consultas distribuídas, o custo de comunicação domina o custo de processamento local (I/O e custo de CPU). Então, a otimização de consulta distribuída visa reduzir o problema, minimizando o custo de comunicação geralmente gasto no processamento local. A vantagem é que a otimização local pode ser feita usando os métodos conhecidos em sistemas centralizados. Recentes pesquisas consideram significantes estes três componentes de custo, desde que todos eles contribuem para o custo total na avaliação de uma consulta.

A topologia de uma grande rede influencia a relação entre estes componentes de custo. Em uma rede como a Internet, geralmente o tempo de comunicação é o fator dominante, em redes de área locais, porém, há um equilíbrio entre os componentes.

O fator principal que afeta o desempenho de uma estratégia de execução é o tamanho das relações intermediárias que são produzidas durante a execução. Quando uma operação subseqüente é encontrada em um local diferente, a relação intermediária deve ser transmitida na rede. Então, é de interesse principal calcular o tamanho das operações da álgebra relacional intermediária resultante, para minimizar o tamanho das transferências de dados.

Outro ponto a ser destacado no processamento de consultas distribuídas é com relação à complexidade das operações da álgebra relacional, complexidade esta, que pode afetar diretamente o tempo de execução dos princípios úteis no processamento da consulta. Os princípios úteis podem ajudar na escolha de estratégias de execução final. O modo mais simples para definir complexidade está diretamente relacionado a termos de cardinalidade de relação, como fragmentação e estruturas de armazenamento. A complexidade é relativa à cardinalidade das relações. As operações mais seletivas que reduzem a cardinalidade, por exemplo a operação de seleção, deveriam ser executadas primeiro e logo em seguida as operações deveriam ser ordenadas para aumentar a complexidade, de forma que evitassem os produtos cartesianos.

A junção ordenada é um aspecto importante na otimização de consulta centralizada e num contexto distribuído é mais importante que junção entre fragmentos, pois pode aumentar o tempo de comunicação. Existem algumas aproximações básicas para junção ordenada em fragmentos de consultas. Tenta-se otimizar a junção ordenada diretamente, considerando que a junção seja substituída por combinações de semijunções para minimizar os custos de comunicação. Alguns algoritmos otimizam a junção ordenada, diretamente, sem usar semijunções. O Algoritmo INGRES distribuído e R* representam algoritmos que usam junção no lugar de semijunções.

Considerando que uma consulta seja localizada e fragmentada, neste momento não há necessidade de distinguir fragmentos da mesma relação e fragmentos de relações diferentes. Primeiramente, deve-se considerar o problema mais simples de transferência de operador em uma única junção [Özsu e Valduriez, 1999].

Supondo uma consulta R × S, onde R e S são relações armazenadas em locais

diferentes. A escolha óbvia da relação para transferência é enviar a relação menor para o local da relação maior, dando lugar às duas possibilidades. Para fazer esta escolha é preciso avaliar o tamanho de R e de S. Como no caso de uma única junção, o objetivo do algoritmo de junção ordenada é transmitir o menor operador. As operações da junção podem reduzir ou podem aumentar o tamanho dos resultados intermediários. Uma alternativa para solucionar este problema é calcular os custos de comunicação de todas as estratégias alternativas e escolher o melhor, isto é, o de menor custo. Porém, o número de estratégias cresce rapidamente com o número de relações. Esta aproximação, usada pelo Sistema R*, torna a otimização cara, embora seja amortizada rapidamente se a consulta for executada freqüentemente.

A operação de semijunção pode ser usada para diminuir o tempo total de junção das consultas. A semijunção age como um redutor de tamanho de uma relação. A junção de duas relações, R e S, com atributo A, armazenada nos sites 1 e 2, respectivamente, pode ser computada substituindo uma ou ambas relações de operador, por uma semijunção com a outra relação e podem ser usadas as seguintes regras:

$$R \times_A S \Leftrightarrow (R \times_A S) \times_A S$$
$$\Leftrightarrow R \times_A (S \times_A R)$$
$$\Leftrightarrow (R \times_A S) \times_A (S \times_A R)$$

A escolha entre uma das três estratégias de semijunção requer calcular os custos respectivos. O uso da semijunção é benéfico se o custo para produzir e enviar a consulta para o outro local for menor que o custo de enviar a relação do operador inteiro e de fazer a atual junção. A aproximação da semijunção melhora, se agir como um redutor suficiente, quer dizer, se algumas tuplas de R participarem da junção. A aproximação da junção melhora, se quase todas as tuplas de R participarem da junção, porque a aproximação da semijunção requer uma transferência adicional de uma projeção no atributo da junção.

As semijunções podem ser úteis, reduzindo o tamanho dos operadores das relações envolvidas, em múltiplas junções de consultas. Porém, a otimização de consulta fica mais complexa nestes casos. Comparada à junção, a semijunção induz mais operações, embora possui operadores menores. A junção de duas relações, EMP X ASG, acaba enviando uma relação ASG, para o local do outro e EMP completa a junção localmente. Quando uma semijunção é usada, a transferência da relação ASG é evitada. Se na junção a duração do atributo é menor que a duração de uma tupla inteira e a

semijunção tem seletividade boa, então a aproximação de semijunção pode resultar em poupanças significantes em tempo de comunicação. Usando semijunções, existe a possibilidade de aumentar o tempo do processo local, desde que, uma das duas relações unidas tenha acesso duas vezes.

A estimação, também pode estar baseada em informações estatísticas sobre as relações básicas e fórmulas de cardinalidade dos resultados das operações relacionais [Özsu e Valduriez, 1999].

Para uma relação R definida sobre os atributos $A = (A_1, A_2,..., A_n)$ e fragmentos $R_1, R_2,..., R_r$, os dados estatísticos são tipicamente o seguinte:

a) para cada atributo A_i, sua duração (em número de bytes), denota através da duração (A_i), e para cada atributo A_i de fragmento R_j, o número de valores distintos de A_i, com cardinalidade da projeção dos fragmento R_j em A_i, denota através da $card(\pi A_i(R_j))$;

b) para o domínio de cada atributo A_i que é definido em um jogo de valores que podem ser ordenados (por exemplo, inteiros ou reais), o mínimo e máximo dos possíveis valores, denota através de $min(A_i)$ e $max(A_i)$.

c) para o domínio de cada atributo A_i, a cardinalidade do domínio A_i, denota através de ($dom[A_i]$).

d) número de tuplas em cada fragmento R_j, denota através da $card(R_j)$.

As estatísticas do banco de dados são úteis, pois avaliam a cardinalidade de um resultado intermediário de consultas. É suposto, que a distribuição dos valores do atributo em uma relação é uniforme e todos os atributos são independentes, significando que o valor de um atributo não afeta o valor de qualquer outro. Estas duas suposições estão erradas em prática, mas considera-se o problema tratável. Logo abaixo serão fornecidas as fórmulas para calcular a cardinalidade dos resultados das operações da álgebra relacional (seleção, projeção, produto cartesiano, junção, semijunção, união e diferença). Os operadores das relações são denotados por R e S [Özsu e Valduriez, 1999].

a) seleção: a cardinalidade de seleção é $card(\sigma_F(R)) = SF_S(F) * card(R)$ onde $SF_S(F)$.

b) projeção: pode ser com ou sem eliminação de duplicadas. A projeção com eliminação de duplicada é difícil ser avaliada, justamente porque as correlações entre

os atributos projetados são normalmente desconhecidas. Se a projeção da relação R está baseada em um único atributo A, a cardinalidade é simplesmente o número de tuplas quando a projeção é executada. Se um dos atributos projetados é uma chave de R, então card(πA(R)) = card(R).

c) **produto cartesiano:** a cardinalidade do produto cartesiano de R e S é card(R × S) = card(R) * card(S).

d) **junção:** não há nenhum modo geral, para calcular a cardinalidade de uma junção, sem informação adicional. O salto superior da cardinalidade da junção é a cardinalidade do produto cartesiano.

e) **semijunção:** a seletividade da semijunção de R por S resulta na fração (porcentagem) das tuplas de R que fazem junção com as tuplas de S. Uma aproximação para o fator de seletividade de semijunção é determinada como SF_{SJ} (R $\times_A$ S) = $\frac{\text{card } (\pi I_A (S))}{\text{card (Dom [A])}}$

Esta fórmula só depende do atributo A para S. Assim é chamado freqüentemente de fator de seletividade de atributo A para S, denotado SF_{SJ}(S.A) e é o fator de seletividade de S.A em qualquer outro atributo. Então, a cardinalidade de semijunção é determinado por card(R $\times_A$ S) = SF_{SJ}(S.A) * card(R).

f) **união:** é difícil calcular a cardinalidade da união de R e S porque as duplicatas entre R e S são removidas pela união. Serão dadas somente as fórmulas mais simples para os saltos superiores e inferiores que são respectivamente:

card(R) + card(S)

max{card(R), card(S)}

Observe que estas fórmulas assumem que R e S não contêm tuplas duplicadas.

g) **diferença:** como na união são dados somente os saltos superiores e inferiores. O salto superior da card (R-S) é card(R), considerando que o salto inferior é 0.

CAPÍTULO IV

BANCO DE DADOS ORIENTADO A OBJETO

Ao contrário de Banco de Dados Relacional, Banco de Dados Orientado a Objeto trabalha com dados complexos, cujos são classificados como imagens estáticas, vídeo, gráficos em 3D, som (voz e música) e documentos.

O capítulo IV trabalha Banco de Dados Orientado a Objeto. Este capítulo tem como objetivo descrever o processamento de consultas desta tecnologia, abordando as linguagens de consulta, a otimização e as estratégias de execução.

4. Características de Banco de Dados Orientado a Objeto

Banco de Dados Orientado a Objeto corresponde à integração da tecnologia da orientação a objetos com aptidões de banco de dados, conforme ilustra a figura 4.1. Através de construções orientadas a objetos, os usuários podem esconder os detalhes da implementação de seus módulos, compartilhar a referência a objetos e expandir seus sistemas através de módulos existentes [Khoshafian, 1994].

Na definição de Banco de Dados Orientado a Objeto (figura 4.1) as setas maiores representam "herança", portanto os bancos de dados orientados a objetos herdam características e aptidões de banco de dados.

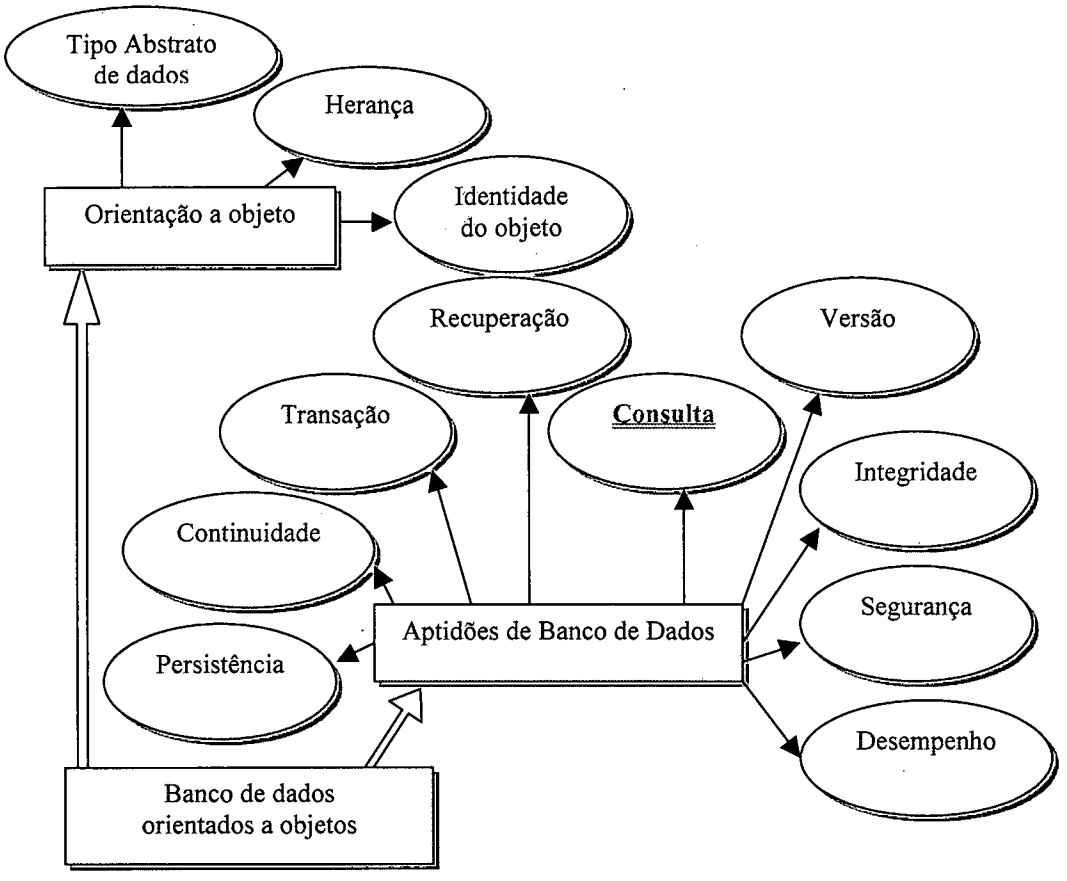

Figura 4.1 – Banco de Dados Orientado a Objeto
Fonte: KROSHAFIAN, Setrag. *Banco de dados orientado a objeto.* Rio de Janeiro: Infobook, 1994.

O uso de Banco de Dados Orientado a Objeto, ou Banco de Dados de Objetos, é um fator emergente, entretanto, o mercado de Banco de Dados ainda é dominado pelos Bancos de Dados Relacionais. Devido ao surgimento de novas técnicas de programação e modelagem, percebeu-se que o atual modelo relacional não atende as expectativas ou situações que estão ocorrendo no mundo. Com a popularização das linguagens Orientadas a Objetos e com o surgimento de técnicas de modelagem Orientadas a Objetos, questionou-se como estes aspectos seriam implantados em um Banco de Dados Relacional. Simultaneamente, os Banco de Dados Relacionais começaram a enfrentar outros problemas, como por exemplo, a falta de capacidade de manipulação de dados complexos [Khoshafian, 1994].

Os Bancos de Dados Orientados a Objetos pretendem solucionar problemas como a resistência da migração da gigantesca quantidade de dados dos sistemas relacionais para os de orientados a objetos, desempenho e escalabilidade, e maturidade tecnológica [www.members.nbci.com/_XMCM/orajac/orajac/BDOO.hym-18/09/2001].

Ao contrário de Banco de Dados Relacional, Banco de Dados Orientado a Objeto trabalha com dados complexos, cujos são classificados como imagens estáticas, vídeo, gráficos em 3D, som (voz e música) e documentos.

Alguns autores caracterizam os Bancos de Dados Orientados a Objetos como SGBD de próxima geração para aplicações avançadas. Tradicionalmente, estas aplicações avançadas incluíram projeto auxiliado por computador (CAD), manufatura auxiliada por computador (CAM) e escritórios inteligentes, o que inclui automação de escritórios e documentação de imagens [Khoshafian, 1994].

Os bancos de dados relacionais usam uma arquitetura tabular ou matricial onde os dados são referenciados através de linhas e colunas, enquanto os bancos de dados orientados a objetos podem ser inteligentes combinando lógica e os dados. Nos relacionais as tabelas são definidas tendo como base a teoria da normalização evitando a redundância dos dados e facilitando a pesquisa e atualizações. Os orientados a objetos possuem métodos, classes e outros mecanismos do modelo de orientação por objetos.

Os objetos são ativos já que podem conter lógica, enquanto os relacionais são passivos necessitando de um programa para manipular os dados. O mercado é dominado pelos relacionais, posição reforçada pelo fracasso dos bancos orientados a objetos no passado. Entretanto, com as novas aplicações multimídia e da Internet com o uso da linguagem Java os bancos de dados orientados a objetos estão se posicionando como uma boa opção tecnológica.

Os dados dos bancos relacionais são descritos em 2-D, através de linhas e colunas. A linguagem SQL é um padrão aberto para consulta e manipulação dos bancos de dados relacionais de todos os fornecedores. A SQL permite que os sistemas relacionais desenvolvidos por muitos fornecedores possam se comunicar entre si e acessar banco de dados comuns. Em contra partida, os bancos de dados orientados a objetos não possuem uma linguagem padrão dificultando a interoperacionalidade entre os bancos de dados de diferentes fornecedores.

Para definir as tabelas dos bancos relacionais é utilizado o processo de normalização, que consiste em definir nas tabelas apenas os dados que sejam únicos para a entidade descrita e definindo relacionamentos para outras tabelas também normalizadas. Os bancos relacionais estão fundamentados em uma forte teoria matemática e ferramentas bem desenvolvidas. Enquanto os bancos orientados a objetos

não possuem uma forte teoria como apoio e não existem ferramentas que descrevam o modelo de objetos.

Com o crescimento do mercado de multimídia, vídeo games e aplicações Web que utilizam a linguagem orientada a objetos Java o uso de bancos orientados a objetos está crescendo. Para atender essa demanda os fornecedores de bancos de dados relacionais estão introduzindo facilidades de armazenamento de objetos em seus bancos de dados, chamando-os de banco de dados relacional por objetos. Porém, esse tipo de banco de dados não é possível otimizar as consultas a objetos, fazer indexação direta dos objetos e usar os algoritmos de pesquisa do banco para buscar novos objetos. Somente um banco de dados orientado a objetos puro pode manipular objetos.
Os bancos de dados tradicionais foram desenvolvidos para empresas relativamente estáveis com grandes volumes de dados de baixa complexidade. Em um ambiente dinâmico com dados complexos na forma de vídeo, áudio, imagens e textos é necessário um novo modelo de banco de dados. A vantagem do banco orientado a objetos é a lógica contida no objeto e a possibilidade de ser reutilizado várias vezes em diversas aplicações.

4.1 Sistema Gerenciador de Banco de Dados Orientado a Objeto

O Sistema Gerenciador de Banco de Dados Orientado a Objeto (SGBDOO) é definido como um sistema de banco de dados que armazena além de dados comuns, como os sistemas de arquivos e estruturas de dados em Banco de Dados Relacional, outros tipos de dados que não podem ser convertidos somente em arquivos lineares ou bidimensionais, e sim um como tipo especial de objetos [Celko et. al., 1997].

A principal característica do SGBDOO é a capacidade de modelar estruturas complexas, armazenando não somente a estrutura de dados, mas também seu comportamento. Os tipos de dados complexos são representados como objetos que encapsulam os detalhes dos dados, estruturas e métodos que dizem respeito à forma como o objeto interage com outros objetos [www.unifieo.br/revista/banco.html - 18/09/2001]. Os SGBDOOs, ao contrário dos SGBDRs, não se beneficiam da definição

inicial de um modelo de consulta preciso e formal e de um conjunto de primitivas algébricas.

Os SGBDOOs ainda estão em desenvolvimento e não possuem um modelo definido, por este motivo, é pouco usado nas empresas. No entanto, o surgimento cada vez maior de banco de dados não convencionais para aplicações específicas aumenta o valor e o interesse para a tecnologia orientada a objeto.

Como qualquer tecnologia, o SGBDOO possui vantagens e desvantagens. Os aspectos caracterizados como vantagens são:

- as aplicações utilizadas na Internet são particularmente adequadas para Banco de Dados de Objetos, já que a maioria das aplicações é desenvolvida em Java, que é uma linguagem Orientada a Objeto;
- o SGBDOO é ideal para as aplicações mais populares. O crescimento da Internet, videogames, aplicações para multimídia e o desenvolvimento de banco de dados distribuído, que não se adequam ao modelo relacional, estão trazendo o foco para o SGBDOO menos complexo, do ponto de vista do desenvolvimento e utilização;
- o SGBDOO segue os princípios das atuais linguagens de programação.

As desvantagens do SGBDOO são:

- o SGBDOO não possui embasamento teórico, como o caso do banco de dados hierárquico baseados na metodologia de "árvore", os bancos de dados em rede baseados em "grafo" e os bancos de dados relacionais baseados em "matemática dos conjuntos". Porém, nem tudo que não tem embasamento teórico é inútil, por exemplo, o cálculo numérico é utilizado até os dias de hoje;
- possui poucos recursos de ferramentas gráficas para o desenvolvimento;
- é instável com relação ao direcionamento de suas aplicações, já que tudo se resume em objetos e as linguagens para consultas de objetos são difíceis e nem um pouco padronizadas;
- é pouco explorado por ser uma nova tecnologia.

4.2 Armazenamento de Objetos

Entre muitas características relacionadas ao armazenamento de objetos, as mais relevantes são o agrupamento de objetos em *clusters* e a coleta de lixo distribuída. Os objetos compostos e complexos fornecem oportunidades que agrupam os dados em *clusters* no disco, reduzindo o custo de E/S de sua recuperação. A coleta de lixo é um problema que surge em banco de dados de objetos, porque permite o compartilhamento baseado em referências. Portanto, a eliminação de objetos e a recuperação do espaço de armazenamento exigem cuidados especiais [Özsu e Valduriez, 1999].

4.3 Processamento de Consultas

No processamento de consultas em banco de dados de objetos existe um forte relacionamento entre as técnicas de otimização de consulta, o modelo de consulta e a linguagem de consulta. O modelo de consulta é baseado no modelo de objetos que define as primitivas de acesso usadas pelo modelo de consultas. Essas primitivas determinam, pelo menos parcialmente, a potência do modelo de consulta.

A maioria dos processadores de consultas de objetos utiliza as mesmas técnicas de otimização desenvolvidas para sistemas relacionais, embora, com algumas dificuldades, como:

- as linguagens de consultas relacionais operam sobre sistemas simples que consistem em relações. Já em linguagens de consultas de objetos os sistemas são enriquecidos. Os resultados dos operadores da álgebra de objetos normalmente são conjuntos de objetos de tipos diferentes. É exigindo o desenvolvimento de esquemas elaborados de inferência de tipo para determinar quais métodos podem ser aplicados a todos os objetos em tal conjunto. As álgebras de objetos, freqüentemente, operam sobre tipos de coleções semanticamente distintos, o que impõem requisitos adicionais sobre o esquema de inferência para determinar o tipo dos resultados de operações sobre as coleções diferentes.

- A otimização de consultas relacionais depende do conhecimento do espaço de armazenamento físico de dados (caminhos de acesso), que está prontamente disponível para o otimizador de consultas. O encapsulamento de métodos com os dados, sobre os quais operam em SGBDOOs, cria questões importantes, como: a) estimar o custo da execução de métodos, sendo então consideravelmente mais difícil que calcular ou estimar o custo para acessar um atributo de acordo com um caminho de acesso; e b) gerar questões relacionadas à acessibilidade de informações de armazenamento pelo otimizador de consultas. Alguns sistemas superam essas dificuldades tratando o otimizador de consultas como um aplicativo especial que pode quebrar o encapsulamento e acessar informações diretamente. Outros propõem um mecanismo pelo qual os objetos revelam seus custos como parte de sua interface.
- Os objetos podem ter estruturas complexas através das quais o estado de um objeto se refere a outro. O acesso a esses objetos complexos envolve expressões de caminho. A otimização de expressões de caminho é uma questão importante e difícil de ser tratada em linguagens de consulta de objetos. Além disso, os objetos pertencem a tipos relacionados entre si através de hierarquias de herança. A otimização do acesso a objetos por suas hierarquias de herança também é um problema que distingue o processamento de consultas orientadas a objetos do processamento de consultas relacionais.
- Em SGBDOOs ocorre a falta de uma definição de modelo de objetos universalmente aceita. Embora exista alguma convergência sobre o conjunto de características básicas que devem ser admitidas por um modelo de objetos, como: identidade de objetos, encapsulamento do estado e comportamento, herança de tipo e coleções tipificadas. O modo como essas características são aceitas difere entre modelos e sistemas. Os projetos que fazem experiências com otimizadores de consultas de objetos seguem caminhos bastante diferentes e são até certo ponto incompatíveis, o que dificulta o aproveitamento das experiências de outros. Como essa diversidade de abordagens talvez prevaleça, por algum tempo, a abordagem extensível para otimização de consultas. Essa deve ser explorada para permitir a experimentação com novas idéias à medida que elas forem evoluindo, portanto são importantes no processamento de consultas de objetos.

O processamento e a otimização de consultas de objetos têm sido assunto de pesquisas significativas nos últimos tempos.

4.3.1 Linguagem de Consulta

Uma linguagem de banco de dados completa, tal como SQL, consiste de três sublinguagens: linguagem de definição de dados (DDL), linguagem de consulta e manipulação de dados (DML) e linguagem de controle de dados (DCL) [Kim, 1993].

Em SGBDOO, a DDL e a DML devem ser mais completas do que em banco de dados relacionais, uma vez que os SGBDOOs suportam métodos, objetos complexos, herança, relacionamento entre tipos, encapsulamento e identidade de objetos, além de contemplar as funções específicas de suporte a objetos multimídia.

As consultas são expressas em linguagem declarativa, não requerendo nenhum conhecimento prévio do usuário sobre a implementação de objetos, caminhos de acesso ou estratégias de processamento.

Existem diferentes enfoques na definição e implementação das linguagens de consultas em SGBDOOs. Uma linguagem de consulta pode ser considerada de duas maneiras: a) como uma linguagem de consulta interativa, para a execução de consultas *ad hoc*, capaz de suportar a manutenção do banco de dados e consultá-lo através de métodos de instanciação, recuperação e atualização dos objetos do banco de dados, garantindo a sua consistência. É importante que essa linguagem ofereça grande poder de expressividade e eficiência de execução; e b) embutida numa linguagem hospedeira, de forma a prover um ambiente de desenvolvimento para aplicações de âmbito geral, onde a linguagem deve suprir adicionalmente o controle de processamento de transações [Collyer Junior, 1997].

4.3.2 Etapas do Processamento de Consultas

Primeiramente, a expressão de cálculo é reduzida a uma forma normalizada com o intuito de eliminar os predicados duplicados. A expressão normalizada é então convertida a uma expressão algébrica de objeto equivalente. Esta forma de consulta é uma expressão aninhada, expressa através de uma árvore de predicados, onde seus nós são operadores algébricos e suas folhas representam extensões de classes no banco de dados. A expressão algébrica é validada, garantindo que os predicados e métodos não sejam aplicados a objetos que não suportam as funções requeridas. Este procedimento não é tão simples como validar tipos nas linguagens de programação, já que os resultados intermediários, que também são conjuntos de objetos, podem ser constituídos de tipos heterogêneos.

O próximo passo no processamento de consultas é aplicar as regras de reescrita com preservação da equivalência, garantindo a consistência de tipos em expressões algébricas.

O último passo é a execução do plano, gerado a partir da expressão algébrica otimizada [Freytag, 1994]. A figura 4.2 apresenta as etapas relativas ao processamento de consultas em SGBDOOs.

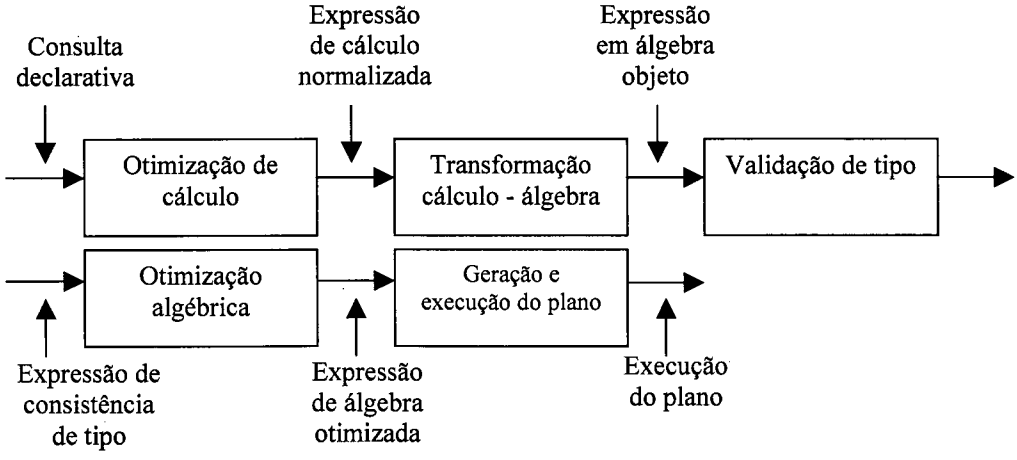

Figura 4.2 – Etapas do processamento de consulta em um SGBDOO
Fonte: COLLYER JUNIOR, Charles Sampaio. *Linguagem de consulta ao SIGO: um sistema gerenciador de objetos.* Rio de Janeiro: Instituto Militar de Engenharia, 1997.

4.3.3 Otimização de Consultas

Muitos otimizadores do SGBDOOs existentes são implementados como parte do gerenciador de objetos sobre um sistema de armazenamento. Na maioria dos casos, os componentes são embutidos no código do otimizador de consultas. Considerando-se que a extensibilidade é uma meta importante dos SGBDOOs, espera-se que seja desenvolvido um otimizador extensível que acomode diferentes estratégias de pesquisa, especificações de álgebra e funções de custo [Özsu e Valduriez, 1999].

O processamento de consultas em SGBDOOs é bastante semelhante ao SGBDR, as diferenças aparecem no resultado das características do modelo de objetos e do modelo de consultas. A otimização algébrica e a execução de expressões de caminho são diferenças significativas em SGBDOOs.

4.3.3.1 Otimização Algébrica

Na otimização algébrica uma expressão de consulta algébrica pode ser transformada com o uso de propriedades algébricas bem definidas, como transitividade, comutatividade e distributividade. Durante esse processo de transformação, são eliminados os piores planos de execução [Özsu e Valduriez, 1999]. Na otimização algébrica são abordados aspectos como: regras de transformação, algoritmos de pesquisa, funções de custo e parametrização.

As regras de transformação são dependentes da álgebra de objetos específica, pois são definidas de forma individual para cada álgebra de objetos e suas combinações. A falta de uma definição de álgebra de objetos padrão é desvantajosa, porque a comunidade não pode se beneficiar das generalizações dos numerosos estudos. A definição das regras de transformação e a manipulação das expressões de consulta são semelhantes às de sistemas relacionais, com uma diferença, as expressões de consultas relacionais são definidas sobre relações planas, enquanto que as consultas de objetos são definidas sobre classes, coleções ou conjuntos de objetos, que têm relacionamentos de

subtipificação e composição entre elas. Dessa forma, pode-se usar a semântica desses relacionamentos em otimizadores de consultas de objeto para obter algumas transformações adicionais.

Considere, como exemplo, três operadores da álgebra de objetos: união ($\cup$), interseção ($\cap$) e seleção (σ), onde os operadores de união e interseção têm a semântica usual da teoria de conjuntos e o operador de seleção seleciona os objetos de um conjunto P, usando os conjuntos de objetos Q_1 ... Q_k como parâmetros. Algumas regras de transformação podem ser aplicadas durante a otimização, obtendo expressões de consulta equivalente. Essas regras são descritas a seguir, para facilitar será usada a expressão QSet para denotar Q_1 ... Q_k e RSet para denotar R_1 ... R_k).

1.ª Regra: $(P\sigma_{F1} < QSet >)\ \sigma_{F2} < RSet> \quad \Leftrightarrow \quad (P\sigma_{F2} < RSet >)\ \sigma_{F1} < QSet>$

2.ª Regra: $(P \cup Q)\ \sigma_F <RSet> \quad \Leftrightarrow \quad (P\sigma_F < RSet >) \cup (Q\sigma_F < RSet >)$

3.ª Regra: $(P\sigma_{F1} < QSet >)\ \sigma_{F2} < RSet> \quad \Leftrightarrow \quad (P\sigma_{F1} < QSet >) \cup (P\sigma_{F2} < RSet >)$

A primeira regras capta a comutatividade de seleção, a segunda regra denota que seleção se distribui sobre união e a terceira regra é uma identidade que utiliza o fato de que seleção apenas restringe sua entrada e retorna um subconjunto de seu primeiro argumento. A primeira e a segunda regras são gerais, no sentido de que representam equivalências herdadas da teoria dos conjuntos. Já a terceira regra de transformação é especial para um operador de álgebra de objetos específico definido com uma semântica específica. Entretanto, todas as três regras são sintáticas por natureza.

Considere as regras a seguir, onde C_i denota o conjunto de objetos na extensão da classe C_i, e C^*_j denota a extensão profunda de C_j, isto é, o conjunto de objetos na extensão de C_j e também nas extensões de todas as subclasses de C_j.

1.ª Regra: $C_1 \cap C_2 \quad = \quad \varnothing$ se $C_1 \neq C_2$

2.ª Regra: $C_1 \cap C_2^* \quad = \quad C_2^*$ se C_1 é uma subclasse de C_2

3.ª Regra: $(P\sigma_F <QSet>) \cap R \quad \Leftrightarrow^c \quad (P\sigma_F <QSet>) \cap (R\sigma_{F'} <QSet>)$

$\Leftrightarrow^c \quad P \cap (R\sigma_{F'} <QSet>)$

Essas regras de transformação são semânticas por natureza, pois dependem das especificações do modelo de objetos e do modelo de consultas. A primeira regra é verdadeira porque o modelo de objetos limita cada objeto que pertence a apenas uma classe. A segunda regra também é válida porque o modelo de consultas permite a recuperação de objetos na extensão profunda da classe de destino. A terceira regra se

baseia nas regras de consistência de tipos para sua aplicabilidade, bem como, em uma condição denotada por C sobre o símbolo ⇔ de que F′ é idêntico a F, exceto pelo fato de cada ocorrência de P ser substituída por R.

Para realizar a otimização algébrica os algoritmos considerados são os de pesquisa exaustiva, pesquisa enumerativos e pesquisa aleatória. Os algoritmos de pesquisa exaustiva enumeram todo o espaço de pesquisa, aplicando uma função de custo a cada expressão equivalente para determinar a menos dispendiosa. A melhora é para ser usada na abordagem de programação dinâmica, onde novas expressões são construídas com um enfoque *bottom-up*, utilizando subexpressões ótimas determinadas previamente. Os algoritmos, cujo gerador de otimizadores emprega uma abordagem *top-down* de programação dinâmica para pesquisa com supressão de desvio e limite, são chamados de algoritmos enumerativos. A natureza combinatória dos algoritmos de pesquisa enumerativos talvez seja mais importante em SGBDOOs do que em SGBDRs. Nesses algoritmos se o número de junções em uma consulta exceder a dez, as estratégias de pesquisa enumerativas se tornam inviáveis. Os algoritmos de pesquisa aleatória foram sugeridos como alternativas para limitar a região do espaço de pesquisa sob análise. Infelizmente, não houve qualquer estudo dos algoritmos de pesquisa aleatória o contexto de SGBDOOs. As estratégias gerais não devem mudar, mais o ajuste dos parâmetros e a definição do espaço de soluções aceitáveis devem se alterar. Estudos adicionais são necessários para estabelecer o relacionamento de modo mais firme. Além disso, as versões distribuídas desses algoritmos não estão disponíveis, e seu desenvolvimento continua a ser um desafio.

Com relação às funções de custo, os argumentos baseiam-se em diversas informações a respeito do armazenamento dos dados. O otimizador de consultas em sistemas relacionais considera o número de itens de dados (cardinalidade), o tamanho de cada item de dados e sua organização, entretanto, essas informações não estão disponíveis nos SGBDOOs. Pesquisadores debatem se o otimizador de consultas em SGBDOOs deve ser capaz de romper o encapsulamento de objetos e examinar as estruturas de dados usadas para implementá-los. Se isso for permitido, as funções de custo poderão ser especificadas de modo semelhante aos sistemas relacionais. Caso contrário, uma especificação alternativa deve ser considerada. A função de custo pode ser definida recursivamente com base na árvore de processamento algébrico. Se a

estrutura interna de objetos não for visível para o otimizador de consultas, o custo de cada nó terá de ser definido. Um modo de defini-lo é fazer os objetos revelarem seus custos como parte de sua interface. A definição de funções de custo, especialmente nas abordagens baseadas em objetos que revelam seus custos, deve ser pesquisada com maior profundidade antes de fazer conclusões satisfatórias.

Na parametrização observa-se que a otimização de consultas em tempo de compilação é um processo estático, cujo otimizador utiliza as estatísticas do banco de dados. No momento em que a consulta é compilada e otimizada, o melhor plano de execução é selecionado. Essa decisão é independente das estatísticas de tempo de execução, como a carga do sistema. Além disso, ela não considera as mudanças das estatísticas do banco de dados como resultado de atualizações, que podem ocorrer entre o momento em que a consulta é otimizada e o momento em que ela é executada. Essa característica é um problema, especialmente em consultas de tipo de produção que são otimizadas uma única vez e executadas várias vezes. Esse assunto é uma questão mais séria em SGBDOOs. Os bancos de dados orientados a objetos são por definição mais voláteis, resultando em mudanças significativas no banco de dados, consequentemente buscam a evolução do esquema dinâmico, esquema esse tão importante em SGBDOOs. A estratégia de otimização de consultas tem de ser capaz de lidar com essas mudanças.

A questão pode ser tratada de duas maneiras:

a) determinar um intervalo de otimização/nova, otimização, e voltar a otimizar a consulta periodicamente. Embora essa seja uma abordagem simples, ela se baseia em um intervalo de tempo fixo, cuja determinação provavelmente seria problemática. Uma ligeira variação pode determinar o ponto de nova otimização, com base na diferença entre o tempo de execução real e o tempo de execução estimada. Consequentemente, o sistema *runtime* poderá controlar o tempo de execução real e, sempre que ele divergir do tempo estimado, além de um limiar fixo, a consulta será novamente otimizada. A determinação desse limiar seria uma preocupação, como também uma sobrecarga *runtime* de execução da consulta de rastreamento;
b) realizar a otimização de consultas paramétricas, também chamada de seleção de plano dinâmico. Nesse caso, o otimizador mantém várias estratégias de execução em tempo de compilação e cria uma seleção de plano final em tempo de execução,

baseada em vários parâmetros do sistema e em estatísticas atuais do banco de dados. Se o otimizador não tiver acesso a todos esses dados, a otimização algébrica poderá ignorar todas as características de execução física, ao invés de gerar um conjunto de expressões de consultas equivalentes "desejáveis", que são repassadas ao gerenciador de objetos. O gerenciador de objetos poderá então comparar as alternativas, em tempo de execução, com base em suas características de execução. Porém, essa abordagem também apresenta um problema significativo, isto é, poderá incorrer potencialmente em uma sobrecarga *runtime* elevada. O problema com a otimização paramétrica em tempo de compilação é a explosão potencial exponencial dos planos dinâmicos, como uma função da complexidade da consulta e também do número de parâmetros de otimização desconhecidos em tempo de compilação. Esse problema, juntamente com os problemas de programação de erros e imprecisão da seletividade e dos métodos de estimativa de custo, torna a otimização de consultas "*runtime*" uma alternativa atraente.

4.3.3.2 Expressões de Caminho

Em SGBDOOs pode-se navegar de um objeto para outro, usando expressões de caminhos que envolvem atributos com valores baseados em objetos. Por exemplo, considere C como do tipo carro, então c.motor.fabricante.nome é uma expressão de caminho. Deve-se supor que o tipo motor é definido com pelo menos um atributo, fabricante, cujo domínio é a extensão do tipo fabricante. O tipo fabricante tem um atributo chamado nome.

A maioria das linguagens de consulta de objetos permite consultas cujos predicados envolvem condições sobre o acesso a objetos ao longo de cadeias de referência. Essas cadeias de referência são chamadas expressões de caminho. O exemplo anterior de expressão de caminho, c.motor.fabricante.nome, recupera o valor do atributo nome do objeto, que é o valor do atributo fabricante do objeto e também o valor do atributo motor do objeto C, definido como um objeto do tipo carro. É possível formar expressões de caminho envolvendo atributos, bem como métodos. A otimização

da computação de expressões de caminho é um problema que recebeu atenção substancial no processamento de consultas de objetos.

As expressões de caminho permitem uma notação sucinta e de alto nível para expressar a navegação através do grafo de composição de objetos (agregação), o que possibilita a formulação de predicados sobre valores profundamente aninhados na estrutura de um objeto. Elas oferecem um mecanismo uniforme para a formulação de consultas que envolvem composição de objetos e funções de membros herdadas. As expressões de caminho podem ser de valor único ou de valor de conjunto e podem aparecer em uma consulta como parte de um predicado, um destino para uma consulta, quando têm valor de conjunto, ou como parte de uma lista de projeção. Uma expressão de caminho é de valor único se todo componente de uma expressão de caminho é de valor único, se pelo menos um componente é de valor de conjunto, a expressão de caminho inteira é de valor de conjunto.

O problema para otimizar expressões de caminho se estende por todo o processo de compilação de consultas. Durante ou após a análise de uma consulta do usuário, mas antes da otimização algébrica, o compilador de consulta deve reconhecer quais expressões de caminho podem ser potencialmente otimizadas. Geralmente, isso é possível através de técnicas de reescrita, que transformam as expressões de caminho em expressões lógicas equivalentes de álgebra. Uma vez que as expressões de caminho são representadas em forma algébrica, o otimizador de consultas explora o espaço de planos algébricos equivalentes e de execução, procurando por um plano de custo mínimo. Finalmente, o ótimo plano de execução pode envolver algoritmos para calcular de modo eficiente as expressões de caminho, inclusive junção de *hash*, montagem de objetos complexos ou varredura indexada através de índices de caminho.

Com relação à reescrita e otimização algébrica, considera-se novamente a expressão de caminho c.motor.fabricante.nome descrito no exemplo anterior. Suponha-se que cada instância de carro tenha uma referência a um objeto motor, que cada motor tenha uma referência a um objeto fabricante e que cada instância de fabricante tenha um campo nome. Além disso, suponha-se que os tipos motor e fabricante tenham uma extensão de tipo correspondente. Os dois primeiros vínculos do caminho anterior podem envolver a recuperação de objetos motor e fabricante a partir do disco. O terceiro caminho envolve apenas uma pesquisa de um campo dentro de um objeto fabricante.

Assim, somente os dois primeiros vínculos apresentam oportunidades para otimização de consultas na computação desse caminho. Um compilador de consulta de objetos precisa de um mecanismo que distingue esses vínculos em um caminho que representa otimizações possíveis. Geralmente, isso é obtido através de uma fase de reescrita.

Uma alternativa é usar uma técnica de reescrita baseada em tipos. Essa abordagem unifica as técnicas algébricas baseadas em tipos para reescrita, permite a fatoração de subexpressões comuns e admite heurísticas para limitar a reescrita. As informações de tipos são exploradas para decompor argumentos iniciais complexos de uma consulta em um conjunto de operadores mais simples, e para reescrever expressões de caminho em junções.

Um operador alternativo proposto para otimizar expressões de caminho é materializar, que representa a computação de cada referência entre objetos, isto é, vínculo de caminho de modo explícito. Isso permite a um otimizador de consultas expressar a materialização de vários componentes como um grupo usando um único operador Mat, ou individualmente, com o uso de um operador Mat por componente. Outra maneira de imaginar esse operador é como uma definição de escopo, porque ele traz elementos de uma expressão de caminho para o escopo, de modo que esses elementos possam ser usados em operações posteriores ou na avaliação de predicados. As regras de escopo são tais que um objeto componente entra no escopo ao ser varrido ou ao ser referenciado. Os componentes permanecem no escopo até uma projeção descartá-los. O operador materializar permite a um processador de consultas agregar todas as materializações de componentes exigidas para a computação de uma consulta, não importando se os componentes são necessários para a avaliação de predicados ou para produzir o resultado de uma consulta. O propósito do operador materializar é indicar ao otimizador onde as expressões de caminho são usadas e onde as transformações algébricas podem ser aplicadas. São definidas várias regras de transformação envolvendo Mat.

Enfocando índices de caminho, uma pesquisa substancial sobre otimização de consultas de objetos foi dedicada ao projeto de estruturas de índices para acelerar a computação de expressões de caminho.

A computação de expressões de caminho através de índices representa apenas uma classe de algoritmos de execução de consultas usados na otimização de consultas

de objetos. Em outras palavras, a computação eficiente de expressões de caminho através de índices de caminhos representa apenas uma coleção de opções de implementação para operadores algébricos, como materialização e junção, usados para representar referências entre objetos.

4.3.4 Estratégias de Execução de Consulta

Os SGBDRs se beneficiam da correspondência íntima entre as operações da álgebra relacional e as primitivas de acesso do sistema de armazenamento. Portanto, a geração do plano de execução para uma expressão de consulta se preocupa basicamente com a escolha e a implementação dos algoritmos mais eficientes para executar os operadores individuais da álgebra e suas combinações. Em SGBDOOs a questão é mais complicada, devido à diferença nos níveis de abstração de objetos definidos de forma comportamental e seu armazenamento. O encapsulamento de objetos, que oculta seus detalhes de implementação, e o armazenamento de métodos com objetos geram um problema de projeto desafiante, que pode ser enunciado assim: "Em que ponto no processamento de consultas o otimizador de consultas deve acessar informações a respeito do armazenamento de objetos?" Uma alternativa é deixar essa tarefa para o gerenciador de objetos. Consequentemente, o plano de execução da consulta, gerado a partir da expressão de consulta, é obtido no final da etapa de reescrita, pelo mapeamento da expressão de consulta em um conjunto bem definido de chamadas de interface do gerenciador de objetos. A interface do gerenciador de objetos consiste em um conjunto de algoritmos de execução. Esta seção revê alguns algoritmos de execução que provavelmente farão parte de futuros mecanismos de execução de consultas de objetos de alto desempenho.

Um mecanismo de execução de consulta exige três classes básicas de algoritmos sobre coleções de objetos: varredura de coleção, varredura indexada e correspondência de coleção. A varredura de coleção é um algoritmo direto que acessa seqüencialmente todos os objetos de uma coleção. A varredura indexada permite o acesso eficiente a objetos selecionados de uma coleção através de um índice. É possível usar um campo de

um objeto ou os valores retornados por algum método como uma chave para um índice. Além disso, é possível definir índices sobre valores profundamente aninhados na estrutura de um objeto, isto é, índices de caminho. Os algoritmos de correspondência de conjuntos tomam várias coleções de objetos como entrada e produzem objetos de agregados inter-relacionados por alguns critérios. A junção, a interseção de conjuntos e a montagem são exemplos de algoritmos dessa categoria.

Como mencionado na seção anterior, o suporte para expressões de caminho é uma característica que distingue consultas de objetos de consultas relacionais. Muitas técnicas de indexação projetadas para acelerar a computação de expressões de caminho foram propostas com base no conceito de índice de junção.

As relações de suporte de acesso constituem uma técnica geral alternativa para representar e calcular expressões de caminho. Uma relação de suporte de acesso é uma estrutura de dados que armazena expressões de caminho selecionadas. Essas expressões de caminho são escolhidas para serem aquelas onde ocorre a navegação mais freqüente. Os estudos fornecem a evidência inicial de que o desempenho de consultas executadas com a utilização de relações de suporte de acesso é cerca de duas ordens de magnitude melhor que as consultas que não empregam relações de suporte de acesso. Um sistema que usa relações de suporte de acesso também deve levar em consideração o custo de mantê-las na presença de atualizações para as relações básicas subjacentes.

As expressões de caminho são percursos ao longo de relacionamentos de composição de objetos compostos. Como já comentado, um modo possível de executar uma expressão de caminho é transformá-la em uma junção entre os conjuntos de origem e destino de objetos. Foram propostos vários algoritmos de junção diferentes, como junção de *hash* híbrido ou junção de *hash* baseado em ponteiros. O primeiro utiliza o princípio de dividir e conquistar para particionar recursivamente as duas coleções operandos em depósitos, com o uso de uma função de *hash* sobre o atributo de junção. Cada um desses depósitos pode caber inteiramente na memória. Cada par de depósito é unido por junção na memória para produzir o resultado. A junção de *hash* baseado em ponteiros é usada quando cada objeto em uma coleção operando, chamada R, tem um ponteiro para um objeto na outra coleção operando, chamada S. O algoritmo segue três etapas:

a) o particionamento de R do mesmo modo que no algoritmo de *hash* híbrido, exceto pelo fato de ser particionado por valores de OIDs, em vez de ser particionado por valore do atributo de junção. O conjunto de objetos S não é particionado.
b) cada partição R_i de R é unida por junção a S, tomando-se R_i e construindo-se uma tabela de *hash* para ela na memória. A tabela é construída através do *hash* de cada objeto $r \in R$ sobre o valor de seu ponteiro ao objeto correspondente a ele em S. Como resultado, todos os objetos de R que se referem à mesma página em S são agrupados na mesma entrada da tabela de *hash*.
c) após a construção da tabela de *hash* de R_i, cada uma de suas entradas é varrida. Para cada entrada de *hash*, a página correspondente em S é lida, e todos os objetos de R que se referem a essa página são unidos por junção aos objetos correspondentes de S.

Um método alternativo de algoritmo de execução de junção, a montagem, é uma generalização do algoritmo de junção de *hash* baseado em ponteiros para o caso em que é necessário calcular uma junção de vários modos. Essa operação monta de forma eficiente os fragmentos de estados de objetos necessários para uma determinada etapa de processamento e os retorna como um objeto complexo na memória. Ela converte as representações de disco de objetos complexos em representações de memória que podem ser prontamente percorridas.

A montagem de um objeto complexo arraigado em objetos de tipo R que contém objetos componentes dos tipos S, U e T é análoga à computação de uma junção de quatro modos desses conjuntos. Existe uma diferença entre montagem e junção de ponteiro de n modos, porque a montagem não precisa que a coleção inteira de objetos de raiz seja varrida antes de produzir um único resultado.

Ao invés de montar um único objeto complexo de cada vez, o operador de montagem monta uma janela de tamanho W de objetos complexos ou mesmo tempo. Logo que qualquer desses objetos complexos é montado e repassado para cima na árvore de execução de consulta, o operador de montagem recupera outro objeto para trabalhar. O uso de uma janela de objetos complexos aumenta o tamanho do *pool* de referências não resolvidas e resulta em mais opções para otimização de acessos ao disco. Devido ao caráter aleatório com que as referências são resolvidas, o operador de montagem entrega objetos montados ao acesso à árvore de execução de consulta. Esse

comportamento é correto no processamento de consultas orientadas a conjuntos, mas pode não ser para outros tipos de coleções, como listas.

O objetivo do algoritmo de montagem é montar simultaneamente uma janela de objetos complexos. Em cada ponto no algoritmo, é escolhida a referência pendente que otimiza os acessos de disco. Há diferentes ordens, ou escalonamentos, nos quais as referências podem ser resolvidas, como primeiro em profundidade, primeiro em extensão e elevador. Os resultados de desempenho indicam que o elevador supera os algoritmos primeiro em profundidade e primeiro em extensão sob diversas situações de agrupamento de dados em *clusters*.

Existem algumas possibilidades de implementação de uma versão distribuída dessa operação, podendo ser:

- A primeira estratégia envolve o transporte de todos os dados até um site central para processamento. Essa é simples de implementar, embora possa ser ineficiente no caso geral.
- A segunda estratégia envolve a realização de operações simples em sites remotos, transportando, posteriormente, todos os dados até um site central para montagem final. Essa estratégia final também exige controle simples, pois toda a comunicação ocorre através do site central. Um exemplo de operações simples seria seleções e montagem local.
- A terceira estratégia é significativamente mais complicada. Ela deve executar as operações complexas em sites remotos e depois transportar os resultados até o site central para montagem final. Um exemplo de operações complexas seria junções e montagem completa de objetos remotos.

Um SGBD de objetos distribuídos pode incluir todas ou apenas algumas dessas estratégias.

O exemplo ilustrado na figura 4.3 monta um conjunto de objetos Carro. Os retângulos da figura representam instâncias de tipos indicados à esquerda, e as arestas denotam os relacionamentos de composição. Considere um atributo de cada objeto do tipo Carro que aponta para um objeto do tipo Motor. Suponha que a montagem esteja usando uma janela de tamanho 2. O operador de montagem começa preenchendo a janela com duas referências ao objeto Carro do conjunto, pois W = 2.

Observando a figura 4.4(a) nota-se que o operador de montagem se inicia escolhendo uma das referências pendentes no momento, ou seja, C1. Depois de buscar C1, duas novas referências não resolvidas são somadas à lista, conforme a figura 4.4(b). A resolução de C2 resulta em mais duas referências adicionadas à lista e assim por diante (figura 4.4(c)), até o primeiro objeto complexo ser montado (figura 4.4 (d)). Nesse ponto, o objeto montado é repassado para cima na árvore de execução de consulta, liberando algum espaço na janela. Uma nova referência ao objeto Carro, C3, é adicionada à lista e depois resolvida, trazendo duas novas referências E3, B3, conforme figura 4.4 (h).

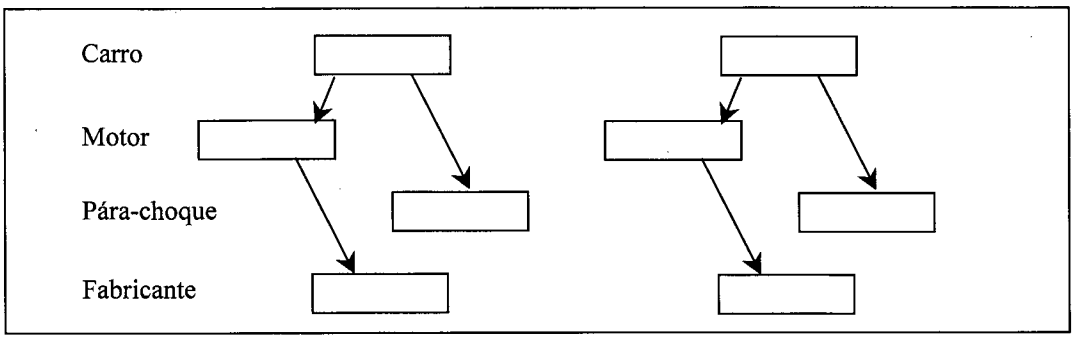

Figura 4.3 – Dois Objetos complexos montados
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

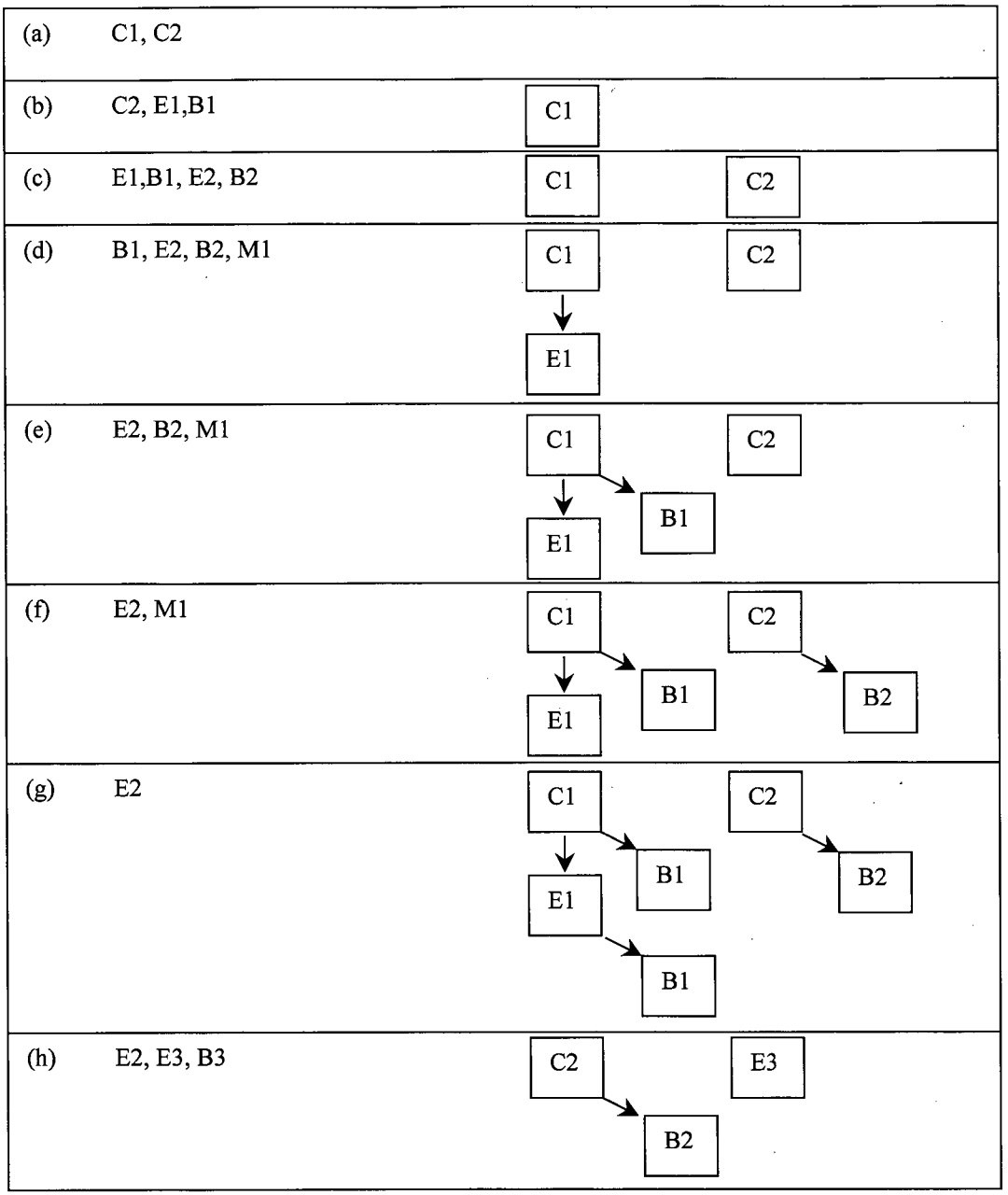

Figura 4.4 – Exemplo de Montagem
Fonte: ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

CAPÍTULO V

BANCO DE DADOS OBJETO-RELACIONAL

O Banco de Dados Objeto-relacional têm capacidade de manipular dados complexos aliados à flexibilidade de execução de consultas típicas de sistemas relacionais. Para que o SGBDR possa suportar a manipulação de dados complexos é necessário adotar algumas características do SGBDOO, tornando-se então, Sistema Gerenciador de Banco de Dados Objeto-relacional (SGBDRO). Este capítulo traz algumas definições de Banco de Dados Objeto-relacional, ilustra uma comparação sucinta entre Banco de Dados Relacional, Banco de Dados Orientado a Objeto e Banco de Dados Objeto-relacional e finalmente aborda as particularidades do processamento de consultas dessa tecnologia.

5. Características do Banco de Dados

O Banco de Dados Objeto-relacional é o caminho natural para a transição das aplicações e modelos do mundo relacional para o mundo de objetos [members.nbci.com /_XMCM/orajac /orajac/BDOO.htm - 18/09/2001].

Os modelos de dados objeto-relacionais estendem o modelo relacional fornecendo um tipo de sistema mais rico, incluindo orientação a objeto e acrescentando estruturas às linguagens de consultas relacionais, como a SQL, para tratar os tipos de dados acrescentados. Os sistemas de banco de dados objeto-relacionais fornecem um

caminho de migração conveniente para os usuários de banco de dados relacionais que desejam usar características orientadas a objeto [Silberschatz et al., 1999].

O Sistema Gerenciador de Banco de Dados Relacional (SGBDR) é uma tecnologia que armazena e gerencia entidades relacionadas entre si, visando prover dados mais confiáveis para evitar a duplicação dos dados, e consequentemente otimizar o espaço em disco. Apesar de todas as características vantajosas, o SGBDR não atende as expectativas quando se trata de tipos de dados complexos como imagens ou entradas em séries, que precisam de codificações binárias especiais para representar os dados. Estes tipos de dados complexos são representados como objetos que encapsulam os detalhes dos dados, estruturas e métodos, que dizem respeito à forma como o objeto interage com outros. Para que o SGBDR possa suportar todas estas funções é necessário adotar algumas características do Sistema Gerenciador de Banco de Dados Orientado a Objeto (SGBDOO), tornando-se então, Sistema Gerenciador de Banco de Dados Objeto-relacional (SGBDRO), sistema este que suporta todas as funções relacionais tradicionais, oferecendo ainda a possibilidade de gerenciar com consistência os objetos empresariais heterogêneos [www.unifieo.br /revista/banco.html - 18/09/2001].

Aplicações com dados simples e consultas que são facilmente expressas por linguagens de consultas, como SQL, podem ser identificadas como aplicações de processamento de negócios, e normalmente são implementadas utilizando SGBDRs. Já o modelo orientado a objeto é baseado em extensões do conceito de tipos abstratos de dados. Os SGBDs baseados neste modelo foram propostos para atender às necessidades de novas aplicações, incluindo objetos com estruturas complexas, novos tipos de dados e a definição de operações específicas das aplicações. O modelo de dados orientado a objeto oferece a flexibilidade de manipular estes requisitos sem, no entanto, estar limitado aos tipos de dados e linguagens de consultas disponíveis nos sistemas de banco de dados tradicionais.

Os SGBDROs com um modelo de dados relacional estendido surgiram para representar as semânticas e construções de modelagens. Estes SGBDs nasceram pela necessidade das aplicações em utilizar certas características orientadas a objetos sem desconsiderar os grandes investimentos que foram feitos nos SGBDRs.

Existem algumas características para que os SGBDROs se tornem habilitados para objetos, como: a) possuir técnicas de indexação e armazenamento feitas sob

medida para cada estrutura de dados; b) possibilitar a recuperação baseada em conteúdos, pois essa requer métodos espaciais; e c) propiciar a máxima performace na busca e recuperação de dados complexos. A otimização de consultas e o processo de recuperação precisam ser adaptados para o tipo de dado que está sendo recuperado. O custo de recursos de sistemas em termos de consultas complexas para a recuperação de desenhos de CAD/CAM é bastante diferente da recuperação de linhas e colunas. Essa diferença de custo aumenta substancialmente se os objetos de dados que estão sendo procurados residem em um ambiente de computação distribuída.

Os SGBDROs proporcionam acesso baseado em conteúdo, como por exemplo, o acesso direto baseado em identificadores únicos. A facilidade de uso e a independência dos dados, que são marca registrada de um SGBDR, amadureceram para abranger os recursos necessários de performace e controle, como otimizadores de consultas, segurança de dados e *backup*, e recuperação.

Nos últimos anos, novos sistemas de gerenciamento de banco de dados têm surgido para substituir os SGBDRs. Os fabricantes alegam que os sistemas de banco de dados relacionais têm se tornado deficientes no poder de representação dos dados, comparados com as aplicações que necessitam cada vez mais representar dados complexos. Entre estes novos sistemas, surgiram os SGBDROs, com um modelo de dados relacional estendido, visando ser mais representativo em semânticas e construções de modelagens.

A maior parte dos Sistemas Gerenciadores de Bancos de Dados (SGBDs), utilizados nos últimos anos, é baseada no modelo relacional. Há atualmente, uma grande demanda de migração de sistemas relacionais para sistemas que suportam, de forma mais direta, a representação de objetos complexos, tais como SGBDOOs ou SGBDROs. Esta migração ocorre em função de certas características técnicas, ou mesmo por simples motivos comerciais [Klein, 1999].

Os problemas relacionados à migração podem envolver desde a transposição dos dados até a adaptação dos esquemas e aplicativos. É devido a esta atual demanda de mercado, que várias investigações estão sendo feitas nesta área, buscando soluções aos diversos problemas que surgem no processo de reengenharia de sistemas de banco de dados relacionais.

Como dito no início deste capítulo, os SGBDROs podem ser considerados como relacionais e orientado a objetos, uma vez que suportam sentenças SQL e tipos complexos, respectivamente. Entretanto, não existe um modelo de dados que atenda a todos os SGBDROs disponíveis no mercado. Com a falta de padronização, cada fabricante definiu as características do seu SGBDRO. Uma delas, comum a todos eles, é permitir ao usuário definir novos tipos de dados. Nesse sentido, o SGBDRO da *Oracle* (*Oracle*8) permite definir uma tabela onde cada linha é um objeto, ou ainda, utilizar objetos em colunas da tabela. No entanto, o SGBDRO da IBM (DB2) também permite definir objetos, mas, na atual versão, os objetos estão restritos às linhas das tabelas [Klein, 1999]. No SGBDRO da *Oracle*, o usuário pode definir tipos adicionais de dados, especificando a estrutura e a forma de como operá-lo, pode também usar estes tipos no modelo relacional. Assim, os dados são armazenados em sua forma natural.

Além dos tipos de dados definidos pelo usuário e os tipos base já existentes, o SGBDRO também inclui novos tipos de dados, que permitem as aplicações trabalharem com dados complexos como imagens, som e vídeo.

Existem duas categorias de tipos definidos pelo usuário: tipo objeto *(object type)* e tipo coleção *(collection type)*. Um tipo objeto é uma estrutura de dados definida pelo usuário que serve como base para criar objetos (instâncias). São esquemas de objetos com três classes componentes: nome, atributos e métodos. O tipo coleção é do tipo *array (array type)* ou do tipo tabela *(table type)*, que descreve uma unidade de dado composta de um número indefinido de elementos do mesmo tipo. As unidades de dados correspondentes são chamadas *ARRAYs* e tabelas aninhadas, respectivamente [www.dcc.ufmg.br/pos/html/spg98/anais/alecia/alecia.html - 05/10/2001]. Um *array* é um conjunto ordenado de elementos de dados. Todos os elementos são do mesmo tipo, e cada um tem um índice, que é o número correspondente no *array*. O número de elementos de um *array* é seu tamanho, sendo que este pode ser variável.

Uma tabela aninhada é um conjunto não ordenado de elementos do mesmo tipo. A tabela tem uma única coluna e o tipo desta coluna pode ser um tipo base ou tipo objeto. No último caso, a tabela pode ser vista como uma tabela multicoluna, com uma coluna para cada atributo do tipo objeto. Quando um tipo tabela aparece como o tipo de uma coluna em uma tabela relacional ou como um atributo do tipo objeto da tabela adjacente, o SGBD armazena todos os dados da tabela aninhada em uma única tabela.

5.1 Comparação entre Banco de Dados

Na seção anterior foram comentadas algumas características de Banco de Dados Relacional, Banco de Dados Orientado a Objeto e Banco de Dados Objeto-relacional, esses sistemas de banco de dados estão no mercado e o projetista do banco de dados precisa escolher o tipo de sistema apropriado às necessidades da aplicação em que está trabalhando.

As extensões persistentes às linguagens de programação e os sistemas objeto-relacionais estão direcionados a diferentes mercados. A natureza declaratória e o poder limitado da linguagem SQL fornecem boa proteção de dados contra erros de programação e tornam otimizações de alto nível, como redução de I/O, relativamente fáceis. Os sistemas objeto-relacionais visam tornar a modelagem de dados e as consultas mais fáceis pelo uso de dados complexos, incluindo dados multimídia. Entretanto, uma linguagem declaratória, como a SQL, compromete o desempenho para certos tipos de aplicações que executam, principalmente, na memória principal e que realizam um grande número de acesso ao banco de dados.

As linguagens de programação persistentes destinam-se a aplicações que tenham altas exigências de desempenho. Elas fornecem acesso com baixo *overhead* ao dado persistente e eliminam a necessidade de tradução dos dados se tiverem de ser manipulados usando uma linguagem de programação. Entretanto, elas são mais susceptíveis à corrupção, devido a erros de programação e, normalmente, não têm uma capacidade poderosa de consulta. Aplicações típicas incluem banco de dados CAD. A eficácia dos vários sistemas de banco de dados pode ser comparada resumidamente, conforme tabela abaixo.

BD Relacional	**BD Orientado a Objeto**	**BD Objeto-relacional**
tipos de dados simples;	tipos de dados complexos;	tipos de dados complexos;
linguagens de consulta poderosas;	integração com linguagem de programação;	linguagens de consultas poderosas;
alta proteção.	alto desempenho.	alta proteção.

Tabela 5.1 - Comparação dos vários Sistemas de Banco de Dados.

Essas descrições geralmente se mantêm, mas é importante saber que alguns sistemas de banco de dados obscurecem esses limites. Por exemplo, alguns sistemas de banco de dados orientados a objeto, construídos em torno de uma linguagem de programação persistente, são implementados no topo de um sistema de banco de dados relacional. Tais sistemas podem fornecer desempenho inferior aos sistemas de banco de dados orientados a objeto construídos diretamente sobre um sistema de arquivos, mas fornecem algumas das mais fortes garantias de proteção dos sistemas relacionais.

5.2 Processamento de Consultas

5.2.1 Métodos de Acesso

O índice árvore-B+ é um dos métodos mais eficientes para incrementar o acesso aos tipos de dados convencionais, entretanto é necessário estendê-lo para adequá-lo aos novos tipos de dados. Novos métodos de acesso criados também devem ser assimilados naturalmente, pois seria impraticável a existência de um servidor que já contasse com todas as possibilidades de índices adequados a cada novo tipo de dado criado [Klein, 1999]. Existem três possibilidades para associar novos métodos de acesso: a) integrar métodos de acesso externo ao servidor universal; b) estender os métodos de acesso existentes, adaptando-os aos novos tipos; e c) definir novos métodos de acesso. Observe a figura 5.1.

Escolhendo a possibilidade de integrar métodos de acesso externo ao servidor universal, obtém-se duas vantagens principais: a) para o produtor do banco de dados é a alternativa mais simples, pois não são necessárias mudanças complexas do sistema; e b) para o desenvolver do novo tipo de dado é permitido a criação de uma ótima solução, tornando seu produto competitivo e adequado ao SGBDRO.

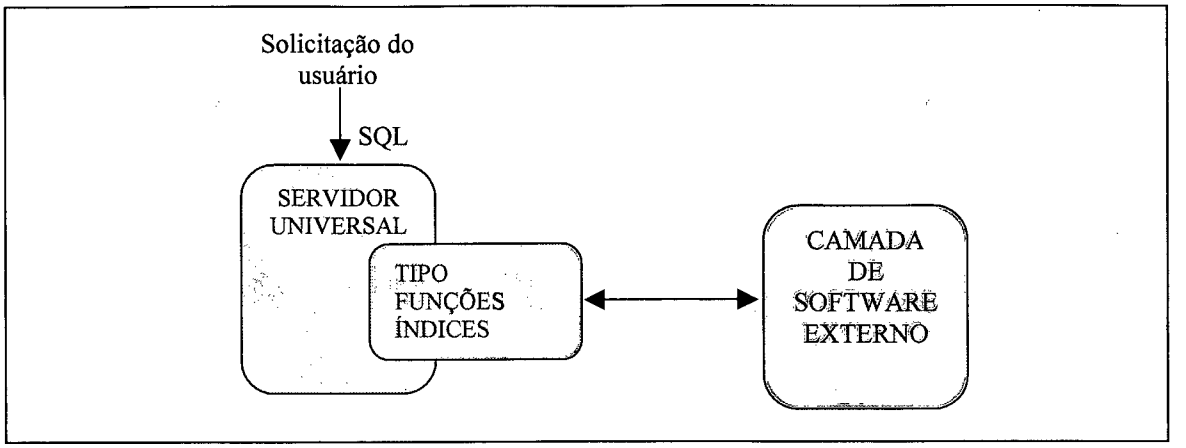

Figura 5.1 – Visão de Integração de *Software* Externo
Fonte: KLEIN, Lawrence Zordam. *A tecnologia Objeto-relacional em um ambiente de data warehouse.* Rio de Janeiro, 1999.

Apesar das facilidades desta opção, é melhor estender os próprios métodos de acesso do servidor ou incorporar novos índices, já que a utilização de índices externos exige cuidados na manutenção da sincronia entre o banco de dados e o índice externo. Também é possível indexar os resultados obtidos, a partir de funções definidas pelo usuário, o que é muito útil quando esses resultados são utilizados como filtros em consultas. O exemplo abaixo mostra a sintaxe de criação de um índice árvore B+ sobre o resultado da função Azul, aplicada sobre a coluna imagem, da tabela Fábrica.

```
TABLE          Fábrica (Id, Preço, Fabricante, Imagem)
CREATE INDEX   Cor_index
ON             Fábrica
USINS B-TREE   (Azul (Imagem));
```

5.2.2 Linguagem de Consultas

A linguagem SQL e as outras linguagens de consultas têm sido estendidas para tratar tipos complexos e orientação a objetos [Silberschatz et al., 1999].

Ao realizar consultas em tipos complexos utiliza-se a extensão da linguagem de consulta SQL. Considere o exemplo: "encontre o nome e o ano de publicação, de cada documento".

```
SELECT    nome, data.ano
FROM      doc
```

Observe que o campo ano do atributo composto data é referido pelo uso de uma notação de ponto. A SQL estendida permite que uma expressão para uma relação apareça em qualquer lugar em que o nome da relação possa aparecer, como em uma condição FROM.

Uma outra situação possível é encontrar todos os documentos que têm a palavra "banco de dados" como uma de suas palavras-chave, para isso, é formulada a seguinte consulta:

```
SELECT    nome
FROM      pdoc
WHERE     "banco de dados" IN lista_palavra_chave
```

Na expressão acima foi utilizado o atributo relação-valorado lista_palavra_chave em uma posição em que a SQL, sem o uso de relações aninhadas, teria exigido uma subexpressão *SELECT-FROM-WHERE.*

Como outro exemplo, considere uma relação que contêm pares da forma "nome_documento e nome_autor" para cada documento, e cada autor do documento.

```
SELECT    B.nome, Y.nome
FROM      pdoc AS B, B.lista_autor AS Y
```

Já que o atributo lista_autor de pdoc é um campo conjunto-valorado, ele pode ser usado em uma condição *FROM*, em que deveria ser usada uma relação.

Em sistemas objeto-relacionais é permitido aos usuários definir funções. Estas funções podem ser definidas em uma linguagem de programação, como a linguagem C ou C++, ou ainda, em uma linguagem de manipulação de dados, como a SQL.

Considere uma função que, dado um documento, retorne a contagem do número de autores. A definição da função é a seguinte:

```
CREATE FUNCTION    total_autor (um_doc Documento)
                   RETURNS INTEGER AS
                   SELECT COUNT (lista_autor)
                   FROM um_doc
```

A função é chamada com um único objeto documento e a declaração *select* é executada com uma relação um_doc contendo somente uma única tupla, isto é, o

argumento da função. O resultado da declaração *select* é um único valor, ou seja, é uma tupla com um único atributo, cujo tipo é convertido em um valor.

A função anterior pode ser usada em uma consulta que retorna os nomes de todos os documentos, que tem mais de um autor:

```
SELECT    nome
FROM      doc
WHERE     total_autor (doc) > 1
```

Observe que, na expressão SQL anterior, embora doc refira-se a uma relação na condição *from*, ela é tratada implicitamente como uma variável de tupla na condição *where* e pode, portanto, ser usada como um argumento para a função contagem_autor.

Normalmente, uma declaração *select* pode retornar um conjunto de valores. Se o retorno da função é um tipo complexo, o resultado da função é todo o conjunto. Se o tipo do retorno não é um tipo conjunto, como é o caso no exemplo anterior, o resultado gerado pela declaração *select* deve conter somente uma tupla, que é retornada como a resposta. Se existe mais de uma tupla no resultado do comando *select*, o sistema tem duas possibilidades: tratar esta situação como um erro ou selecionar uma tupla arbitrariamente e retorná-la como a resposta [Silberschatz et al., 1999].

5.2.3 Otimização

O otimizador do SGBDRO deve escolher de forma eficaz a estratégia de acesso aos dados, avaliando as diferentes opções de acesso. O otimizador precisa estar claramente ciente das estruturas de armazenamento e dos métodos de acesso em efeito.
A figura 5.2 apresenta algumas das possibilidades que devem ser levadas em consideração pelo otimizador.

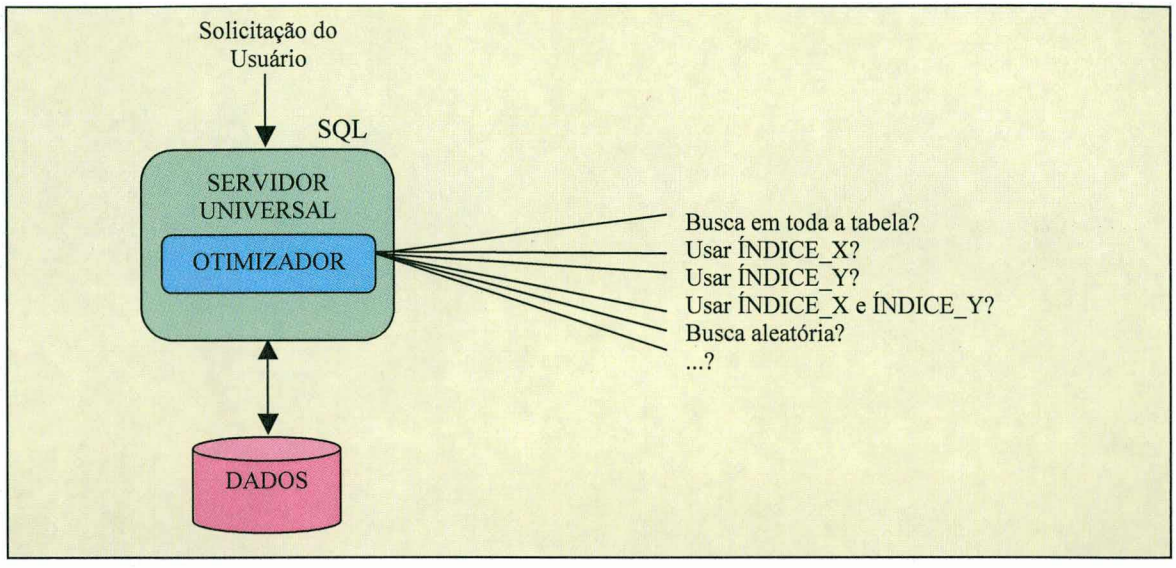

Figura 5.2 – Opções de um Otimizador
Fonte: KLEIN, Lawrence Zordam. *A tecnologia Objeto-relacional em um ambiente de data warehouse.* Rio de Janeiro, 1999.

Ao acessar várias estratégias, o otimizador estima o custo de cada uma das alternativas viáveis, através de informações estatísticas armazenadas no catálogo do sistema. Devido à maior diversidade de tipos de dados, a tarefa de otimizar torna-se mais complexa, no ambiente Objeto-relacional. Para superar essa dificuldade, alguns produtores permitem que o otimizador possa ser educado ou influenciado para trabalhar de forma mais eficiente com novos tipos de dados e funções.

Uma das soluções disponibilizadas pela linguagem *ORACLE* 8 é a inclusão de uma dica, diretamente no comando SQL, que guiará o otimizador na realização dessa consulta. Entretanto, este mecanismo depende da qualidade da dica, pois uma escolha errada pode prejudicar a realização da operação. Uma segunda possibilidade seria permitir ao usuário a influência na estimação de custos entre as várias estratégias de acesso. Novos tipos de dados e índices podem dificultar essa estimação, portanto existe, em alguns casos, um fator de ajuste que facilita esta tarefa. A terceira opção busca o aprendizado contínuo do otimizador a partir dos resultados obtidos por suas escolhas. A quarta e última opção seria a inclusão de informações para o otimizador nas definições das funções criadas pelo usuário [Date, 2000].

Um otimizador de consulta convencional aplica certas leis de transformação para reescrever consultas. Contudo, historicamente, essas leis de transformação foram todas embutidas no código do otimizador. Por contraste, em um sistema Objeto-relacional, o

conhecimento relevante precisa ser mantido no catálogo, implicando em mais extensões. Nesses sistemas objeto-relacionais o próprio otimizador precisa ser reescrito.

Por exemplo, considere uma expressão como *not* (QDE > 500). Um bom otimizador transformaria essa expressão em (QDE ≤ 500), pois a segunda opção pode fazer uso de um índice sobre QDE, enquanto que a primeira não pode. Por razões análogas, é necessário haver um caminho para informar ao otimizador quando um operador definido pelo usuário é a negação de outro.

Um bom otimizador também saberia, por exemplo, que as expressões (QDE > 500) e (500 < QDE) são logicamente equivalentes. Contudo, é preciso haver um meio para informar ao otimizador quando dois operadores definidos pelo usuário são opostos nesse sentido.

O otimizador deverá saber também que, por exemplo, os operadores "+" e "-" se cancelam, isto é, são inversos. É necessário haver um meio para informar ao otimizador quando dois operadores definidos pelo usuário são inversos nesse sentido.

Dada uma expressão *booleana* como (QDE > 500), geralmente, os otimizadores fariam uma suposição sobre a seletividade dessa expressão, isto é, a porcentagem das tuplas que a torna verdadeira. Em tipos de dados e operadores embutidos as informações sobre a seletividade podem ser embutidas no código do otimizador. Em contraste, em tipos e operadores definidos pelo usuário, é necessário haver um meio para fornecer ao otimizador algum código definido pelo usuário a ser invocado, objetivando fazer suposições sobre as seletividades.

O otimizador precisa conhecer o custo de execução de determinado operador definido pelo usuário. Por exemplo, dada uma expressão com p *and* q, onde p corresponde a um operador área de algum polígono complicado e q a uma comparação simples, como (QDE > 500). Nesse exemplo, seria preferível que o sistema executasse q primeiro, de forma que p somente fosse executada sobre tuplas, onde q tivesse valor verdadeiro.

Uma parte da heurística clássica de transformação de expressões, como a de sempre executar restrições antes de junções, não é necessariamente válida para tipos e operadores definidos pelo usuário [Date, 2000]

CAPÍTULO VI

BANCO DE DADOS MÓVEIS

A utilização de Banco de Dados Móveis está baseada na capacidade que os usuários têm, desde que munidos de um dispositivo móvel, de se comunicarem com a parte fixa da rede e possivelmente com outros dispositivos móveis, independentemente da sua localização. Primeiramente este capítulo traz as características e a aplicabilidade de uma tecnologia que surgiu recentemente no mercado, a Computação Móvel. Posteriormente, o capítulo se preocupa em definir Banco de Dados Móveis. Finalmente, na última seção encontra-se a descrição das etapas do processamento de consultas em Banco de Dados Móveis.

6. Computação Móvel

Para comentar Bando de Dados Móveis é importante conhecer as características da Computação Móvel. Novas tendências tecnológicas têm surgido no mercado computacional, o avanço da tecnologia tem possibilitado aos usuários o acesso às informações a partir de diferentes pontos, a qualquer momento, limitado somente pela localização de uma área geograficamente determinada, por isso está havendo uma grande procura pelos computadores pessoais, *notebooks* e *laptops*.

Basicamente a arquitetura da computação móvel é formada por *hosts* móveis, estações fixas e a comunicação entre eles. A figura 6.1 mostra a Arquitetura da Computação Móvel.

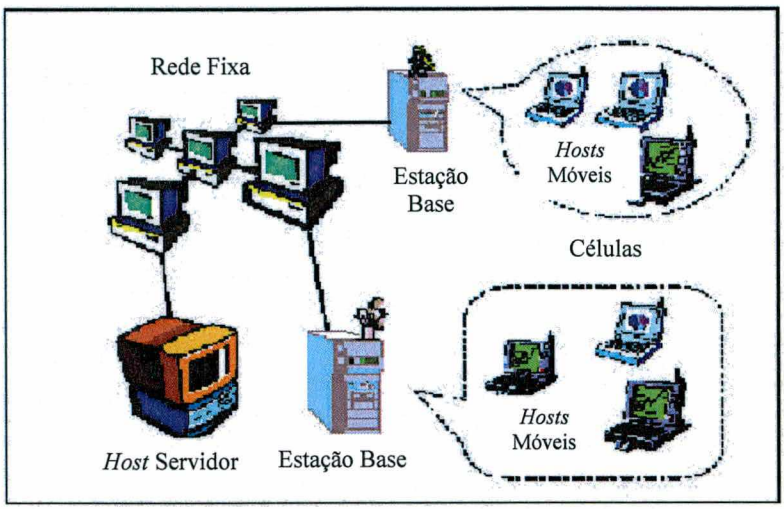

Figura 6.1 – Arquitetura da Computação Móvel
Fonte: SSU, Kuo; YAO, Bin; FUCHS, Kent; NEVES, Nuno Ferreira. *Adaptive Checkpointing with storage management for móbile environments, manuscript.* Dezembro 1998.

A mobilidade é a principal característica da Computação Móvel. Ela aponta limitações e desafios que em ambientes fixos são tratados com facilidade e funcionam sem maiores dificuldades. Existem problemas que já foram resolvidos na computação convencional, mas persistem ou são dificultados em ambientes móveis [http://www.dimap.ufrn.br (abril/2001)].

A comunicação dentro de uma célula é realizada por difusão (*broadcasting*), assim, com uma única transmissão é possível enviar uma mensagem a todos os MH's que estão localizados em uma célula. Surgindo então, alguns problemas: a) como os MH's podem se mover entre as células da rede pode ocorrer, por exemplo, que um determinado MH receba duas vezes a mesma mensagem, fato este que não traria muito prejuízo; b) um MH pode deixar uma célula onde sua MSS correspondente ainda não enviou determinada mensagem global, ocorrendo assim a perda da mesma; e c) um MH pode entrar nas limitações de outra célula vizinha onde a mensagem já tenha sido enviada. Estes problemas podem ser visualizados graficamente na figura 6.2.

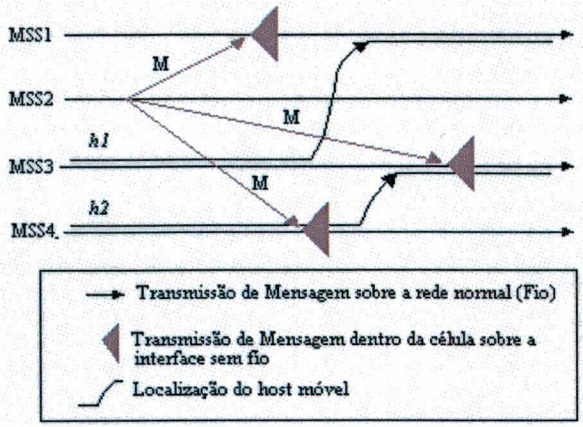

Figura 6.2 – Problema da Comunicação sem fio e Mobilidade
Fonte: BADRINATH, B. R; ACHARYA, A. e IMIELINSKI, T. *Impact of mobility on distributed computations.* Rutgers University – Departament of Computer Science: New Brunswick, NJ, 1993.

6.1 Vantagens e Restrições da Computação Móvel

A computação móvel é indiscutivelmente vantajosa em suas aplicações, pois oferece conforto na utilização de computadores em qualquer ambiente, possibilita a flexibilidade na utilização de diversas aplicações que exijam movimento, tem disponibilidade, independente da localização do usuário, fornece portabilidade que facilita o transporte e possui autonomia de energia, garantindo o funcionamento dos equipamentos.

Apesar das vantagens e praticidade, ainda persistem algumas limitações na Computação Móvel. Muitos parâmetros são fundamentais na hora de projetar algoritmos para ambientes móveis. A computação móvel possui quatro restrições básicas, que decorrem de fatos inerentes à própria mobilidade, e não são dependentes da tecnologia atual. Estas restrições complicam o projeto de sistemas móveis e exigem atenção dos grupos de pesquisa. As restrições são descritas a seguir [Satyanarayanan, 1999]:

a) Os elementos móveis possuem fracos recursos em comparação aos elementos estáticos, pois os elementos móveis dependem de uma tecnologia que diminua o peso, consumo de energia, tamanho e ergonomia dos mesmos.
b) A mobilidade é inerentemente arriscada, pois os elementos móveis (*notebook's*, *laptop's*, etc.) são dispositivos pequenos, que permitem serem transportados com facilidade, por este motivo podem ser roubados, perdidos ou mesmo danificados por

algum acidente. Assim, é preciso que o usuário decida se a mobilidade oferecida por estes dispositivos, compensa o risco que o mesmo terá que correr.

c) A conectividade móvel é altamente variável em relação à performance e confiabilidade. Certas configurações móveis podem oferecer tamanhos de banda altos e confiabilidade, enquanto outras oferecem apenas um tamanho de banda pequeno. Quando o usuário móvel está ao ar livre, ele pode ter necessidade de utilizar redes sem fio com tamanhos de banda (*low-bandwidth*) pequenos, tendo falta de cobertura em certas áreas.

d) Os *hosts* móveis necessitam de fontes de energia finitas. Talvez uma das principais restrições da computação móvel é o fato da tecnologia atual de baterias não permitirem que *hosts* móveis funcionem por mais de 2 ou 3 horas, sem a necessidade do usuário ter que desligar seu elemento móvel para recarregar a bateria ou ter que se conectar a uma tomada de parede, perdendo assim praticamente toda a característica de mobilidade.

6.2 Banco de Dados Móveis

A possibilidade de se utilizar um computador móvel como um servidor de banco de dados, desperta o interesse de muitas áreas de negócio realizado em ambientes externos. Em ambientes móveis, os dados podem residir em redes fixas ou em *hosts* móveis. Banco de Dados Móveis se refere a qualquer arranjo, onde o acesso ao Banco de Dados é realizado por estações móveis através de um *link* sem fio (*wireless link*). Banco de Dados Móveis é um ambiente cujas estações móveis, possuem um maior poder de computação e podem armazenar dados nativos, que necessitam ser compartilhados por outros [Özsu e Valduriez, 1999].

A arquitetura de um banco de dados móveis é formada pelas plataformas de aplicação, por um *host* servidor e pela comunicação realizada entre eles. A plataforma de aplicação de banco de dados móveis consiste em um computador portátil, que possua um banco de dados, com informações armazenadas e algumas camadas de software. É

através do SGBDD, que os usuários podem interagir com a aplicação do banco de dados e acessar informações armazenadas.

O banco de dados servidor pode ser um computador de médio ou grande porte, coordenado por um sistema de gerenciamento. O *host,* que armazena o banco de dados, contém dados dinâmicos, que podem ser usados pelos usuários móveis. A posição do *host* na organização depende do tipo de informações que estão sendo processadas.

O banco de dados móveis troca informações com o computador que contém o banco de dados, de modo a manter o processo de atualização, consultas e etc. A comunicação entre ambos, acontece com interrupções e em intervalos irregulares, por curtos períodos de tempo, sendo que, algumas vezes, não estão conectados em uma mesma rede.

Existe a possibilidade de se fazer consultas sobre servidores de Banco de Dados fixos, a partir de *hosts* móveis.

Os paradigmas de acesso dos dados, o gerenciamento de transações, a replicação dos dados, a recuperação de falhas, a segurança e o processamento de consultas são aspectos importantes no âmbito de Banco de Dados Móveis. A seguir serão comentadas algumas particularidades desses aspectos.

Paradigmas de Acesso aos Dados Móveis

Os computadores portáteis possuem um grande número de limitações. Devido a este fator, buscam-se formas de acesso aos dados que, reduzam o número de acessos aos servidores de arquivos, diminuindo o tráfego de informações na rede de comunicação [Ito, 2001].

No método *caching* a comunicação sem fio utiliza a desconexão por períodos substanciais como meio de economia, onde a largura de banda é limitada e variável. Mesmo estando desconectadas, algumas unidades móveis podem permanecer em operação, permitindo que os dados sejam utilizados, resultando em diminuição de carga tanto no servidor quanto no canal de comunicação. O uso de memória *cache* para consultas e atualizações pode reduzir o número de acessos aos servidores de arquivos, diminuindo o tráfego de informações na rede de

comunicação. Um cliente móvel armazena dados acessados freqüentemente para melhorar o desempenho das consultas durante a desconexão.

O mecanismo de *caching* (armazenamento) em um ambiente móvel tem como objetivo o acesso mais eficiente aos dados. É utilizado a fim de aumentar a probabilidade do dado procurado estar armazenado localmente na unidade móvel, fazendo com que permaneçam em memória os dados que serão possivelmente reutilizados, de modo a reduzir a contenção de dados e o tempo de respostas das consultas e consequentemente, melhorar o desempenho do sistema.

Por outro lado, o mecanismo de *caching* pode ocasionar inconsistência nos dados utilizados pelos usuários, tornando-os desatualizados, pois algumas vezes é praticamente impossível atualizar os mesmos itens em diferentes servidores simultaneamente. Além disso, canais sem fio sofrem de baixa largura de banda e estão vulneráveis a freqüentes desconexões. Neste caso, a consistência dos dados também pode ser afetada, pois, as mensagens enviadas entre servidor e unidade móvel podem ser invalidadas.

O mecanismo de difusão de dados (*broadcasting*) consiste num sistema, onde a mesma mensagem é enviada de uma estação base (MSS) para todas as unidades móveis que estejam em uma célula. Os dados difundidos podem ser recebidos por um grande número de unidades móveis de uma só vez, sem custos adicionais.

Existem duas razões para a utilização da difusão dos dados. Primeiro, a unidade móvel evita o gasto de energia com a transmissão de solicitações de dados. Segundo, os dados difundidos podem ser recebidos por um grande número de unidades móveis de uma só vez, sem custos adicionais [Silberschatz et al., 1999].

- **Gerenciamento de Transações**

Pelo fato de computadores móveis terem uma quantidade não trivial de armazenamento local, o processamento de transações é tão complexo quanto aqueles realizados em sistemas de banco de dados distribuídos tradicionais.

Para garantir a consistência dos dados compartilhados, propensos à falhas e desconexões, usuários de unidades móveis e fixas atualizam e recuperam dados através de transações.

Devido à mobilidade, as transações podem ser executadas de diversas localizações, e conseqüentemente de diferentes servidores. Diferentemente de transações realizadas em uma rede fixa, as transações móveis podem acessar dados que estão em constantes movimentações, sendo então dependentes da localização dos dados.

Embora o ambiente de computação móvel é apenas um sistema distribuído, os aspectos da relação custo/performance do sistema diferem dos tradicionais sistemas distribuídos. A comunicação para um computador móvel custará mais do que entre computadores não móveis. Este custo não é só em termos de custo de comunicação de rede, mas também em termos de duração da bateria. A recepção de mensagens dentro de um ambiente móvel consome menos energia do que a transmissão (difusão - broadcasting) das mesmas, assim, a função de custo de comunicação não é simétrica.

- **Replicação dos Dados**

A replicação é o processo no qual transações executadas em um banco de dados são propagadas assincronamente, para um ou mais banco de dados de forma serial. A serialização significa que as transações, assim como todas as operações, são replicadas na mesma ordem na qual elas foram solicitadas. Caso isto não aconteça, podem ocorrer inconsistências entre a unidade móvel e o servidor. A propagação de forma assíncrona significa uma forma de armazenar e enviar replicação, ou seja, as operações que serão replicadas do servidor podem ser armazenadas em um banco de dados fonte, até que sejam propagadas para as unidades móveis, no momento da sincronização, pois algumas vezes a conexão se torna impossível [DBMS ONLINE, 1997].

Em um ambiente móvel é importante que estações móveis acessem o banco de dados de maneira que a comunicação seja rápida e reduzida. A replicação é necessária para sincronizar os dados e as operações no banco de dados. Usuários móveis necessitam de replicação de dados para armazenar dados utilizados mais freqüentemente durante o período de desconexão fraca ou total. O item é copiado

para a estação móvel, onde a leitura é realizada localmente, sem a necessidade de comunicação da estação móvel com o servidor.

- **Recuperação de Falhas**

O processo de recuperação de falhas é responsável por preservar a consistência do banco de dados após falhas do sistema de transações ou dos meios de comunicação [Ito, 2001]. Um sistema em ambiente móvel está sujeito à falhas tanto de *hardware* como de *software*. Em cada um destes casos, informações que se referem ao sistema de banco de dados podem ser perdidas. Uma parte essencial do sistema de banco de dados é um esquema de recuperação responsável pela detecção de falhas e pela restauração do banco de dados para um estado consistente que existia antes da ocorrência da falha.

Para que o sistema não seja prejudicado devido às falhas em seus componentes, erros devem ser detectados o mais rápido possível, através de um diagnóstico apropriado. Para recuperar os dados, informações relevantes são armazenadas em um local fixo durante o processamento de transações.

- **Segurança**

A segurança está relacionada à necessidade de proteção contra o acesso ou manipulação de informações confidenciais, por elementos não autorizados, e à utilização não autorizada do computador ou de seus dispositivos periféricos [Ito, 2001]. A proposta de um sistema de segurança é restringir o acesso a informações e recursos para apenas alguns usuários autorizados. Para produzir um sistema seguro contra ameaças específicas é necessário classificar as ameaças e os métodos pelo qual cada uma delas pode ser produzida.

Devido à necessidade de mobilidade, ocorreu o crescimento da utilização de computadores portáteis e de outros equipamentos móveis, e consequentemente, a necessidade de proteção dos dados. A segurança em aparelhos móveis é um dos maiores problemas em redes sem fio. Para minimizar o problema de segurança em Banco de Dados Móveis, deve-se fazer uso de técnicas de criptografias.

- **Processamento de Consultas**

Através de redes sem fio e computadores móveis é possível consultar bancos de dados em máquinas remotas. Estes bancos de dados podem conter dados referentes a hotéis, restaurantes, hospitais e etc, existentes em uma localidade, surgindo assim as chamadas consultas dependentes de localização. O processamento de consultas dentro de ambientes móveis é um aspecto que requer ainda muitas pesquisas, pois a mobilidade adiciona novas características que não aparecem em ambientes convencionais. Para que uma consulta seja realizada em banco de dados móveis, é necessário conhecer a localização precisa da unidade móvel. O uso de computadores em navios, aviões, carros, entre outros meios de transporte, dificulta esta localização, devido as constantes alterações de lugares. O aspecto "Processamento de Consultas em Banco de Dados Móveis" será melhor explorado na seção 6.3.

6.2.1 Banco de Dados Móveis e Banco de Dados Distribuídos

Ao observar a literatura sobre Computação Móvel e Banco de Dados Móveis, é possível detectar semelhanças em diversos aspectos com relação a Sistemas Distribuídos e Sistemas Gerenciadores de Banco de Dados Distribuídos.

Em sistema de computação móvel inclui-se uma rede fixa, sendo essa um sistema distribuído. Um sistema de computação móvel pode ser visto como um tipo dinâmico de sistema distribuído onde os *links* entre os nodos na rede mudam dinamicamente. Então, um ambiente de banco de dados móveis pode ser classificado como um Sistema *Multidatabase* Móvel e Heterogêneo.

A diferença mais significante, entre ambientes distribuídos e ambientes móveis, é a transparência de localização em aplicações distribuídas, e o conhecimento de localização em aplicações móveis.

O gerenciamento de dados também é um aspecto diferenciador entre sistemas distribuídos e sistemas móveis. A tabela apresentada logo a seguir indica as principais características tratadas em gerenciamento de dados móveis.

GERENCIAMENTO DE DADOS NA COMPUTAÇÃO MÓVEL	
Aplicações	Pode ser dependente de localização; Necessita se adaptar as mudanças do contexto;
Transações	Novos modelos que capturem mobilidade;
Recuperação (*recovery*)	Particionamento de rede freqüente; *Shutdown* voluntário da unidade móvel não é uma falha de sistema; Mobilidade pode causar mais logs (*logging*); Técnicas para recuperação de desconexão durante o *handoff*;
Replicação	Restrições de consistência diferentes; Novas técnicas para atualização de *cache* devido a freqüência de desconexões;
Processamento de Consultas	Dependente de localização; Diferentes fatores de custo; Respostas de consultas retornadas de diferentes localizações; Necessidade de técnicas adaptativas;
Resolução de Nomes	Nova estratégia global de nomes devido a mobilidade e desconexão;

Tabela 6.1 – Características do Gerenciamento de Dados Móveis
Fonte: DUNHAM, Margaret H. e HELAL, Abdelsalam. *Mobile computing and databases: anything new?*. Department of Computer Sciences and Engineering, Southern Methodist University, Dallas, Texas, 1995. Disponível em < http://www.seas.smu.edu/~mhd/pubs/95/record.ps>. Acesso em 10 Nov 2000.

6.3 Processamento de Consultas em Banco de Dados Móveis

A tecnologia da Computação Móvel e das redes sem fio possibilitou a consulta ao banco de dados de algumas máquinas remotas. O processamento de consultas dentro de ambientes móveis é um aspecto que requer ainda algumas soluções para determinadas restrições.

Os computadores móveis utilizam baterias para realizar suas tarefas, portanto, pela escassez de energia elétrica, começam a surgir limitações que influenciam diretamente no projeto do sistema. Pelo fato dos computadores móveis necessitarem de uma fonte de energia que também seja móvel, as tecnologias de baterias estão requerendo pesquisas com a intenção de melhorar a duração dessas baterias. As tecnologias atuais de baterias representam uma das maiores restrições

da computação móvel. O uso da difusão escalonada de dados é um fator importante que necessita da eficiência energética, para reduzir a necessidade dos sistemas móveis transmitirem consultas [Silberschatz et al., 1999].

Um outro ponto relevante está relacionado às quantidades crescentes de dados que podem ficar em máquinas que são administradas pelos próprios usuários, ao invés de serem monitoradas pelo administrador do banco de dados.

A mobilidade em ambientes móveis também adiciona novas características que não aparecem em ambientes convencionais. O processamento de consulta é uma das funções do SGBD mais afetadas pela mobilidade do ambiente. Os efeitos surgem tanto em termos das consultas propostas quanto das técnicas de otimização que podem ser usadas [Özsu e Valduriez, 1999]. As máquinas móveis não mais possuem endereços fixos de rede, por esse motivo o processamento de consultas é dificultado, visto que a ótima localização para a resposta da consulta depende deste endereço.

O aspecto de segurança também é uma restrição no processamento de consultas em Banco de Dados Móveis. Existe a facilidade de interceptar mensagens na comunicação sem fio, causando sérios problemas de segurança, devendo então fazer uso de técnicas de criptografia para tentar minimizar este problema. Existe também a facilidade de se fazer o rastreamento do computador móvel, a partir do momento que ele se comunica com a rede fixa.

As seções subseqüentes comentam algumas etapas importantes no processamento de consultas em banco de dados móveis.

6.3.1 Localização

Os computadores convencionais interligados em rede possuem um endereço, que determina o roteamento de pacotes a serem entregues a um destinatário. O endereço dos computadores convencionais é estático e determina a localização de um computador em relação ao restante da rede. No entanto, no caso de computadores móveis, isto não é válido, pois o endereço está em constante mudança. Se o endereço associado com o

computador móvel permanecer o mesmo, independente de sua localização, esse endereço não mais poderá ser usado para rotear pacotes, pois não representa a localização correta e atual do computador móvel. Por outro lado, se um computador móvel possuir um endereço que é função de sua posição, então todas as outras entidades (computadores, processos, aplicações e etc.) em contato com esse computador móvel, precisariam ser informadas das alterações do endereço. Portanto, as consulta efetuadas em ambientes móveis, são altamente "dependentes da localização" [Özsu e Valduriez, 1999].

Assim que localizadas as estações é necessário efetuar a alocação de canais. Entre as alternativas adotadas, destaca-se a alocação fixa, onde o mesmo número fixo de canais é alocado a cada estação. Explorando a mobilidade do usuário, a alocação dinâmica procura alocar os canais conforme as demandas em cada área de abrangência de uma estação.

Dentre as características de processamento de consulta nos ambientes móveis, a mais importante deve ser a consulta baseada no contexto de localização, direção e velocidade em que o MH está localizado. Desta maneira, os parâmetros para uma consulta podem expressar para o servidor, a atual localização do computador móvel onde a consulta será realizada. Estes parâmetros também podem depender da direção e velocidade com que o MH está se movimentando.

Os diferentes valores de resultados para uma mesma consulta em localizações diferentes, produzem um novo tipo de replicação de dados, que é baseado na localização dos mesmos, chamado de replicação espacial [Özsu e Valduriez, 1999].

Como dito anteriormente, um computador móvel está constantemente se locomovendo. Ao contrário de banco de dados convencionais, banco de dados móveis tem necessidade de conhecer a localização geográfica dos dados que serão processados.

Os dados dependentes de localização podem mudar durante o processamento da consulta. Estes tipos de consultas também são denominados consultas dependentes de localização. Os dados denominados LDD (*Location Dependent Data*) são dados cujos valores são determinados por sua localização [Dunham e Kumar, 1998]. Por exemplo, dados sobre a associação de estações de TV e redes nacionais irão variar de cidade para cidade. O LDD pode ser usado para responder a consultas dependentes da localização.

Com essa abordagem, a mesma consulta enunciada em diferentes localidades obterá resultados diferentes, porque os próprios valores de dados serão diferentes (replicação espacial). Diferentes réplicas espaciais do mesmo objeto de dados podem ter valores diferentes, porque estão associadas a localizações diferentes. O LDD complica as funções de cachê. Os dados colocados no cachê podem se tornar ultrapassados, não por causa da atualização dos dados no servidor, mas porque a unidade móvel se deslocou para uma nova região, na qual os dados colocados no cachê não são válidos.

As consultas dependentes de localização podem ser processadas aumentando-se cada consulta com informações de localização, supondo-se que as consultas não são modificadas, mas que são usados dados dependentes da localização, ou ainda por uma combinação dessas duas abordagens. Independente da abordagem usada, quando uma consulta é solicitada, ela deve estar vinculada a uma localização e a um conjunto de valores de dados. Como a unidade móvel está se movendo, a consulta poderia estar vinculada a localizações diferentes: a localização da unidade móvel quando a consulta foi solicitada, a localização da unidade móvel quando a consulta termina, uma localização projetada da unidade móvel baseada em seu movimento atual, ou então uma localização indicada de forma específica na consulta.

Considere o exemplo da seguinte consulta: Quais os nomes e endereços dos restaurantes Chineses nesta cidade? Na resposta dessa consulta, serão retornados diferentes valores, pois depende da localização da cidade em que o computador móvel está localizado, isto é, depende da localização de onde foi feita a consulta. Se a mesma consulta fosse realizada várias vezes, a partir de posições (lugares) diferentes, várias respostas diferentes seriam obtidas.

A dependência de localização pode ser sofisticada. Como exemplo, considere um usuário no interior de um carro, realizando a seguinte consulta: Qual o desvio para o Banco *BB* dentro de 1 quilômetro a partir da localização do computador móvel? Na construção da resposta, o sistema tem que observar a trajetória de movimento atual do carro, para que possa descobrir sua localização e responder a consulta. Se o sistema demorar em enviar a resposta ao usuário, ele pode, já que está em movimento, passar pelo Banco sem saber da resposta. Por este motivo, a posição do usuário após alguns momentos é dependente de muitos fatores, como: velocidade, aceleração, tipo de movimento (uniforme, uniformemente variado e etc), direção, sentido, trajetória, tempo

de processamento de consulta, demora na transmissão de dados, desconexões, quedas de sinal e etc [Ito, 2001].

A localização é tratada como fragmento de dados que pode ser consultado e atualizado, onde a informação da mesma é mantida apenas por unidades que estão ativas. Um sistema pode ser fisicamente móvel, embora logicamente estático. Apesar de todos os componentes serem móveis, suas posições são sempre conhecidas.

O gerenciamento da localização das unidades móveis ainda requer algumas pesquisas. O desempenho de várias propostas de gerenciamento de localização dos usuários móveis depende de diversos fatores como mobilidade, número de usuários, características da rede e a topologia do banco de dados utilizado. Em SGBDs tradicionais, apenas as características do nó de processamento são levadas em consideração no gerenciamento de dados, e não sua localização física. Assim, a abordagem tradicional promove a transparência de localização. Contudo, a necessidade de admitir consultas dependentes de localização muda esse requisito na área da computação móvel.

6.3.2 Otimização de Consulta

Independentemente se o ambiente é centralizado, distribuído ou móvel, a otimização de consulta refere-se ao processo de produzir um plano de execução de consulta que representa uma boa estratégia de execução para a mesma.

Uma consulta tem muitas possíveis estratégias de execução e o processo de escolher uma estratégia satisfatória para processá-la é conhecido como otimização de consulta [Elmasri e Navathe, 2000].

Para otimizar uma consulta em banco de dados móveis, diversos fatores são analisados como: custo de comunicação, custo de localização, tempo de resposta e etc.

O custo de comunicação é considerado, quando uma ótima estratégia de processamento de consulta é escolhida. Portanto, a mobilidade desta nova tecnologia resulta em alterações dinâmicas no custo da comunicação e consequentemente dificulta a otimização. Por este motivo, outras noções de custo começam a ser consideradas

como: o tempo do usuário, tempo de conexão, número de bytes ou de pacotes transferidos, tarifas com base em horário e energia limitada. Os otimizadores de consultas no âmbito de mobilidade são obrigados a lidar com aspectos como: a) otimizar a consulta para que o consumo de energia, ao consultar, seja o menor possível; e b) determinar o custo de comunicação entre a estação que está consultando e a estação onde residem os dados.

O custo para gerenciar a localização de um elemento móvel, deve incluir o custo da comunicação. Para minimizar o custo final, algoritmos e estruturas de dados eficientes e planos de execução de consultas, devem ser projetados para realizar a localização dos elementos móveis.

A conectividade entre elementos computacionais não pode ser sempre garantida e quando presente, possui confiabilidade e vazão variáveis. Em ambientes externos (*outdoors*) a velocidade da comunicação, geralmente é mais baixa que em ambientes internos, oferecendo uma conectividade mais confiável ao dispositivo móvel.

Em banco de dados móveis diversos fatores contribuem no tempo de resposta do sistema, sendo eles:

a) o tempo de transmissão de dados pode ser alto, pois a baixa taxa de transmissão e a baixa confiabilidade influencia no tempo;
b) a execução da consulta ao banco de dados pode ser demorada, especialmente em grandes cidades; e
c) o usuário pode estar, voluntariamente ou não, desconectado da rede. Se o tempo de desconexão for longo a resposta pode demorar para o usuário solicitante [Ito, 2001].

Com relação a otimização de consultas, a mobilidade de estações de acesso torna difícil determinar custos de comunicação entre a estação de acesso e a estação em que os dados residem. Além disso, se os dados solicitados estão armazenados em uma das *walkstations*, então ambos os nós de rede podem estar em movimento. A baixa largura de banda entre a parte sem fio e o *backbone* complica ainda mais o quadro.

Em um ambiente de banco de dados centralizado, geralmente a otimização de consultas envolve determinar a melhor abordagem, entre um conjunto de alternativas, para processar uma consulta que minimize o custo de E/S. Em um ambiente distribuído, o custo dominante normalmente é visto como o do tráfego de rede. Embora os custos de E/S e de rede ainda sejam importantes, em um ambiente de computação móvel a

mobilidade e o estado dinâmico da unidade móvel complicam ainda mais a otimização. O conteúdo do cache pode mudar dinamicamente. À medida que a unidade móvel se desloca, os dados podem ou não se tornar disponíveis por meio de um disco de difusão. A melhor localização a partir da qual é possível acessar dados em um servidor de banco de dados na rede fixa pode se alterar conforme a unidade móvel se desloca. Assim, no ambiente de computação móvel, os planos e custos associados com diversos planos mudam com base no movimento. As estratégias estáticas de otimização não são apropriadas. São necessárias estratégias dinâmicas que se adaptem ao ambiente variável.

CAPÍTULO VII

BANCO DE DADOS MULTIMÍDIA

Devido à evolução tecnológica observa-se que inúmeros setores estão exigindo novos tipos de dados, até mesmo a parte tradicional das empresas vem sofisticando e demandando novos serviços, tais como: videoconferências, ilustrações de documentos e apresentações animadas e com som, incluindo todos os recursos gráficos. O Banco de Dados Multimídia vem responder à demanda crescente por novos ambientes interativos que agregam texto, vídeo e áudio, permitindo o manuseio e a apresentação de grandes volumes de dados. O objetivo desse capítulo é apresentar como o processamento de consultas é executado.

7. Definição

Banco de Dados Multimídia é um repositório que, além dos dados convencionais (textos, números e etc), armazena também áudio, imagens, animações e informações em vídeo, que poderão ser manipuladas e recuperadas. Um documento multimídia pode ser apresentado com algumas combinações de texto e vídeo, juntamente com um material áudio [Aragão e Riecken, 1998].

Devido à natureza heterogênea de dados, sistemas multimídias devem suportar armazenamento, transporte, exibição e administração destes tipos de dados. Assim, devem possuir maior capacidade que sistemas convencionais. O Banco de Dados

Multimídia é considerado mais potente, pois mantém e distribui os meios digitais complexos (textos com livres formatos e volume expressivo de informações, gráficos estruturados, imagens estáticas e em movimento, voz e sons).

Permite que qualquer tipo de mídia seja acessado dos computadores de mesa, podendo estar disponível em um servidor de banco de dados central com distribuição em redes compatíveis. Essas informações podem ser desde vídeo conferência, apresentações interativas, treinamento baseado em computador, até dados computacionais animados.

O Banco de Dados Multimídia vem responder à demanda crescente por novos ambientes interativos que agregam texto, vídeo e áudio, permitindo o manuseio e a apresentação de grandes volumes de dados [Aragão e Riecken, 1998].

Os Bancos de Dados Multimídia podem ter as seguintes funções: a) leitura (recuperação e visualização dos dados); b) atualização (inclusão da criação de novos dados multimídia e a modificação dos mesmos); c) composição (criação de composições e apresentações utilizando dados multimídia básicos); d) pesquisa (localização dos dados em um banco de dados multimídia); e) interação (inclusão da interação de um usuário e um Sistema Gerenciador de Banco de Dados com dados multimídia).

7.1 Sistema Gerenciador de Banco de Dados Multimídia (SGBDM)

Um Sistema de Banco de Dados é basicamente um sistema de manutenção dos registros armazenados, visando mantê-los e disponibilizá-los quando necessário. O Sistema Gerenciador de Banco de Dados (SGBD) é um software que se coloca entre o banco de dados físico e as aplicações dos usuários. Assim, todas as solicitações desses usuários são manipuladas pelo SGBD, o qual desobriga as aplicações dessas funções, com simplificação das mesmas e ganho de desempenho. Em outras palavras, o SGBD faz com que os usuários tenham uma visão do banco de dados acima do nível de hardware [Date, 1990].

Os Sistemas Gerenciadores de Bando de Dados Multimídia podem propiciar diferentes níveis de suporte aos dados multimídia. Os diferentes níveis de suporte podem variar desde nenhum suporte até o suporte extensivo (pleno). Serão descritos cinco níveis básicos de suporte que podem ser oferecidos por Sistemas Gerenciadores de Bancos de Dados [Pazandak e Srivatana, 1997].

No primeiro nível o SGBD pode simplesmente agir como um repositório. Nesse nível mais simples de suporte, conterá no banco de dados o nome dos arquivos dos dados multimídia, com o último conteúdo armazenado dentro do sistema de arquivos. O sistema de arquivos pode ser localizado num servidor de dados de meio magnético contínuo especializado e dedicado. A segurança dos dados é deixada para o sistema operacional, pois o SGBD não pode assegurar a integridade dos dados ou validar as referências dos arquivos.

No segundo nível os dados podem ser armazenados dentro do banco de dados, provendo-se o suporte limitado. Os dados são armazenados como BLOB, desta maneira, o SGBD não entende os tipos de dados que estão sendo armazenados. Eles meramente armazenam e recuperam os dados tal como solicitados. Os dados são seguros e podem ser atualizados utilizando-se sistemas de transações, como também podem ser distribuídos para múltiplos usuários. O SGBD suporta o acesso aleatório aos dados, permitindo atualização de trechos do BLOB, mas o usuário deve definir e manter os ponteiros referenciais.

No terceiro nível de suporte os dados são armazenados dentro do banco de dados e o SGBD entende a estrutura dos dados. Nesse nível, o SGBD proverá as definições de classe para os inúmeros padrões de formatos de dados multimídia, por exemplo, JPEG, GIF, MPEG, e outros. Para cada formato, devem ser associados métodos que operaram os dados. O SGBD pode prover, para cada tipo de dados, um suporte para o formato de dados único, não exigindo que todos os dados nesse formato sejam convertidos na importação.

No quarto nível é fornecido um suporte direto para a classe de tipos de dados temporais (ou continuados), incluindo áudio e vídeo. Entretanto, os atuais SGBD comerciais, notadamente com orientação a objeto, não propiciam, na maioria dos casos, esse suporte. Isso pode ser devido à quantidade de espaço em disco requerida para suportar o armazenamento de tais dados.

Finalmente, no quinto nível os tipos de dados multimídia, particularmente os dados temporais, são suportados como tipos de dados básicos dentro do SGBD. Os melhores ganhos de performance podem ser alcançados se os tipos de dados forem suportados pelo sistema operacional. O suporte internalizado para os tipos de dados multimídia permite que o SGBD propicie uma otimização no manuseio dos mesmos.

7.2 Aplicações do Banco de Dados Multimídia

O Banco de Dados Multimídia surgiu em função da necessidade de se armazenar dados não convencionais como: textos com livres formatos, gráficos estruturados, imagens estáticas, imagens em movimento, voz e sons [Aragão e Riecken, 1998].

No passado, os Sistemas Gerenciadores de Bancos de Dados (SGBD) basicamente gerenciavam tipos simples de dados, tais como inteiros e literais. As simples estruturas de registros eram suficientes para representar os dados que seriam armazenados. Os dados mais complexos requeriam definições específicas do usuário quanto aos tipos de dados. Uma das tendências atuais é o uso de SGBD, que façam o gerenciamento de dados multimídia, pois os softwares, as redes e os computadores estão melhores capacitados para realizar o manuseio dos dados de áudio e vídeo [Pazandak e Srivatana, 1997].

Observa-se atualmente, a existência de inúmeros setores que necessitam de novos tipos de dados, até mesmo a parte tradicional das empresas vêm sofisticando e demandando novos serviços, tais como: videoconferências, apresentações animadas e com som, e ilustrações de documentos, incluindo todos os recursos gráficos.

A necessidade de armazenar diferentes mídias é somada a outras demandas, como: a) efetuar pesquisas no conteúdo dos dados contidos na mídia; b) obter facilidade de sincronismo nas trilhas dos textos, sons e vídeos; e c) mesclar os dados de forma transparente para o usuário.

A multimídia fornece benefícios para muitas aplicações de negócios, pois integram voz, som, imagens, animação e vídeo. Com esta nova tecnologia, será possível converter toda informação que esta armazenada em papel, vídeo e filme, dentro de um

ambiente, permitindo aos usuários organizar, pesquisar e rotular objetos multimídias. Estes documentos multimídia podem ser armazenados em diferentes servidores de arquivos, inclusive geograficamente dispersos.

O Banco de Dados Multimídia pode ser útil em muitos campos, por exemplo, na medicina o computador pode auxiliar projetos e sistemas de recuperação de informação. Está ocorrendo na saúde, um aumento significativo no uso de dados baseados em imagens, radiografias e outros. Esses sistemas permitem que os médicos incluam mídias relacionadas aos dados dos pacientes, dentro de uma comunicação interativa. Além disso, os dados em meio magnético podem ser analisados, posteriormente armazenados ou então direcionados para um especialista da área de saúde apropriada.

Um exemplo interativo de Banco de Dados Multimídia seria um livro texto com recursos multimídia, podendo aceitar constantes interações e produzir muitas combinações de mídia. Utilizando o livro texto com recursos multimídia, os estudantes poderiam fazer uma leitura seqüencial, ou ainda, poderiam acessar os hipertextos complementares e seus *links*. Poderiam também especificar o nível acadêmico ou o nível de profundidade dos dados apresentados no livro. O livro texto poderia ser usado como livro de referência para acessar diretamente temas ou áreas específicas, através de *hiperlinks* na tabela de assuntos.

As experiências do uso da multimídia no ambiente escolar demonstra que a informática não significa somente uma renovação técnico-didática, como foram os episcópios, os projetores de slides, os retroprojetores e os vídeos. Mais uma vez o conceito básico e, ao mesmo tempo, o grande diferencial, chama-se interatividade. A informática significa uma verdadeira revolução em todos os ramos do conhecimento e notadamente no ensino, isso porque ela pode garantir, o que nem mesmo um professor particular consegue, que é a interação em quantidade e qualidade das perguntas e respostas no interior de uma situação de ensino.

As aplicações multimídia utilizam os serviços do sistema multimídia para prover funcionalidades adicionais aos usuários ou para outras aplicações. A seguir serão listadas outras aplicações do Banco de Dados Multimídia: aplicações interativas com videodisco; jogos eletrônicos; navegadores hipermídia; sistemas de apresentação; correio eletrônico multimídia; sistemas de vídeo *desktop*; sistema de conferência *desktop*; sistemas de autoria: captura, conversão, composição, adição de interatividade;

e serviços multimídia / TV interativa: telecompras, telebanco, serviços educacionais e vídeo sob-demanda.

Os Bancos de Dados Multimídia estão evoluindo rapidamente, esse crescimento conjugado ao aparecimento dos dispositivos de armazenamento removíveis, como o *DVD* (*Digital Video Disk*), o qual permite armazenar dezenas de *gigabytes*, fomentará uma explosão de aplicações multimídia. Essa explosão, em troca, vai retroalimentar uma necessidade intensa por bancos de dados multimídia ainda mais potentes.

O mundo da Multimídia, do CD-ROM, do *DVD*, das grandes redes *LAN's*, da Internet, etc., será o principal facilitador dos vários ramos de atividade, permeando empresas, governo, escolas e cidadãos [Bairon, 1995].

7.3 Aspectos do Banco de Dados Multimídia

Para que um Banco de Dados Multimídia atinja os seus objetivos, ele deve atender certos requisitos, com uma interação entre os seus componentes, propiciando desta maneira os serviços esperados. Os requisitos a serem atendidos são os seguintes: proporcionar pelo menos as mesmas facilidades do banco de dados tradicional; possuir grande capacidade de armazenamento; possibilitar a recuperação das informações; permitir a integração de mídias, composição e capacidade de apresentação adequada dos dados; dar suporte às consultas multimídia; oferecer interface multimídia e interatividade; e desempenho (*performance*).

A figura 7.1 ilustra a arquitetura de alto nível para um SGBD Multimídia, que endereça alguns dos requisitos mencionados anteriormente. Essa configuração inclui a maioria dos módulos gerenciadores associados aos SGBD tradicionais. Adicionalmente, contem alguns módulos que são requeridos, especificamente, para os gerenciadores de dados multimídia, tais como: integrador de mídia e gerente de objetos. Entretanto, a maior parte dos módulos adicionais, relativamente aos SGBD tradicionais, é externos ao núcleo do SGBD Multimídia, isso inclui: os módulos de apresentação, interface e gerentes de configuração. A configuração também inclui a base de contexto e o gerente

de informações semânticas, os quais fazem parte do módulo de desempenho *(performance)* [Adjeroh e Nwosu, 1997].

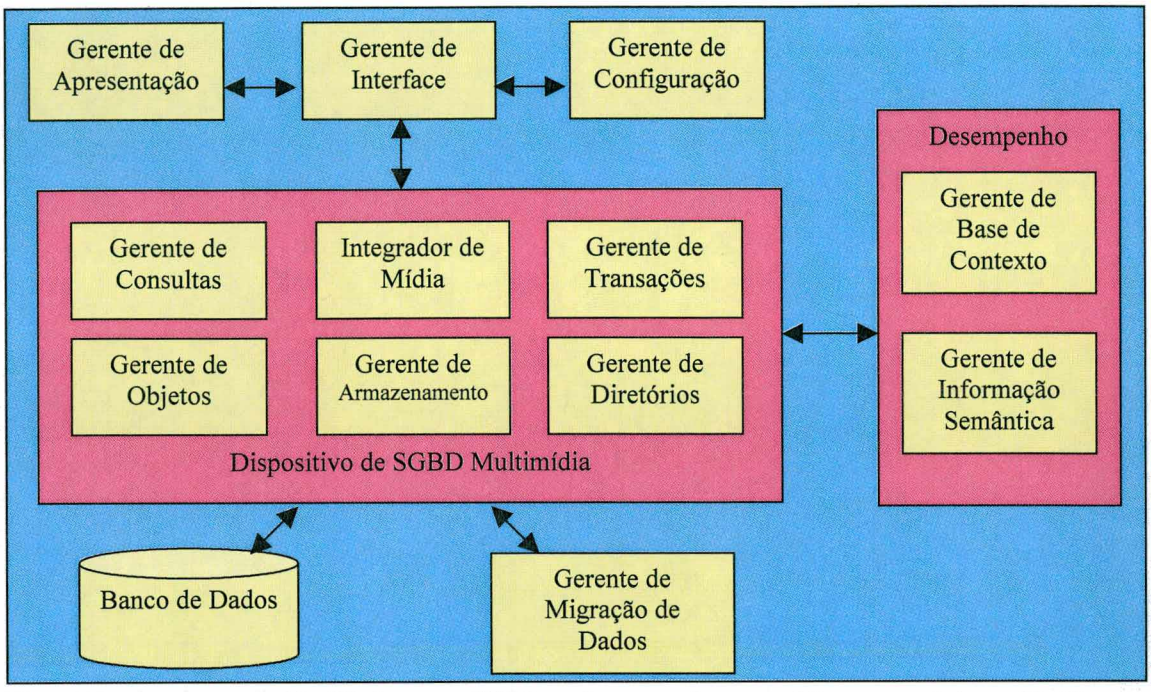

Figura 7.1 – Arquitetura de Alto Nível para um SGBD que busca atender os requisitos para os dados multimídia.
Fonte: ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

Tamanho dos Dados Multimídia

O tamanho dos dados multimídia é substancial, mesmo comprimidos os formatos de vídeo (padrão cinematográfico), em duas horas de filme, resultam em muitos *gigabytes*. Esse fato tem efeito substancial no projeto de *hardware* e de *software* desenvolvido para manipulação de dados multimídia. Devido ao significativo espaço ocupado em disco, a produção de dados relativa a dois ou três filmes pode, por exemplo, ocupar inúmeros *gigabytes* em um único disco rígido. Isso significa que o repositório de dados multimídia terá de incluir uma significativa quantidade de *arrays* de disco, tais como uma composição de discos e coletânea de CDs, como auxiliares no armazenamento terciário. Devido aos requerimentos da taxa de transferência no envio de dados de vídeos (*throughput*), os dispositivos de armazenamento secundário de dados precisarão ser velozes suficientemente para manusear múltiplas solicitações. Portanto, são requeridos sistemas de discos que operem em paralelo.

Ao contrário dos dados tipo numérico ou caracter, os dados multimídia requerem significativa capacidade de armazenamento e aumento da velocidade de processamento. Essas exigências têm prejudicado a popularidade de aplicações multimídia nos computadores pessoais, mas com certeza, não por muito tempo. A recente geração da tecnologia de CD e do *DVD* (*Digital Video Disk*) já está disponível no mercado. A tecnologia *DVD* habilita o desenvolvimento de aplicações poderosas com o uso de multimídia interativa nos PCs, provendo acesso para uma enorme quantidade de armazenamento removível e barato. Esse avanço rapidamente estimulará a indústria de aplicações multimídia interativas para o popular PC [David, 1997].

Armazenamento dos Dados Multimídia

Os objetos multimídia possuem atributos descritivos que indicam onde foi criado, quem foi o criador e qual categoria pertencem. Para construir um banco de dados para objetos multimídia é necessário utilizar bancos de dados que armazenam os dados descritivos e mantêm os registros dos arquivos que armazenam os objetos multimídia.

Normalmente, quando o número de objetos multimídia é relativamente pequeno, os recursos fornecidos por um banco de dados não são importantes. Por este motivo, os dados multimídia podem ser armazenados fora do banco de dados, isto é, podem ser armazenados em sistemas de arquivos. A funcionalidade do banco de dados torna-se importante quando o número de objetos multimídia armazenados cresce e torna-se relativamente grande. Portanto, as atualizações transacionais, as facilidades de consultas e de indexação tornam-se relevantes [Silberschatz et al., 1999].

O armazenamento dos dados multimídia fora do banco de dados dificulta as funcionalidades comuns do banco de dados, como por exemplo, a indexação com base no conteúdo real dos dados multimídia. Diante disto, é preferível que os dados sejam armazenados no próprio banco de dados.

Existem vários aspectos importantes que devem ser considerados quando os dados multimídia são armazenados em um banco de dados, como:

a) O banco de dados deve dar suporte a grandes objetos, pois os dados multimídia podem ocupar alguns *gigabytes*. Muitos sistemas relacionais não dão suporte a

objetos tão grandes, embora sistemas de armazenamento de objetos, como *Exodus*, forneçam suporte a grandes objetos;

b) O resgate de alguns tipos de dados, como áudio e vídeo, exige reentrada de dados a uma taxa constante. Esses dados são chamados de dados de mídia contínuos. Por exemplo, se os dados de áudio não forem recuperados no tempo certo, haverá lacunas no som. Se os dados forem fornecidos muito rapidamente, o sistema de *buffers* pode ficar sobrecarregado, resultando na perda dos dados. Os tipos mais importantes de dados de mídia contínuos são dados de áudio e de vídeo. Os sistemas de mídia contínuos são caracterizados por seus grandes volumes e pela necessidade de entrega da informação em tempo real.

c) A recuperação com base em similaridade é necessária em muitas aplicações de banco de dados multimídia. Por exemplo, um banco de dados que armazena imagens de impressões digitais. Ao realizar uma consulta de determinada impressão digital, devem ser recuperadas todas as imagens presentes no banco de dados que são similares à impressão digital pesquisada. As estruturas de índices como árvores-B^+ e árvores-R, não podem ser usadas para esse fim. É necessário criar estruturas de índices especiais.

O armazenamento dos dados no sistema multimídia pode ser caracterizado pela gigantesca capacidade de manuseio dos dados e pela organização baseada em sistema de armazenamento hierárquico (piramidal). A figura 7.2 mostra essa organização.

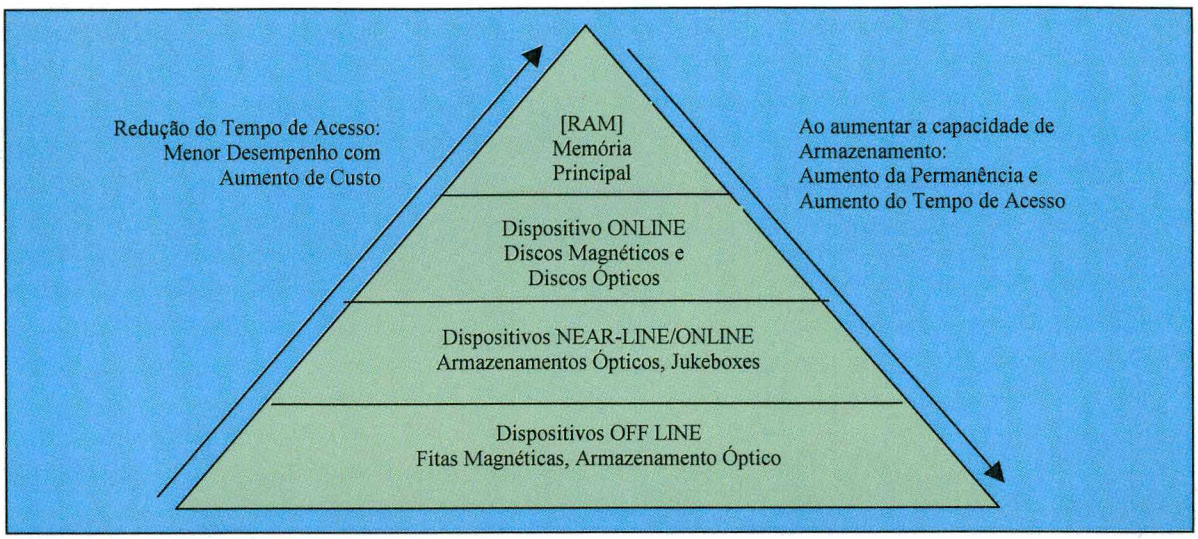

Figura 7.2 – Armazenamento Organizado Hierarquicamente para Banco de Dados Multimídia.
Fonte: ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

O armazenamento hierárquico coloca os objetos de dados multimídia numa hierarquia de dispositivos (*online*, *near-line*, *offline*). Geralmente, o nível mais alto da pirâmide proporciona a melhor performance, embora com maiores custos, menor capacidade de armazenamento e menor permanência. Entretanto, o aumento da permanência dos dados significativo custo adicional.

Outra técnica usada na organização de armazenamento no modelo hierárquico seria a utilização do nível mais alto da hierarquia para armazenar menores abstrações ou representações dos dados multimídia atuais, os quais podem ser usados para facilitar a localização (*browsing*) e a visão previa do conteúdo do banco de dados.

O custo e o desempenho, em termos de tempo de acesso, decrescem na medida em que desce a hierarquia da pirâmide, enquanto que a capacidade de armazenamento e a permanência crescem. Tipicamente, na maioria dos sistemas de armazenamento multimídia, os níveis mais altos de armazenamento estão na memória de acesso randômico volátil, seguido dos *drives* de disco magnético. Esses provêm serviços *online*.

Os dispositivos de armazenamento óticos propiciam o próximo nível de armazenamento. Em alguns casos o acesso é *online*, em outros é *near-line*, tais como nos *jukeboxes* (dispositivos que manuseiam pilhas de CDs).

O nível mais baixo na hierarquia do armazenamento representa os dispositivos de armazenamento *offline*, incluindo-se fitas magnéticas, discos óticos e outros. Esses

dispositivos podem estar ou não conectados ao computador. Oferecem uma alta capacidade de armazenamento e desempenho, mas incorrem no pior desempenho em termos de tempo de acesso.

O Sistema Gerenciador de Banco de Dados Multimídia deve, entretanto, gerenciar e organizar os dados multimídia armazenados em qualquer nível da hierarquia. Deve ainda haver mecanismos para migrar automaticamente objetos de dados multimídia de um nível para outro da hierarquia [Adjeroh e Nwosu, 1997].

Tipos de Repositórios

No início desse capítulo foi descrito que a multimídia engloba muitos tipos de mídia, incluindo-se imagens, vídeo, áudio, gráficos, animação, hipertexto e hipermídia. Também pode incluir dados MRI e outros tipos de dados abstratos (ADTs). Muitos sistemas de banco de dados padrão, como a linguagem SQL, suportam o armazenamento de ADTs em grandes objetos em formato binário, ou BLOBs, a versão do padrão ANSI SQL3 suporta o armazenamento de ADTs como objetos [David, 1997].

Um cabeçalho (*header*) no início do arquivo e de cada registro informa as características dos dados armazenados. Para que a imagem seja gravada é necessária a reserva de uma área física, designando-se o endereço inicial e final dos dados gravados. A imagem gravada na forma de metadado pode ser recuperada através de índices nos quais os atributos mencionados são passíveis de filtro. Um exemplo seria permitir a recuperação de imagens de alta definição de cores. A imagem também poderia, ser recuperada segundo o assunto, ou uma identificação da mesma, por exemplo: exibir a foto tirada em Ubatuba, por ocasião das bodas de prata de um cliente. Pode-se também fazer a busca por palavras-chave, por exemplo, as palavras: crianças brincando, parque, etc. Uma vez construída a indexação, pode ser possível a recuperação de um conjunto de imagens que satisfaçam determinada condição. Como exemplo considere a consulta: exiba todas as imagens com a figura de um chapéu.

Há duas características de aplicações multimídia em termos de armazenamento dos dados: os repositórios de dados e o gerenciamento inteligente dos dados [Pazandak e Srivatana, 1997]. Não há necessidade de entender os formatos dos dados que estão sendo armazenados, nem como os repositórios de dados os recuperam fisicamente. O

propósito do repositório é prover um suporte simples de gerenciamento de banco de dados, nos aspectos de segurança e salvamento dos dados (*backup*).

Desde que os objetos são armazenados como BLOBs eles existem como um objeto simples singular. As pesquisas podem ser formuladas envolvendo metadados, ou outros dados no repositório, mas dificilmente a pesquisa pode ser feita contra o dado multimídia. Adicionalmente, qualquer restrição temporal inerente aos dados, tal como vídeo não é entendido pelos repositórios. O dado é simplesmente enviado aos clientes, os quais têm aplicações que o manuseiam apropriadamente. A seguir, são oferecidos alguns exemplos de repositórios [Pazandak e Srivatana, 1997].

- O pseudo-repositório contém metadados multimídia, tais como: nomes, comprimentos, decodificação ou codificação utilizada, descrições e palavras-chave de dados de vídeo. Os valores substitutivos armazenados dentro do banco de dados descrevem o nome da rota (diretório e subdiretórios) dos objetos multimídia, os quais estão armazenados na forma de arquivo simples dentro de arquivo local, ou arquivo acessado via rede local. O SGBD tem controle limitado desses tipos de arquivos uma vez que esses objetos residem fora do repositório.
- O repositório simples pode prover acesso restrito aos dados e também uma localização centralizada de quais dados podem ser restaurados. Algumas aplicações podem simplesmente desejar que uma central gerencie o armazenamento, ou mesmo a extração de dados multimídia. Os dados são gerenciados por um SGBD e podem ser armazenados em discos locais ou em dispositivos de armazenamento terciários, tais como discos óticos. As aplicações extraem os dados multimídia, utiliza-os localmente e posteriormente efetuam a atualização no repositório.
- O correio eletrônico pode incluir o envio de dados multimídia. O sistema de correio pode usar um repositório para armazenar dados, ou os dados podem simplesmente originar-se de um repositório. Em ambos os casos, o repositório age como um servidor, meramente enviando as mensagens de correio para o cliente, quando solicitadas. Para ler uma mensagem oriunda do correio, o cliente deve ter uma aplicação que entenda o formato do dado multimídia.
- Projetos de Engenharia. Para o propósito de segurança e talvez do gerenciamento da configuração, os desenhos de engenharia e os modelos sólidos podem ser armazenados dentro de um repositório. Qualquer operação, tal como modificação, é

executada no cliente, utilizando-se de *softwares* que façam a interpretação dos dados.

- Sistemas de Informações de Saúde. Para fim de arquivamento, os dados dos pacientes, tais como radiografias e anotações médicas podem ser armazenadas em repositórios.

Persistência dos Dados

A orientação a objeto viabilizou novos tipos de dados, encapsulando os métodos e o valor dos dados (seu conteúdo). As aplicações do banco de dados multimídia manipulam esses novos dados (objetos). Um objeto poderia ser, por exemplo, um carro, já os métodos seriam, acelerar e frear. Todos os atributos de valor desse carro, como cores, ano de fabricação e especificações formariam dados que, junto com os métodos, integrariam o objeto carro. Um banco de dados multimídia que utilizasse o objeto carro e dele, por herança, gerasse um objeto-carro-novo, precisaria garantir que o objeto primordial carro sempre existiu. Essa seria uma forma simples de compreender o problema da persistência dos dados, ou armazenamento físico dos objetos multimídia, seja centralizado ou distribuído.

Assim, muitas aplicações precisam de objetos persistentes. Esses objetos têm de ser alcançados por diferentes usuários e devem continuar existindo até mesmo se o computador que os contêm for removido ou desconectado da rede [Elmer,1998].

Mapeamento de Imagens e Vídeo

Uma imagem imóvel pode ser considerada um quadro estático. Uma única imagem imóvel pode requerer grande espaço de armazenamento. Por exemplo, a imagem de uma latinha de refrigerante colorida pode consumir tanto espaço de armazenamento, quanto um livro com mais de um milhão de bytes. É por esse motivo, que o armazenamento das informações de cores, relativas a uma imagem, inclui na verdade, a combinação de três cores básicas, e triplica as exigências de armazenamento, comparativamente a uma imagem em preto e branco.

O vídeo inclui uma sucessão de imagens, chamadas estruturas ou armações (*frames*), que provêem a ilusão de movimento. A compressão de vídeo e os armazenamentos padronizados reduzem a memória em pelo menos 40:1. Os dispositivos *DVD* são capazes de armazenar um filme de cinema típico por completo. Esses dispositivos oferecem taxas de compressão mais altas do que os padrões anteriores (MPEG-1 e MPEG-2), também produzem melhor qualidade de demanda de vídeo (*video on demand*).

O áudio, assim como as músicas e a voz humana, inclui freqüência e amplitude. O áudio pode ser comprimido e usualmente intercalado (comprimido e/ou armazenado em conjunto) com o vídeo com o qual se relaciona. O padrão MPEG suporta essa junção e armazena múltiplas trilhas de vídeo e áudio, tais como diferentes idiomas ou ângulos de visão da câmera da mesma película, os quais podem ser selecionados durante a exibição (*playback*). Essas intercalações de formatos de arquivos também são críticas com relação à sincronização dos vários fluxos de dados a ela relacionados [David, 1997].

- **Independência dos Dados**

A primeira vista, a importância da independência dos dados para a multimídia não é óbvia. Mas, o fato dos Sistemas Gerenciadores de Banco de Dados Multimídia assegurarem a independência dos dados entre a lógica das aplicações e seu armazenamento físico, representa benefícios significativos. Esse processo otimiza o armazenamento, a busca e a recuperação, porque assim o banco de dados multimídia tem conhecimento da estrutura dos dados e da localização do seu armazenamento.

- **Operações Assíncronas**

O apoio às solicitações e operações assíncronas é uma importante característica operacional dos bancos de dados multimídia. Esse suporte é necessário porque as aplicações multimídia normalmente têm muitas operações que ocorrem simultaneamente. Por exemplo, para exibir um filme de longa-metragem, tal operação pode requerer uma largura considerável da banda (*bandwidth*) de passagem de sorte,

fazendo jus ao enorme fluxo de *megabytes* por segundo com transporte de dados contínuos.

- **Integridade dos Componentes**

Outra tarefa útil para os bancos de dados multimídia é prover dispositivos que garantam a integridade entre os componentes. Por exemplo, a relação entre a imagem mídia composta de um grupo de pessoas e o áudio associado a cada pessoa. Se uma nova imagem é adicionada, na qual as pessoas estão em diferentes localizações, o áudio descrevendo as primeiras imagens não pode ser associado com o quadro mais novo. Em outras palavras, os materiais relativos à mídia mais recente nem sempre são os que devam ser usados.

7.4 Processamento de Consultas em Banco de Dados Multimídia

Consultar dados multimídia pode apresentar algumas possibilidades fascinantes como encontrar uma combinação particular de palavras em uma coleção de pequenos trechos demonstrativos de áudio (*clips*), ou localizar uma pessoa que executa uma certa seqüência de operações numa coleção de *clips* de vídeos. Essas operações de pesquisa recaem sob o título das análises dos dados abstratos e podem incluir operações de procura em dados do tipo MRI, para localizar automaticamente, por exemplo, tumores ou outras patologias.

A análise de dados abstratos requer grande quantidade de recursos computacionais. Por essa razão, a busca multimídia e os dispositivos de recuperação de dados confiam nas anotações do autor, colocadas no banco de dados para evitar ou limitar as análises em tempo real. Essa filtragem pode se basear nas anotações que contêm as especificações de datas, localizações, ou quaisquer outras anotações que descrevam ou documentam porções dos dados abstratos.

Pesquisas de dados multimídia e dados abstratos são temas desafiadores, mas os bancos de dados multimídia tornam isso possível dada a sua flexibilidade de formato de

armazenamento e eficiência de operação. O SGBD deve ter conhecimento significativo sobre os dados e sua estrutura, para possibilitar otimizações semânticas poderosas e pesquisas inteligentes. As operações de pesquisa dão à aplicação acesso aos componentes de mídia de forma que eles possam ser processados dinamicamente e por inteiro, quando necessário.

O Processamento de Consulta é uma atividade comum nos diversos bancos de dados. As pesquisas que envolvem dados típicos (inteiros e *strings*) são geralmente imediatas. Entretanto, os dados multimídia ao realizar uma pesquisa, requerem a interpretação de seu conteúdo. Podendo, ao gerar o resumo dessa pesquisa, resultar em sofisticados esquemas de indexação e análise de algoritmos de imagem e áudio.

É permitido aos usuários realizar pesquisas por imagens ou por ações específicas, tal como correr. Sendo assim, é necessária a existência de mecanismos que geram índices, interfaces e linguagens, cujas pesquisas são apresentadas e os componentes apontados, propiciando-se a otimização das mesmas.

Se os dados multimídia estiverem distribuídos através de rede, deve-se dar uma maior importância à verificação da taxa de rendimento e a confiabilidade do transporte desses dados. Os fluxos de dados utilizados por protocolos e *hardwares* não são passíveis de transporte adequado com alta qualidade, pois os usuários, de um modo geral, toleram a baixa qualidade dos quadros, acima da velocidade de degradação, por exemplo, uma tremulação na imagem.

Um usuário pode requisitar múltiplos trechos de áudio e vídeo extraídos, ao mesmo tempo, do disco. Além disso, inúmeros usuários podem estar requisitando simultaneamente diferentes dados do mesmo disco. Os dispositivos usados para reproduzir dados multimídia, tais como monitores e alto-falantes, bem como os dispositivos usados para gravar dados multimídia devem ser escalonados sem conflitos.

A próxima seção descreve como uma consulta de dados multimídia pode ser realizada.

7.4.1 Consultas a Dados Multimídia

A reaquisição de informações é comum não apenas em sistemas de informação que armazenam textos, mas também em sistemas que armazenam outros tipos de dados, como dados multimídia. O processo de reaquisição de informações consiste em localizar os documentos relevantes, baseando-se em informações fornecidas pelo usuário, como palavras-chave ou exemplos de documentos (similaridade). [Silberschatz et al., 1999]. Por exemplo, um filme pode ter palavras-chave associadas a ele, entre as quais seu título, diretor, atores, classificação e assim por diante.

Alguns sistemas permitem a reaquisição baseada em similaridade. Neste caso, o usuário fornece ao sistema um documento x e solicita a reaquisição de documentos similares a x. A similaridade entre dois documentos pode ser definida, por exemplo, baseada em palavras-chave comuns. Se o conjunto de documentos similares x for grande, o sistema pode apresentar ao usuário alguns poucos documentos, permitindo que ele escolha os mais relevantes e comece uma nova procura baseada na similaridade entre x e tais documentos selecionados.

Os sistemas de reaquisição de informações baseados em palavras-chave, geralmente, permitem consultas que usem expressões compostas a partir de palavras-chave. Um usuário pode solicitar todos os documentos que contenham as palavras-chave "carro e manutenção". Por exemplo, considere o problema de localizar entradas em um catálogo sobre livros de manutenção de carros usando a palavra-chave "carro" e "manutenção". Suponha que as palavras-chave para cada entrada no catálogo sejam as palavras no título e os nomes dos autores. A entrada de catálogo de um livro chamado *Conserto de Carro* não seriam recuperada, já que a palavra "manutenção" não ocorre em seu título. Para solucionar este problema faz-se o uso de sinônimos. Cada palavra pode ter um conjunto de sinônimos definidos e a ocorrência de uma palavra pode ser trocada pelo ou de todos os seus sinônimos, incluindo a própria palavra. Assim, a consulta "carro" e "conserto" seria trocada por "carro *e* (consertou *ou* manutenção)". Essa consulta encontraria a entrada desejada no catálogo.

Outro ponto importante em reaquisição de informações é o grau de relevância de um documento. Suponha que, adicionalmente ao título e autores, uma lista de tópicos coberta pelo livro seja criada e utilizada como parte das palavras-chave no catálogo. Pode haver livros sobre corridas de automóveis que possuam seções sobre carros e motocicletas, assim como seções sobre manutenção. As palavras-chave "carro" e

"manutenção" estariam associadas às entradas no catálogo, e esses livros mais os livros sobre manutenção de carros seriam recuperados em resposta à consulta sobre manutenção de carros. Mas, esses outros livros apresentam apenas relevância marginal à consulta. Um livro no qual as palavras-chave "carro" e "manutenção" ocorram próximas uma da outra é provavelmente mais relevante do que outro em que elas apareçam separadas. Alguns sistemas de reaquisição de informações permitem que as consultas especifiquem se as duas palavras devem ocorrer uma após a outra no texto, ou dentro de uma mesma sentença. Geralmente, os documentos cujas palavras-chave especificadas ocorrem próximas umas das outra são, provavelmente, mais relevantes do que outros em que elas aparecem separadas. Pode ser atribuído um número de relevância aos documentos que contenham todas as palavras-chave especificadas e considere a proximidade dessas palavras-chave, e os documentos podem ser classificados por ordem de relevância quando apresentados ao usuário. Se a lista é comprida, o sistema tem a liberdade de apresentar apenas os documentos mais relevantes.

A linguagem de pesquisa para banco de dados multimídia opera navegando uma estrutura de dados hierárquica definida pelo autor. É possível alcançar essa tarefa, se o autor especificar a estrutura que melhor se adapta às aplicações e aos dados [Aragão e Riecken, 1998]. Como exemplo, considere a tabela Listando1, a seguir. Essa tabela demonstra como a estrutura de dados (figura 7.3) pode ser definida para um Banco de Dados Multimídia. A estrutura de dados mostrada na figura 7.3 é satisfatória para uma aplicação simples, e demonstra um livro texto multimídia interativo, organizado por capítulos e seções, contendo hipertextos, vídeos, vozes e imagens.

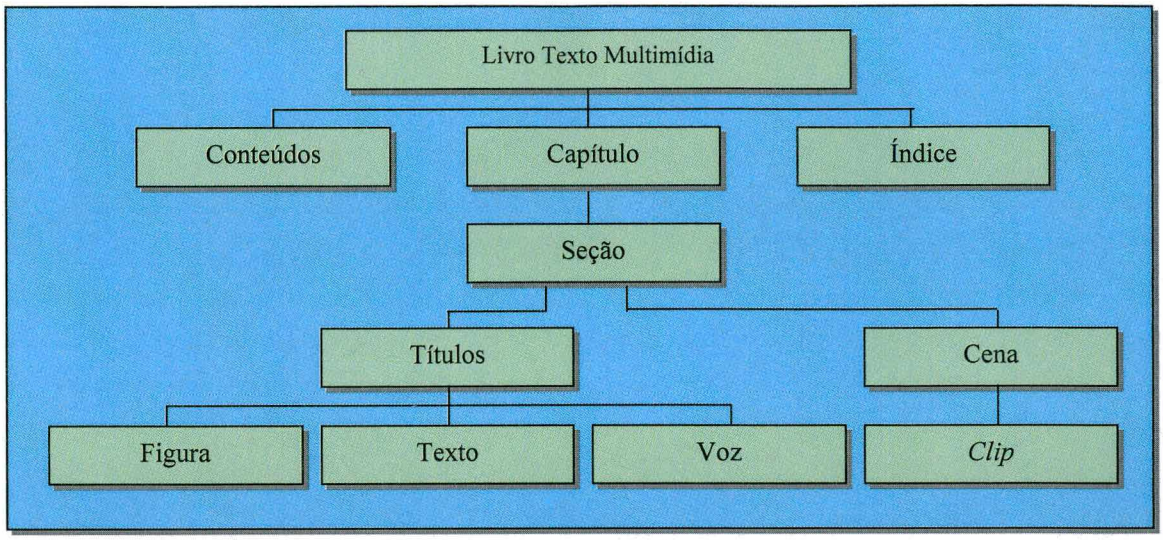

Figura 7.3 – Estrutura de Dados - Livro Multimídia (Metáfora com a Realidade Operacional).
Fonte: ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

LISTANDO 1		
Define	*Structure MMBooks, Table Dir 1*	
Directory	*Book*	*Parent Book, Type Hypertext*
ADT	*Contents,*	*Parent Book, Table Dir2,*
Directory	*Chapter,*	*Field (Overview, Character)*
Directory	*Sections,*	*Parent Chapter*
Directory	*Heading,*	*Parent Section*
ADT	*Text,*	*Parent Heading, Type Hypertext*
ADT	*Voice,*	*Parent Heading, Type Audio*
ADT	*Picture,*	*Parent Heading, Type Image*
Directory	*Scene,*	*Parent Section, Index SceneX*
ADT	*Clip,*	*Parent Scene, Type Video*
ADT	*Index,*	*Parent Book, Type Hypertext;*

Tabela 7.1 – LISTANDO 1
Fonte: ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

Na tabela Listando 1, a estrutura definida de palavras-chave nomeia uma estrutura de dados que inclui uma hierarquia de diretórios. As palavras-chave diretório identificam o diretório de "seus pais", e a palavra-chave *ADT* identifica um segmento *ADT*, armazenado em algum lugar do banco de dados relacional *BLOB*. A estrutura de dados dos metadados também é armazenada no banco de dados relacional.

Para auxiliar a otimização da regra da organização hierárquica, as tabelas relacionais, as quais armazenam as tabelas de diretórios, podem ser especificadas em separado pela tabela de palavras-chaves, limitando a pesquisa por área (observe a figura 7.3). Se não especificada, assumirá a especificação da tabela de diretório constante do parâmetro *Define*. Usando o índice de palavras-chave, pode-se especificar um índice para as anotações usadas, acelerando as buscas.

O campo palavra-chave pode ser usado com diretórios para incluir campos de dados especificados pelo autor. Os campos do autor são especificados no parâmetro nome do campo (*field name*) e combinações de tipo (*type combinations*). Se eles forem especificados, a entrada de diretório deve especificar uma tabela que se utiliza outra tabela contendo as palavras-chave, caso contrário, a tabela de diretório será de tamanho diferente e maquiarão o parâmetro que deveria ser assumido (*default*).

A figura 7.4 ilustra as tabelas de relacionamento e as relações necessárias para suportar a estrutura de diretórios hierárquicos que mantém a estrutura de dados especificada pelo autor, tal como em Listando 1. Cada diretório contém as entradas das tabelas de diretório de mais baixo nível, as quais dispõem do nome de diretório (Capítulo), nome da entrada de diretório (Capítulo Cinco), chaves para subdiretórios (Seção), entradas *ADT* (*Clip*), valores seqüenciais para ordenar as entradas de diretório (Capítulo Três antes de Capítulo Quatro), critério de pesquisa das anotações (*GradeLevel* = 12), e quaisquer campos especificados pelo autor.

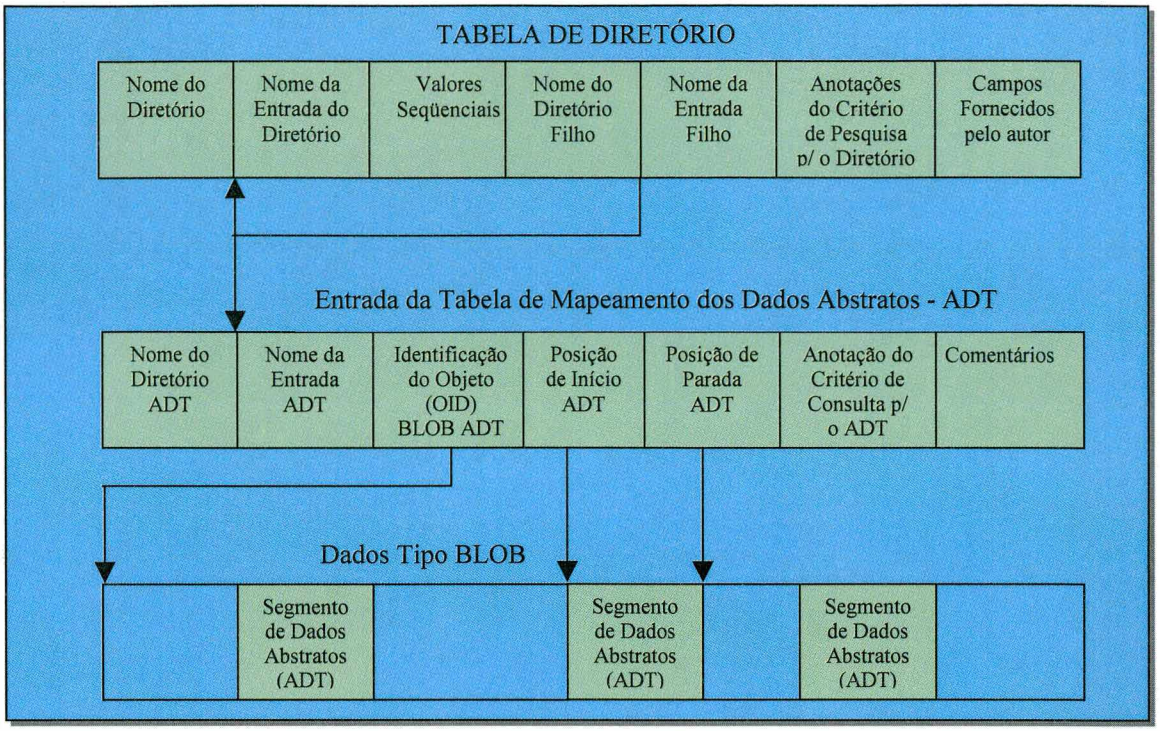

Figura 7.4 – Estrutura Interna de um Banco de Dados Multimídia.
Fonte: ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

As entradas de tabelas *ADT* identificam uma área *BLOB* contendo um segmento *ADT*. No caso de um *clip* de vídeo, esse segmento é uma série de estruturas (*frames*) de filme. Por motivo dos segmentos *ADT* não serem armazenados diretamente na estrutura do diretório, eles são referenciados por uma Identificação de Objeto (*OID*) e estão livremente disponíveis para referenciar sem duplicação de dados. Qualquer número de *BLOBs* pode ser usado em um banco de dados multimídia, por exemplo, um livro pode incluir segmentos *ADT* armazenados em um ou mais *BLOBs*. E em virtude dos *BLOBs* serem armazenados fora da estrutura de diretório, podem ser livremente acessados e atualizados por aplicações externas e por outros diretórios.

As linguagens de pesquisa a dados multimídia suportam uma combinação de navegação no banco de dados automática e procedural, com filtragem e controle da pesquisa. A seguir são apresentados alguns exemplos de pseudolinguagens de pesquisa a dados multimídia, utilizando a estrutura de dados do livro multimídia e a metáfora definida na figura 7.3.

Open Database "MMBook"; {Abra o banco de dados de nome "*MMBook*"}
Play VideoDevice, Book="Jaws", {Exiba no vídeo o livro denominado "*Mandíbula*"}
Scene("Rating=PG13"), {Cena ("Taxa = PG13 ")}
Clip;
Play VideoDevice, Book(Type=Adventure "), {Exiba no Vídeo os Livros de Aventuras}
Clip;
Where Like("Shark"); {Onde contiver ("Tubarão ")}

Essas operações são transmitidas para o banco de dados multimídia. Como se poderia esperar, a declaração *Open* (abra) abre e especifica o banco de dados ativo "*MMBook*" (livro-texto multimídia) que contém muitos livros. Os nomes de diretório, especificados na definição da estrutura de dados da figura 7.3, são integrantes da linguagem de pesquisa. Por conseguinte, a metáfora do autor especificada de um livro multimídia é íntegra tanto na operação quanto na especificação.

As operações de exibição contêm informações de navegação para localizar e devolver o segmento de *ADT* desejado para processamento. A navegação é indicada pela especificação dos nomes de diretório abaixo da rota da estrutura hierárquica, para os dados desejados da estrutura definida *ADT*.

São possíveis três níveis de filtragem: seleção, anotações e *UDFs*. Com a opção de seleção, pode-se especificar explicitamente quais as entradas do diretório, caso contrário, todas as entradas de diretório possíveis para o diretório especificado são incluídas. Essa seleção é especificada pela posição do sinal de igual ("=") depois do nome do diretório e selecionando a(s) entrada(s) de diretório pertinente(s). Também pode ser utilizado o recurso das anotações como um filtro, incluindo-as como critério de pesquisa entre parênteses, seguido das especificações de diretório às quais elas pertencem. Seleção e anotações são possíveis no mesmo diretório, se nenhum for necessário, a especificação de nome de diretório pode ser eliminada da navegação. O terceiro tipo de filtragem (*UDFs*) é efetuado após as aplicações dos recursos de seleção e anotações.

No exemplo anterior, a função *Clip* é usada para identificar o *ADT* na estrutura de dados acessada. Os diretórios Capítulo e Seção não são especificados, pois essa especificação é desnecessária. A navegação automática preencherá os vagos.

A segunda operação de exibição não especifica um nome de livro, mas invoca um filtro para pesquisas nas anotações de um filme do tipo "Aventura", significando que somente os filmes classificados como de "Aventura" serão selecionados. Logo, à função *Clip* indica qual *ADT* a ser acessado. A operação Onde (*where)*, especificada como exibir, analisa o segmento de vídeo *ADT*, para localizar as estruturas de vídeo que podem conter as imagens de "tubarões". A especificação similar à operação Onde é um *UDF* do tipo ANSI SQL3, que analisa o vídeo para certos tipos de figuras, no caso os "tubarões". Esta análise somente acontece nos segmentos de *ADT*, que tenham sido qualificados depois que os filtros de seleção e anotações tenham sido aplicados.

7.4.2 Indexação de Documentos

Uma estrutura de índices efetiva é importante para o processamento eficiente de consultas em sistemas de reaquisição de informações. Esses sistemas podem ter de localizar os documentos que contenham uma palavra-chave específica usando um índice invertido, que mapeie cada palavra-chave K_i no conjunto S_i de identificadores de documentos que contêm K_i. Para possibilitar a classificação de relevância baseada na proximidade de palavras-chave, esse índice pode fornecer não apenas identificadores de documentos, mas também uma lista de pontos no documento em que a palavra-chave aparece. Como os índices devem ser armazenados em discos, a organização do índice tentará minimizar o número de operações de I/O para recuperar um conjunto de documentos que contenham uma palavra-chave. Assim, o sistema tentará manter um conjunto de documentos com uma palavra-chave em páginas de disco consecutivas.

Cada palavra-chave pode estar contida em um grande número de documentos, portanto, para manter baixa ocupação de espaço pelo índice, a representação compacta é um fator crítico. Assim, os conjuntos de documentos para uma palavra-chave são mantidos compactados. Dessa forma, economiza-se espaço de armazenamento e o índice é, às vezes, armazenado de tal forma que a reaquisição não é completa. Alguns poucos documentos relevantes podem não ser recuperados (falso abandono) ou alguns poucos documentos irrelevantes podem ser recuperados (falso positivo). Uma boa

estrutura de índice não tem nenhum falso abandono, mas pode permitir alguns falsos positivos. O sistema poderá filtra-los posteriormente verificando as palavras-chave que de fato contêm.

Duas métricas são utilizadas para indicar quão bem um sistema de reaquisição de informações é capaz de responder a consultas. A primeira, precisão, mede qual porcentagem de documentos recuperados é realmente relevante para a consulta. A segunda, lembrança, mede qual a porcentagem de documentos relevantes para a consulta foi recuperada.

A operação e encontra documentos que contenham todo o conjunto de palavras-chave K_1, K_2, ...,K_n especificadas. O primeiro passo na implementação da operação e é recuperar todos os conjuntos S_1, S_2,..., S_n de identificadores de documentos de todos os documentos que contenham as respectivas palavras-chave. A interseção, $S_1 \cap S_2 \cap ... \cap S_n$, dos conjuntos fornece os identificadores dos documentos do conjunto de documentos desejados. A operação ou fornece o conjunto de todos os documentos que contenham pelo menos uma das palavras-chave K_1, K_2,..., K_n. A implementação da operação ou é obtida por meio da união, $S_1 \cup S_2 \cup ... \cup S_n$, dos conjuntos. A operação não mais encontra os documentos que não contenham uma palavra-chave K_i especificada. Dado um conjunto S de identificadores de documentos, é possível eliminar os documentos que contenham uma palavra-chave K_i especificada tomando a diferença $S - S_i$, em que S_i é o conjunto de identificadores de documentos que contêm a palavra-chave K_i.

Um índice texto-completo de um documento usa cada palavra do documento como uma palavra-chave. Esse tipo de índice pode ser usado em documentos curtos. Entretanto, quase todos os documentos de texto contêm palavras como "e", "ou", "um" e outras. Essas palavras são inúteis como índices. Os sistemas de reaquisição de informações definem um conjunto de palavras, chamado palavras excluídas, contendo cem ou mais palavras entre as mais comuns e removem esse conjunto dos documentos indexados. Eles também não permitem utilizar essas palavras em consultas.

CAPÍTULO VIII

ANÁLISE COMPARATIVA

8.1 Análise Comparativa do Trabalho

Cada aplicação de banco de dados abordada neste trabalho busca atender uma necessidade de mercado. Devido às particularidades das aplicações de banco de dados, conclui-se que o processamento de consultas possui aspectos relevantes que são tratados diferencialmente em cada aplicação.

O processamento de consultas em Banco de Dados Centralizado é relativamente simples, embora exija um conhecimento completo do banco de dados, já o processamento de consultas distribuído requer informações somente locais. Portanto, num ambiente distribuído é necessário selecionar os melhores locais para processar os dados.

O processamento de consultas em Banco de Dados Distribuído é bastante semelhante ao processamento de consultas em Banco de Dados Centralizado, entretanto possui maior complexidade, pois são implementados através de fragmentos que afetam diretamente o desempenho das consultas. A álgebra relacional, num ambiente distribuído, não é suficiente para expressar estratégias de execução, por isso é preciso complementar com operações de troca de dados.

Em Banco de Dados Distribuídos, para que todos os usuários tenham acesso às informações simultaneamente, é necessário o uso de replicações. O sistema mantém várias cópias idênticas de uma relação original, onde cada cópia é armazenada em

diferentes *hosts*. O uso de replicações traz algumas vantagens, como por exemplo, a disponibilidade (caso ocorra falha em um dos *hosts* onde a relação está armazenada, esta pode ser encontrada em um outro *host*, podendo desta forma continuar o processamento das consultas envolvendo a mesma, apesar da falha). Outra vantagem é com relação ao aumento do paralelismo (a replicação dos dados provê maior paralelismo para o sistema, quando se refere a acessos mais direcionados à leitura da relação, permitindo o processamento de consulta simultânea de vários *hosts* a uma mesma relação, minimizando o movimento dos dados entre os nós).

O processamento de consultas em Banco de Dados Distribuído, ao contrário de Banco de Dados Centralizado, considera a localização como uma de suas etapas, onde a distribuição dos dados é justamente tratada nesta fase. A junção ordenada é um aspecto importante na otimização de consulta centralizada e num contexto distribuído é mais importante que junção entre fragmentos, pois pode aumentar o tempo de comunicação. Existem algumas aproximações básicas para junção ordenada em fragmentos de consultas. Tenta-se otimizar a junção ordenada diretamente, considerando que a junção seja substituída por combinações de semijunções para minimizar os custos de comunicação. O custo de comunicação é o fator considerado como mais importante em Banco de Dados Distribuído. Na otimização das consultas distribuídas, o custo de comunicação domina o custo de processamento local (I/O e custo de CPU). Então, a otimização de consulta distribuída visa reduzir o problema, minimizando o custo de comunicação geralmente gasto no processamento local. A vantagem é que a otimização local pode ser feita usando os métodos conhecidos em sistemas centralizados.

Em Banco de Dados Orientado a Objeto o processamento de consulta utiliza as mesmas técnicas desenvolvidas para sistemas relacionais, embora, com algumas dificuldades. A definição das regras de transformação e a manipulação das expressões de consulta são semelhantes às de sistemas relacionais, com uma diferença, as expressões de consultas relacionais são definidas sobre relações planas, enquanto que as consultas de objetos são definidas sobre classes, coleções ou conjuntos de objetos, que têm relacionamentos de subtipificação e composição entre eles.

A otimização do acesso a objetos por suas hierarquias de herança é um problema que distingue o processamento de consultas orientadas a objetos do processamento de consultas relacionais. O otimizador de consultas em sistemas relacionais considera o

número de itens de dados (cardinalidade), o tamanho de cada item de dados e sua organização, entretanto, essas informações não estão disponíveis nos SGBDOOs. A definição de funções de custo, especialmente nas abordagens baseadas em objetos que revelam seus custos, deve ser pesquisada com maior profundidade antes de fazer conclusões satisfatórias. A geração do plano de execução para uma expressão de consulta se preocupa basicamente com a escolha e a implementação dos algoritmos mais eficientes para executar os operadores individuais da álgebra e suas combinações. Em SGBDOOs a questão é mais complicada, devido à diferença nos níveis de abstração de objetos definidos de forma comportamental e seu armazenamento.

Os bancos de dados relacionais usam uma arquitetura tabular ou matricial onde os dados são referenciados através de linhas e colunas, enquanto os bancos de dados orientados a objetos podem ser inteligentes combinando lógica e os dados. Nos relacionais as tabelas são definidas tendo como base a teoria da normalização evitando a redundância dos dados e facilitando a pesquisa e atualizações. Os orientados a objetos possuem métodos, classes e outros mecanismos do modelo de orientação por objetos.

Os objetos são ativos, uma vez que podem conter lógicas, já os relacionais são passivos necessitando de um programa para manipular os dados. O mercado é dominado pelos relacionais, posição reforçada pelo fracasso dos bancos orientados a objetos no passado. Entretanto, com as novas aplicações multimídia e da Internet com o uso da linguagem Java os bancos de dados orientados a objetos estão se posicionando como uma boa opção tecnológica.

Os dados dos bancos relacionais são descritos em 2-D, através de linhas e colunas. A linguagem SQL é um padrão aberto para consulta e manipulação dos bancos de dados relacionais de todos os fornecedores. A SQL permite que os sistemas relacionais desenvolvidos por muitos fornecedores possam se comunicar entre si e acessar banco de dados comuns. Em contra partida, os bancos de dados orientados a objetos não possuem uma linguagem padrão dificultando a interoperacionalidade entre os bancos de dados de diferentes fornecedores.

Para definir as tabelas dos bancos relacionais é utilizado o processo de normalização, que consiste em definir nas tabelas apenas os dados que sejam únicos para a entidade descrita e definindo relacionamentos para outras tabelas também normalizadas. Os bancos relacionais estão fundamentados em uma forte teoria

matemática e ferramentas bem desenvolvidas. Enquanto os bancos orientados a objetos não possuem uma forte teoria como apoio e não existem ferramentas que descrevam o modelo de objetos.

Com o crescimento do mercado de multimídia, vídeo games e aplicações Web que utilizam a linguagem orientada a objetos Java o uso de bancos orientados a objetos está crescendo. Para atender essa demanda os fornecedores de bancos de dados relacionais estão introduzindo facilidades de armazenamento de objetos em seus bancos de dados, chamando-os de banco de dados relacional por objetos. Porém, esse tipo de banco de dados não é possível otimizar as consultas a objetos, fazer indexação direta dos objetos e usar os algoritmos de pesquisa do banco para buscar novos objetos. Somente um banco de dados orientado a objetos puro pode manipular objetos.

Os bancos de dados tradicionais foram desenvolvidos para empresas relativamente estáveis com grandes volumes de dados de baixa complexidade. Em um ambiente dinâmico com dados complexos na forma de vídeo, áudio, imagens e textos é necessário um novo modelo de banco de dados. A vantagem do banco orientado a objetos é a lógica contida no objeto e a possibilidade de ser reutilizado várias vezes em diversas aplicações.

Em Banco de Dados Objeto-relacional a linguagem SQL e as outras linguagens de consultas têm sido estendidas para tratar tipos complexos e orientação a objeto. Devido à maior diversidade de tipos de dados, a tarefa de otimizar torna-se mais complexa, neste ambiente. Para superar essa dificuldade, alguns produtores permitem que o otimizador possa ser educado ou influenciado para trabalhar de forma mais eficiente com novos tipos de dados e funções. Uma das soluções disponibilizadas pela linguagem *ORACLE* 8 é a inclusão de uma dica, diretamente no comando SQL, que guiará o otimizador na realização dessa consulta. Uma segunda possibilidade seria permitir ao usuário a influência na estimação de custos entre as várias estratégias de acesso.

Um otimizador de consulta convencional aplica certas leis de transformação para reescrever consultas. Contudo, historicamente, essas leis de transformação foram todas embutidas no código do otimizador. Por contraste, em um sistema objeto-relacional, o conhecimento relevante precisa ser mantido no catálogo, implicando em mais extensões. Nesses sistemas o próprio otimizador precisa ser reescrito.

Em Banco de Dados Móveis a mobilidade aponta limitações e desafios que em ambientes fixos são tratados com facilidade e funcionam sem maiores dificuldades. As dificuldades que a mobilidade impõe vão desde a velocidade do canal, passando por interferências do ambiente e localização da estação móvel, até a duração da bateria desta estação.

O processamento de consultas dentro de ambientes móveis é um aspecto que requer ainda muitas pesquisas, pois a mobilidade adiciona novas características que não aparecem em ambientes convencionais. Os computadores móveis devem operar mesmo desconectados da rede, o que não acontece em sistemas de banco de dados distribuídos. Embora o ambiente de computação móvel seja apenas um sistema distribuído, os aspectos da relação custo/performance do sistema diferem dos tradicionais sistemas distribuídos. A comunicação para um computador móvel custará mais do que entre computadores não móveis. Este custo não é só em termos de custo de comunicação de rede, mas também em termos de duração da bateria.

A diferença mais significante, entre ambientes distribuídos e ambientes móveis, é a transparência de localização em aplicações distribuídas e o conhecimento de localização em aplicações móveis. As máquinas móveis não mais possuem endereços fixos de rede, por esse motivo o processamento de consultas é dificultado, visto que a ótima localização para a resposta da consulta depende deste endereço.

O aspecto de segurança também é uma restrição no processamento de consultas em Banco de Dados Móveis. Existe a facilidade de interceptar mensagens na comunicação sem fio, causando sérios problemas de segurança, devendo então fazer uso de técnicas de criptografia para tentar minimizar este problema.

Para otimizar uma consulta diversos fatores são analisados, como: custo de comunicação, custo de localização, tempo de resposta e etc. Portanto, a mobilidade desta nova tecnologia resulta em alterações dinâmicas no custo da comunicação e conseqüentemente dificulta a otimização. Por este motivo, outras noções de custo começam a ser consideradas como: o tempo do usuário, tempo de conexão, número de bytes ou de pacotes transferidos, tarifas com base em horário e energia limitada.

Os sistemas multimídias devem possuir maior capacidade que sistemas convencionais. Ao contrário dos dados tipo numérico ou caracter, os dados multimídia requerem significativa capacidade de armazenamento e aumento da velocidade de

processamento. Os dados multimídia, ao realizar uma pesquisa, requerem a interpretação de seu conteúdo. Podendo, ao gerar o resumo dessa pesquisa, resultar em sofisticados esquemas de indexação e análise de algoritmos de imagem e áudio. As consultas em banco de dados multimídia usam expressões compostas a partir de palavras-chave.

O apoio às solicitações e operações assíncronas é uma importante característica operacional dos bancos de dados multimídia. Esse suporte é necessário porque as aplicações multimídia normalmente têm muitas operações que ocorrem simultaneamente.

CAPÍTULO IX

CONCLUSÃO

9.1 Resumo do Trabalho

Este trabalho aborda o processamento de consultas em Banco de Dados Centralizado, Banco de Dados Distribuído, Banco de Dados Orientado a Objeto, Banco de Dados Objeto-relacional, Banco de Dados Móveis e Banco de Dados Multimídia. O processamento de consultas tem como função extrair informações do banco de dados.

Em Banco de Dados Centralizado (BDC), as consultas submetidas pelos usuários são descritas em linguagem não-procedural, isto é, os procedimentos que são utilizados para realizar a consulta não são especificados, embora seja indicado o que os dados da consulta devem satisfazer. Portanto, a consulta deve ser traduzida e transformada em uma seqüência de operações (álgebra relacional), para que o sistema possa compreender e executar essa consulta. No entanto, a tradução da linguagem não-procedural (cálculo relacional) em linguagem procedural (álgebra relacional) permite que o sistema obtenha uma seqüência de comandos otimizados que possibilitam o processamento da consulta de uma forma eficiente.

Em Banco de Dados Distribuído (BDD) os dados são armazenados em diversos lugares. O processamento de consulta em BDD tem uma complexidade maior que em BDC, devido ao grande número de parâmetros a serem considerados. Resumidamente, o processamento de consulta distribuída funciona da seguinte forma: inicialmente a consulta expressa em cálculo relacional deve ser decomposta em uma seqüência de

operações relacionais, chamada de consulta algébrica, logo depois o dado acessado deve ser localizado, assim que as operações sobre relações são traduzidas, conduzindo-o num determinado local (fragmentos). A consulta algébrica, a partir de fragmentos, deve ser estendida com operações de comunicação e otimizada com referência a uma função de custo para ser minimizada. Esta função minimizada refere-se ao cálculo de recursos como disco de I/O, CPU e redes de comunicação.

O Banco de Dados Orientado a Objeto (BDOO) surgiu para atender a necessidade de trabalhar com dados complexos. Esse banco de dados corresponde à integração da tecnologia da orientação a objetos com aptidões de banco de dados. No processamento de consultas em BDOO, primeiramente, a expressão de cálculo é reduzida a uma forma normalizada, eliminando assim os predicados duplicados. A expressão normalizada é convertida a uma expressão algébrica de objeto equivalente. Essa expressão algébrica é validada, garantindo que os predicados e métodos não sejam aplicados a objetos que não suportam as funções requeridas. Posteriormente, aplicam-se as regras de reescrita com preservação da equivalência, buscando sempre garantir a consistência de tipos em expressões algébricas. O último passo é a execução do plano, gerado a partir da expressão algébrica otimizada.

O Banco de Dados Objeto-relacional (BDOR) tem capacidade de manipular dados complexos aliados à flexibilidade de execução de consultas típicas de sistemas relacionais. Os sistemas de banco de dados objeto-relacionais fornecem um caminho de migração conveniente para os usuários de banco de dados relacionais que desejam usar características orientadas a objeto. Para submeter uma consulta ao BDOR utiliza-se uma extensão da linguagem de consulta SQL.

O Banco de Dados Móveis surgiu para atender a necessidade dos usuários em manter contato com o banco de dados, sem estar em um local fixo. Através de redes sem fio e computadores móveis é possível consultar bancos de dados em máquinas remotas. O processamento de consultas em ambientes móveis é um aspecto que requer ainda algumas soluções para determinadas restrições. Existem muitas características particulares da computação móvel que dificultam o processamento de consultas, como: utilização de baterias, mobilidade, segurança e outras mais. A localização e a otimização dos dados são etapas importantes no processamento de consultas em Banco de Dados Móveis.

O Banco de Dados Multimídia é um repositório que, além de dados convencionais, armazena também áudio, imagens, animações e informações em vídeo que poderão ser manipuladas e recuperadas. Consultar dados multimídia pode apresentar algumas possibilidades fascinantes como encontrar uma combinação particular de palavras em uma coleção de pequenos trechos demonstrativos de *clips*, ou localizar uma pessoa que executa uma certa seqüência de operações numa coleção de *clips* de vídeos. Para realizar uma pesquisa no Banco de Dados Multimídia é necessário que haja interpretação de seu conteúdo. Podendo, ao gerar o resumo dessa pesquisa, resultar em sofisticados esquemas de indexação e análise de algoritmos de imagem e áudio.

9.2 Conclusões

Os seguimentos da sociedade moderna dependem dos recursos computacionais para impulsionar os seus negócios. Os dados processados, armazenados e recuperados no mundo têm crescido de forma exponencial. As tecnologias de Banco de Dados, por sua vez, acompanham esta evolução. Por esta razão, surgiu, e ainda estão surgindo no mercado, novas tecnologias de Banco de Dados que têm como objetivo dar suporte a este crescimento.

Cada aplicação de banco de dados abordada neste trabalho busca atender uma necessidade de mercado. Devido às particularidades das aplicações de banco de dados, conclui-se que o processamento de consultas possui aspectos relevantes que são tratados diferencialmente em cada aplicação.

O processamento de consultas em Banco de Dados Centralizado é relativamente simples, embora exija um conhecimento completo do banco de dados, já o processamento de consultas distribuído requer informações somente locais. Portanto, num ambiente distribuído é necessário selecionar os melhores locais para processar os dados.

O processamento de consultas em Banco de Dados Distribuído é bastante semelhante ao processamento de consultas em Banco de Dados Centralizado, entretanto

possui maior complexidade, pois são implementados através de fragmentos que afetam diretamente o desempenho das consultas. A álgebra relacional, num ambiente distribuído, não é suficiente para expressar estratégias de execução, por isso é preciso complementar com operações de troca de dados.

Em Banco de Dados Distribuídos, para que todos os usuários tenham acesso às informações simultaneamente, é necessário o uso de replicações. O sistema mantém várias cópias idênticas de uma relação original, onde cada cópia é armazenada em diferentes *hosts*. O uso de replicações traz algumas vantagens, como por exemplo, a disponibilidade (caso ocorra falha em um dos *hosts* onde a relação está armazenada, esta pode ser encontrada em um outro *host*, podendo desta forma continuar o processamento das consultas envolvendo a mesma, apesar da falha). Outra vantagem é com relação ao aumento do paralelismo (a replicação dos dados provê maior paralelismo para o sistema, quando se refere a acessos mais direcionados à leitura da relação, permitindo o processamento de consulta simultânea de vários *hosts* a uma mesma relação, minimizando o movimento dos dados entre os nós).

O processamento de consultas em Banco de Dados Distribuído, ao contrário de Banco de Dados Centralizado, considera a localização como uma de suas etapas, onde a distribuição dos dados é justamente tratada nesta fase. A junção ordenada é um aspecto importante na otimização de consulta centralizada e num contexto distribuído é mais importante que junção entre fragmentos, pois pode aumentar o tempo de comunicação. Existem algumas aproximações básicas para junção ordenada em fragmentos de consultas. Tenta-se otimizar a junção ordenada diretamente, considerando que a junção seja substituída por combinações de semijunções para minimizar os custos de comunicação. O custo de comunicação é o fator considerado como mais importante em Banco de Dados Distribuído. Na otimização das consultas distribuídas, o custo de comunicação domina o custo de processamento local (I/O e custo de CPU). Então, a otimização de consulta distribuída visa reduzir o problema, minimizando o custo de comunicação geralmente gasto no processamento local. A vantagem é que a otimização local pode ser feita usando os métodos conhecidos em sistemas centralizados.

Em Banco de Dados Orientado a Objeto o processamento de consulta utiliza as mesmas técnicas desenvolvidas para sistemas relacionais, embora, com algumas dificuldades. A definição das regras de transformação e a manipulação das expressões

de consulta são semelhantes às de sistemas relacionais, com uma diferença, as expressões de consultas relacionais são definidas sobre relações planas, enquanto que as consultas de objetos são definidas sobre classes, coleções ou conjuntos de objetos, que têm relacionamentos de subtipificação e composição entre eles.

A otimização do acesso a objetos por suas hierarquias de herança é um problema que distingue o processamento de consultas orientadas a objetos do processamento de consultas relacionais. O otimizador de consultas em sistemas relacionais considera o número de itens de dados (cardinalidade), o tamanho de cada item de dados e sua organização, entretanto, essas informações não estão disponíveis nos SGBDOOs. A definição de funções de custo, especialmente nas abordagens baseadas em objetos que revelam seus custos, deve ser pesquisada com maior profundidade antes de fazer conclusões satisfatórias. A geração do plano de execução para uma expressão de consulta se preocupa basicamente com a escolha e a implementação dos algoritmos mais eficientes para executar os operadores individuais da álgebra e suas combinações. Em SGBDOOs a questão é mais complicada, devido à diferença nos níveis de abstração de objetos definidos de forma comportamental e seu armazenamento.

Em Banco de dados Objeto-relacional a linguagem SQL e as outras linguagens de consultas têm sido estendidas para tratar tipos complexos e orientação a objeto. Devido à maior diversidade de tipos de dados, a tarefa de otimizar torna-se mais complexa, neste ambiente. Para superar essa dificuldade, alguns produtores permitem que o otimizador possa ser educado ou influenciado para trabalhar de forma mais eficiente com novos tipos de dados e funções. Uma das soluções disponibilizadas pela linguagem *ORACLE* 8 é a inclusão de uma dica, diretamente no comando SQL, que guiará o otimizador na realização dessa consulta. Uma segunda possibilidade seria permitir ao usuário a influência na estimação de custos entre as várias estratégias de acesso.

Um otimizador de consulta convencional aplica certas leis de transformação para reescrever consultas. Contudo, historicamente, essas leis de transformação foram todas embutidas no código do otimizador. Por contraste, em um sistema objeto-relacional, o conhecimento relevante precisa ser mantido no catálogo, implicando em mais extensões. Nesses sistemas o próprio otimizador precisa ser reescrito.

Em Banco de Dados Móveis a mobilidade aponta limitações e desafios que em

ambientes fixos são tratados com facilidade e funcionam sem maiores dificuldades. As dificuldades que a mobilidade impõe vão desde a velocidade do canal, passando por interferências do ambiente e localização da estação móvel, até a duração da bateria desta estação.

O processamento de consultas dentro de ambientes móveis é um aspecto que requer ainda muitas pesquisas, pois a mobilidade adiciona novas características que não aparecem em ambientes convencionais. Os computadores móveis devem operar mesmo desconectados da rede, o que não acontece em sistemas de banco de dados distribuídos. Embora o ambiente de computação móvel seja apenas um sistema distribuído, os aspectos da relação custo/performance do sistema diferem dos tradicionais sistemas distribuídos. A comunicação para um computador móvel custará mais do que entre computadores não móveis. Este custo não é só em termos de custo de comunicação de rede, mas também em termos de duração da bateria.

A diferença mais significante, entre ambientes distribuídos e ambientes móveis, é a transparência de localização em aplicações distribuídas e o conhecimento de localização em aplicações móveis. As máquinas móveis não mais possuem endereços fixos de rede, por esse motivo o processamento de consultas é dificultado, visto que a ótima localização para a resposta da consulta depende deste endereço.

O aspecto de segurança também é uma restrição no processamento de consultas em Banco de Dados Móveis. Existe a facilidade de interceptar mensagens na comunicação sem fio, causando sérios problemas de segurança, devendo então fazer uso de técnicas de criptografia para tentar minimizar este problema.

Para otimizar uma consulta diversos fatores são analisados, como: custo de comunicação, custo de localização, tempo de resposta e etc. Portanto, a mobilidade desta nova tecnologia resulta em alterações dinâmicas no custo da comunicação e conseqüentemente dificulta a otimização. Por este motivo, outras noções de custo começam a ser consideradas como: o tempo do usuário, tempo de conexão, número de bytes ou de pacotes transferidos, tarifas com base em horário e energia limitada.

Os sistemas multimídias devem possuir maior capacidade que sistemas convencionais. Ao contrário dos dados tipo numérico ou caracter, os dados multimídia requerem significativa capacidade de armazenamento e aumento da velocidade de processamento. Os dados multimídia, ao realizar uma pesquisa, requerem a

interpretação de seu conteúdo. Podendo, ao gerar o resumo dessa pesquisa, resultar em sofisticados esquemas de indexação e análise de algoritmos de imagem e áudio. As consultas em banco de dados multimídia usam expressões compostas a partir de palavras-chave.

9.3 Relevância do Trabalho

Considerando as novas tecnologias de banco de dados que estão surgindo no mercado, torna-se possível à realização de diversas pesquisas que enriquecem o desenvolvimento científico.

O tema, Processamento de Consultas em Banco de Dados para Aplicações Avançadas, é bastante interessante, pois existem diferentes aspectos que podem ser estudados. Este trabalho contribui para a solução de problemas apontados nas diversas aplicações de banco de dados, pois descreve sugestões de diversos autores.

9.4 Perspectivas Futuras

A atual demanda de mercado está exigindo investigações que busquem soluções aos diversos aspectos que alguns sistemas de banco de dados não atendem. As recentes tecnologias desenvolvidas, como Banco de Dados Móveis e Banco de Dados Multimídia possuem limitações em determinadas situações que devem ser solucionadas, exigindo ainda diversas descobertas sobre o assunto. Existem alguns aspectos que devem ser aperfeiçoados, buscando qualidade em seu funcionamento, por exemplo, a duração da carga das baterias na computação móvel.

Além das soluções apresentadas, através de sugestões de diversos autores, cada restrição, apontada no decorrer desse trabalho, deve ser individualmente estudada e pesquisada para que novas propostas para a solução dos problemas possam surgir.

REFERÊNCIAS BIBLIOGRÁFICAS

ADJEROH, Donald A. e NWOSU, Kingsley C. *Multimedia database management – requirements and issues.* IEEE Multimedia, v. 4, n.3, Jul-Set 1997, p. 25-33.

ANÁLISE DE PARALELISMO, http://www.dca.fee.unicamp.br/courses/IA875/1s1997/Monografias/azuma2.html, 23/03/2001.

ARAGÃO, Daniel; RIECKEN, Rinalda. *Banco de dados multimídia.* Universidade Católica de Brasília – UCB, 1998.

BADRINATH, B. R.; ACHARYA, A. e IMIELINSKI, T. *Impact of mobility on distributed computations.* Rutgers University – Departament of Computer Science: New Brunswick, NJ, 1993.

BANCILHON, F. *Object, relational, object-relational & relational-object.* SIGS Publications, INC, 1996.

BANCO DE DADOS ORIENTADO A OBJETO, http://www.members.nbci.com/_XMCM/orajac/orajac/BDOO.htm, 18/09/2001.

BANCO DE DADOS ORIENTADO A OBJETO, http://www.unifieo.br/revista/banco.html, 18/09/2001.

BANCO DE DADOS RELACIONAL – OBJETO, http://www.dcc.ufmg.br/pos/html/spg98/anais/alecia/alecia.html, 05/10/2001.

BANCO DE DADOS, http://unifieo.br/revista/banco.html, 18/09/2001.

BANCOS DE DADOS TÓPICOS AVANÇADOS, http://www.lia.ufsc.br/javam/Bd_avançado/indices.pdf, 18/08/2001.

BARON, Sérgio. Multimídia. São Paulo: Global Editora e Distribuidora Ltda., 1995, p. 89-107.

BARRY, D. K. *The object database handbook: how to select, implement and use object-oriented databases.* New York: John Wiley & Sons, 1996.

CASANOVA, Marco Antonio; MOURA, Arnaldo Vieira. *Princípios de sistemas de gerência de bancos de dados distribuídos.* Rio de Janeiro: Campus, 1985. 355p.

CELKO, Joe; CELKO, Jackie. *Verdades e mentiras sobre banco de dados de objetos.* Byte Brasil, São Paulo, v.6, n.10, p. 86-89, out. 1997.

CENTRO DE EDUCAÇÃO PROFISSIONAL DIOMÍCIO FREITAS – Apostila do Curso Técnico em Informática - Banco de Dados – Prof.: Roberto de Medeiros Jr (Tubarão –2001).

COLLYER JUNIOR, Charles Sampaio. *Linguagem de consulta ao SIGO: um sistema gerenciador de objetos.* Rio de Janeiro: Instituto Militar de Engenharia, 1997.

COMPUTAÇÃO MÓVEL, http://www.di.ufpe.br/~cak/movel/movel.html, 05/01/2001.

COMPUTAÇÃO MÓVEL, http://www.dimap.ufrn.br, abril/2001.

COMPUTAÇÃO MÓVEL, http://www.lecom.dcc.ufmg.br, 01/07/2000.

COMPUTAÇÃO MÓVEL, http://www.lecom.dcc.ufmg.br/~sergiool/telefonia/compmovel.htm, 01/07/2000.

DATE, C. J. *Introdução a sistemas de banco de dados.* Rio de Janeiro: Campus, 2000. 7. ed.

DAVID, Michael M. *Multimedia databases through the looking glass.* Database Programming and Design Magazine, 1997.

DBMS on-line, http:// www.dbmsmag.com/9709d17.html, Setembro, 1997.

DUNHAM, Margaret H. e HELAL, Abdelsalam. *Mobile computing and databases: anything new?* Department of Computer Sciences and Engineering, Southern Methodist University, Dallas, Texas, 1995. Disponível em http://www.seas.smu.edu/~mhd/pubs/95/record.ps, 10/11/2000.

DUNHAM, Margaret H. e KUMAR, V. *Location dependent data and its management in mobile databases.* Mobility in databases and distributed systems Workshop at Dexa'98, agosto de 1998, pp.414 - 419

ELLMER, Ernest. *Object orientation – na overview* – persistence. http://www.ifs.univie.ac.at/ISOO/OODB/persistence.html, 16/09/1998.

ELMASRI, Ramez; NAVATHE, Shamkant B. *Fundamentals of database systems.* Addison Wesley, 2000.

FREYTAG, Johann Christoph; MAIER, David; VOSSEN, Gottfried. *Query processing: for advanced database systems.* California: Morgan Kaufmann Publishers, 1994. 481p.

HASHING, http://www.icms.sc.usp.br/~sce183/hash1.html, 18/08/2001.

HEUSER, Carlos Alberto. *Projeto de banco de dados.* Porto Alegre: Sagra Luzzato, 2000.

INDEXAÇÃO E HASHING, http://www.icmsc.sc.usp.br/~sce183/hash1.html, 18/08/2001.

INDEXAÇÃO E HASHING, http://www.lia.ufc.br/javam/bd_avancado/índices.pdf, 18/08/2001.

ITO, Giani Carla. *Banco de dados movéis: uma análise de soluções propostas para gerenciamento de dados.* Florianópolis: UFSC, 2001.

KIM, Won et al. *Observations on the ODMG-93 proposal for an object-oriented database language*, UniSQL Inc., Austin - Texas, 1993,

KLEIN, Lawrence Zordam. *A tecnologia relacional-objeto em um ambiente de data warehouse.* Rio de Janeiro, 1999.

KROSHAFIAN, Setrag. *Banco de dados orientado a objeto.* Rio de Janeiro: Infobook, 1994.

MARSH, Brian; DOUGLIS, Fred; CÁCERES, Ramon. *System issues in mobile computing.* Matsushita Information Technology Laboratory: Princeton, NJ, 1993. Disponível em http://www-tkn.ee.tu-berlin.de/bibl/ps/mobi-system-issues.os.gz, 19/10/2000.

NEC DO BRASIL S.A., http:// www.nec.com.br, 10/05/1999.

ÖZSU, M. Tamer; VALDURIEZ, Patrick. *Principles of distributed database systems.* New Jersey: Prentice-Hall, 1999. 2nd ed. 665 p.

PAZANDAK, Paul e SRIVASTANA, Jaideep. *Evaluating object DBMSs for multimedia.* IEEE Multimedia, v. 4, n.3, Jul-Set. 1997, p. 34-49.

PROCESSAMENTO DISTRIBUÍDO DE CONSULTAS, http://www.dcc.ufmg.br/~barra/siam/processamento, 2303/2001.

RAMAKRISHNAN, Raghu. *Data management systems.* Singapore: WCB McGrall-Hill, 1998.

SATYANARAYANAN, M. *Fundamental challenges in mobile computing.* Relatório Técnico. CMU-CS-96-111, Philadelphia, 1999, p. 1-7.

SGBDOO, http://www.pr.gov.br/celepar/celepar/batetyte/edicoes/1998/bb78/colunado.htm, 18/09/2001.

SILBERSCHATZ, Abraham; KORTH, Henry F. e SUDARSHAN, S. *Sistema de banco de dados.* São Paulo: Makron Books, 1999. 3. ed. 778 p.

SSU, Kuo; YAO, Bin; FUCHS, Kent; NEVES, Nuno Ferreira. *Adaptive Checkpointing with storage management for mobile environments, manuscript.* Dezembro 1998.

UFRGS – Grupo de Redes. *Transmissão de Dados sem Fio.* http://penta.ufrgs.br/redes.94-2/lisianeh/wireless.html. UFRGS, Porto Alegre, 1996.