Capítulo 2

Data Warehousing

O tema *data warehousing* engloba arquiteturas, algoritmos e ferramentas que possibilitam que dados selecionados de provedores de informação autônomos, heterogêneos e distribuídos sejam integrados em uma única base de dados, conhecida como *data warehouse*. Através da arquitetura de um ambiente de *data warehousing* é possível identificar os componentes que participam do ambiente, o relacionamento que existe entre estes componentes e as funcionalidades de cada um. Tais funcionalidades são oferecidas por algoritmos com propósitos específicos e, muitas vezes, assistidas por ferramentas. Já os provedores de informação podem possuir uma variedade de formatos e modelos e podem incluir desde SGBD relacionais, orientados a objetos e objeto-relacionais até bases de conhecimento, sistemas legados (tais como sistemas hierárquicos e de rede), documentos HTML e SGML (*Standard Generalized Markup Language)*, dentre outros [Wid95].

O acesso à informação integrada dos diferentes provedores de informação é realizado, geralmente, em duas etapas:

- a informação de cada provedor é extraída previamente, devendo ser traduzida, filtrada, integrada à informação relevante de outros provedores e finalmente armazenada no *data warehouse*; e
- as consultas, quando realizadas, são executadas diretamente no *data warehouse*, sem acessar os provedores de informação originais.

Desta forma, a informação integrada torna-se disponível para consulta ou análise imediata de usuários de SSD e programas de aplicação. Por exemplo, uma aplicação de *data warehousing* de uma cadeia de supermercados poderia integrar dados relativos a vendas, produtos e promoções ao longo dos anos. Usuários desta aplicação poderiam realizar, utilizando estas informações integradas, análises de tendências simples (quais as vendas mensais de um determinado produto no ano de 1998?), análises comparativas (quais as vendas mensais dos produtos de uma determinada marca nos últimos três anos?) e análises de tendência múltiplas (quais as vendas mensais dos produtos de uma determinada marca nos últimos três anos, de acordo com as promoções realizadas no período do Dia dos Namorados e do Dia das Mães?).

Como o *data warehouse* armazena informações integradas, cujas diferenças semânticas e de modelo já foram eliminadas, ambas as consultas e as análises podem ser realizadas rápida e eficientemente. Além disto, uma vez que as consultas e as análises são executadas diretamente no *data warehouse* sem acessar os provedores de informação originais, o *data warehouse* encontra-se disponível mesmo quando os provedores não estiverem disponíveis [HGMW+95]. Outra vantagem refere-se ao fato de que o processamento local nos provedores de informação originais não é afetado por causa da participação destes no ambiente de *data warehousing*.

Nos últimos anos, tem havido um crescimento explosivo da tecnologia de *data warehousing* com relação ao volume de produtos e serviços oferecidos, ao volume de dados manipulado por aplicações que utilizam esta tecnologia, e à adoção desta tecnologia pela indústria. Outro aspecto através do qual pode-se verificar tal crescimento refere-se ao número cada vez maior de usuários que utilizam estas aplicações, como discutido detalhadamente na seção 1.1.

O mercado de *data warehousing*, incluindo *hardware*, *software* de banco de dados e ferramentas de acesso está expandindo rapidamente. Em 1995, o tamanho deste mercado era de 2 bilhões de dólares [Rud96] e em 1997, de 3,5 bilhões de dólares [Wie97]. Em 1999, pesquisas previam um crescimento deste mercado de 5 bilhões de dólares no início desse ano para 15 bilhões de dólares por volta do ano 2000 [Met99a]. Uma tabela contendo estimativas de vendas, juntamente com a taxa esperada de crescimento anual para seis diferentes categorias de ferramentas pode ser encontrada em [Vas00]. A tabela apresentada refere-se aos anos de 1998 a 2002, enquanto que as seis categorias de ferramenta referem-se a: vendas de SGBD relacionais para *data warehouses*; *data marts*; ferramentas de extração, tradução e limpeza; ferramentas de qualidade dos dados; gerenciadores de metadados; e ferramentas OLAP, incluindo serviços de implementação.

Não apenas o mercado está crescendo, mas também o volume de dados manipulado pelas aplicações de *data warehousing*. Por volta de 1996, a maioria das aplicações utilizava *data warehouses* com volume na faixa de 50 *gigabytes* [Rud96]. Já em 1999 tinha-se a expectativa de que 30% dos *data warehouses* existentes excederiam 1 *terabyte* de dados [Met99a]. Por exemplo, o *data warehouse* Wal-Mart Stores, Inc. [Wal] continha 24 *terabytes* de dados em 1998 [GMLWZ98].

Por fim, diversas corporações, através de todos os tipos de negócio, estão utilizando a tecnologia de *data warehousing* para disponibilizar melhores serviços aos clientes e realizar estratégias de *marketing*. Exemplos de indústrias que exploram esta tecnologia e as suas respectivas aplicações, além de algumas empresas comerciais classificadas de acordo com estas indústrias são: bancária e de serviços financeiros, para a redução de fraudes e a análise de riscos de crédito – Bank of America [Ban], ARM Financial Group, Inc. [ARM]; de serviços e de consultoria, para a análise de utilização – ADP Brokerage Information Service Group [ADP]; de fabricação, para o suporte ao cliente e à remessa de encomendas – Xerox Corporation Corporate Strategic Services (CSS) [Xer], General Electric [Gen]; de telecomunicações, para a análise das chamadas e de tarifas – AT&T Corp. [ATT]; de varejo, para a exploração de vendas de produtos, a análise de promoções e o gerenciamento de estoque – Wal-Mart Stores, Inc.; e de transporte e de distribuição, para o gerenciamento de logística – American Airlines [Ame]. Outras corporações que utilizam esta tecnologia englobam indústrias seguradoras (Fremont Compensation Insurance Company [Fre]), de saúde (Hospital das Clínicas USP – INCOR [Hos]), farmacêuticas (Agouron Pharmaceuticals [Ago]), petroquímicas (Shell E&P Company [She]) e de energia (San Diego Gas & Electric [San]), além de organizações governamentais (United States Postal Service [Uni]) e universidades (Fordham University [For]).

2.1 *Data Warehouse*

O surgimento do *data warehouse*, considerado o coração do ambiente de *data warehousing*, foi motivado pela necessidade de separação entre os ambientes operacional e informacional das empresas. De maneira geral, um *data warehouse* é um banco de dados voltado para o suporte aos processos de gerência e tomada de decisão, e tem como principais objetivos prover eficiência e

flexibilidade na obtenção de informações estratégicas e manter os dados sobre o negócio com alta qualidade.

A obtenção de informações estratégicas, relativas ao contexto de tomada de decisão, é de suma importância para o sucesso de uma empresa. Tais informações permitem à empresa um planejamento rápido frente às mudanças nas condições do negócio, essencial na atual conjuntura de um mercado globalizado.

Inicialmente, a obtenção de informações estratégicas era efetuada diretamente no ambiente operacional, o qual é constituído por aplicações responsáveis pela operação da empresa. Dentro deste contexto, programas de extração foram criados com o objetivo de obter os dados relevantes à tomada de decisão. Programas de extração são caracterizados por selecionarem determinados dados de um arquivo ou banco de dados e por armazenarem estes dados em outro arquivo ou banco de dados separado, conhecido como extrato de dado. Tais extratos permitiam a manipulação dos dados sem o comprometimento da integridade destes com relação às aplicações já existentes, além de um melhor desempenho do que o desempenho proporcionado com o compartilhamento de recursos computacionais.

A proliferação de extratos de dados e o subseqüente uso desses extratos resultou na criação de uma verdadeira "teia de aranha", na perda de credibilidade dos dados gerados nas análises e relatórios produzidos para o nível estratégico e na diminuição da produtividade de relatórios contendo informações oriundas de diversos extratos de dados, uma vez que os dados existentes eram heterogêneos e inconsistentes. Como exemplo, departamentos podiam produzir, a partir de seus extratos de dados, resultados distintos para um mesmo relatório. Tal inconsistência era gerada a partir das características de cada extrator, os quais possuíam algoritmos de extração diferentes contendo fontes de dados e período de coleta de dados distintos. Ademais, a localização e o tratamento dos dados e o desenvolvimento de extratores customizados para tecnologias diferentes demandavam enorme esforço e tempo, diminuindo a produtividade do processo de análise. Muitas vezes, a ausência de dados históricos ou a diferença temporal presente nos diversos sistemas não integrados inviabilizavam a obtenção de informações estratégicas a partir dos dados armazenados.

Devido aos problemas acima destacados, a separação entre o ambiente operacional, constituído por aplicações que dão suporte ao dia a dia do negócio, e o ambiente informacional, constituído por aplicações que analisam o negócio, tornou-se uma forte tendência. Como conseqüência, o *data warehouse* emergiu como a base para o ambiente informacional de uma empresa. Para tanto, manipula os dados presentes nos diversos provedores de informação que compõem o ambiente operacional da empresa, extraindo-os, transformando-os e integrando-os para fins de exploração e análise. Desde que o *data warehouse* é povoado por dados gerados pelo ambiente operacional, o fluxo de informação em um ambiente de *data warehousing* é geralmente no sentido provedores de informação → *data warehouse*.

2.1.1 Diferenças entre os Ambientes Operacional e Informacional

O ambiente de dados para o suporte aos processos de gerência e tomada de decisão (ambiente informacional) é fundamentalmente diferente do ambiente convencional de processamento de transações (ambiente operacional). Tipicamente, o *data warehouse* é mantido separadamente dos bancos de dados operacionais. Segundo Inmon [Inm96], esta separação é motivada pela diferença presente nos dados, tecnologias, usuários e processamento de ambos ambientes, além da necessidade de segurança e de desempenho na execução das aplicações.

A Tabela 2.1 contrasta as principais diferenças existentes entre os ambientes operacional e informacional [BS96, Ken96, BS97, WB97]. Já a seção 2.1.2 descreve detalhadamente as características dos dados do *data warehouse*, enquanto que a seção 2.1.3 aprofunda a discussão sobre granularidade dos dados.

Aspecto	Ambiente Operacional	Ambiente Informacional
ambiente	voltado ao processamento de transações OLTP	voltado ao processamento de consultas OLAP[1]
tipos de operação mais freqüentes	atualização remoção inserção	leitura
volume de transações	relativamente alto	relativamente baixo
características das transações	pequenas e simples acessam poucos registros por vez	longas e complexas acessam muitos registros por vez e realizam várias varreduras e junções de tabelas
tipos de usuários	administradores do sistema projetistas do sistema usuários de entrada de dados	usuários de SSD por exemplo: executivos, analistas gerentes, administradores
número de usuários concorrentes	grande (geralmente milhares)	relativamente pequeno (geralmente centenas)
interações com os usuários	pré-determinadas estáticas	*ad-hoc* dinâmicas
volume de dados	*megabytes* a *gigabytes*	*gigabytes* a *terabytes*
projeto do banco de dados	normalizado para suporte às propriedades ACID (atomicidade, consistência, isolação e durabilidade)	multidimensional, refletindo as necessidades de análise dos usuários de SSD
granularidade dos dados	detalhado	agregado detalhado
principal questão de desempenho	produtividade da transação	produtividade da consulta
tempo de resposta	geralmente poucos segundos	de minutos a horas
exemplos de aplicação	transações bancárias contas a pagar empréstimos de livros	planejamento de *marketing* análise financeira

Tabela 2.1 Ambiente operacional *versus* ambiente informacional.

[1] o termo OLAP foi introduzido em 1993 por Codd *et al*. [CCS93] para definir a categoria de processamento analítico sobre um banco de dados histórico voltado para os processos de gerência e tomada de decisão.

2.1.2 Características dos Dados

Os dados presentes em um *data warehouse* são caracterizados por serem orientados a assunto, integrados, não-voláteis e históricos [Inm96, Mel97].

Em contraste ao ambiente operacional organizado a partir de aplicações funcionais (orientado a aplicações), os dados armazenados em um *data warehouse* são **orientados a assunto**, ou seja, relativos aos temas de negócio de maior interesse da corporação, tais como clientes, produtos, promoções, contas e vendas. Esta abordagem é centrada em entidades de alto nível, com ênfase exclusivamente na modelagem de dados, contendo somente dados relevantes ao contexto de tomada de decisão e estruturados, em geral, de forma desnormalizada.

A **integração** dos dados presentes no *data warehouse* é a base para uma análise precisa sobre toda a organização. Os dados de um *data warehouse* são obtidos, em sua grande maioria, a partir do ambiente operacional, o qual possui espalhado em diversos sistemas não integrados informações importantes relativas ao negócio da empresa. No processo de extração, somente um subconjunto dos dados operacionais necessários à análise estratégica é copiado, sendo requerido nesse processo a correção de possíveis inconsistências devido a diferenças semânticas, nos formatos dos dados e na utilização de sistemas operacionais distintos. Como exemplo, devem ser resolvidos problemas de homônimos e de sinônimos, e conflitos de chave e de domínio, além da forma de representação dos dados.

A característica de **não-volatilidade** está relacionada ao fato de que o conteúdo do *data warehouse* permanece estável por longos períodos de tempo. O ambiente de *data warehousing* é caracterizado pela carga volumosa de dados e pelo acesso a estes dados através de consultas efetuadas por usuários de SSD (ambiente do tipo carregamento de dados e acesso). Assim, apenas dois tipos de transações, ambas caracterizadas por serem de longa duração e complexas, são geralmente efetuadas: transação de manutenção (para a carga dos dados, visando a manutenção da consistência do conteúdo do *data warehouse* com relação ao dados dos provedores de informação) e transação de leitura (consultas dos usuários de SSD, do tipo somente para leitura (*read-only*)).

Em sistemas de *data warehousing* comerciais atuais, a transação de manutenção é tipicamente a única transação a atualizar o conteúdo dos dados. Essa transação é realizada durante a "janela noturna", a qual representa o período no qual o *data warehouse* permanece indisponível para transações de leitura. Ou seja, operações de modificação (remoção, inserção e atualização) não são efetuadas durante o período de utilização do *data warehouse*. Com isto, pode-se efetuar simplificações no gerenciamento dos dados, tais como mecanismos de controle de *deadlock* e recuperação de falhas. Em especial, as operações de atualização e remoção ocorrem somente em caso de carga incorreta, ao passo que a operação de inserção é caracterizada por apenas anexar dados aos já existentes (operação do tipo somente para inserção no final do arquivo (*append-only*)). No entanto, já existe pesquisa no sentido de se aumentar a disponibilidade de utilização do *data warehouse*, visando que transações de atualização e de leitura sejam manipuladas pelo sistema de forma *on-line*, simultaneamente [QW97].

Por fim, os dados do *data warehouse* são **históricos**, ou seja, relevantes a algum período de tempo, em contraste com o ambiente operacional, no qual os dados são válidos somente para o momento de acesso. Para cada mudança relevante no ambiente operacional é criada uma nova entrada no *data warehouse*, a qual contém um componente de tempo associado implícita ou explicitamente. Devido à característica temporal dos dados, torna-se possível efetuar análise de tendências, uma vez que dados relativos a um grande espectro de tempo (5 a 10 anos) encontram-se disponíveis. Por outro lado, tal característica influencia diretamente no tamanho do

banco de dados. Em geral, o volume de dados armazenado é muito superior ao volume presente no ambiente operacional (Tabela 2.1), representando uma complexidade adicional à administração de um *data warehouse*.

2.1.3 Organização dos Dados

Em um *data warehouse*, os dados são organizados segundo diferentes níveis de agregação (Figura 2.1). O nível inferior possui dados primitivos coletados diretamente do ambiente operacional, constituindo a base para futuras investigações desconhecidas. No entanto, devido ao grande volume de dados periodicamente coletado, ocorre o processo de envelhecimento, no qual os dados presentes no nível inferior são transferidos, no mesmo nível de granularidade, para o nível antigo. Enquanto que no nível inferior os dados são armazenados em discos, no nível antigo os dados são armazenados em fitas. A lentidão no acesso aos dados do nível antigo torna necessária a sua monitoração, a qual pode ser efetuada através da taxação pelos recursos consumidos e pela limitação da quantidade de acessos aos dados de acordo com o perfil de cada usuário de SSD.

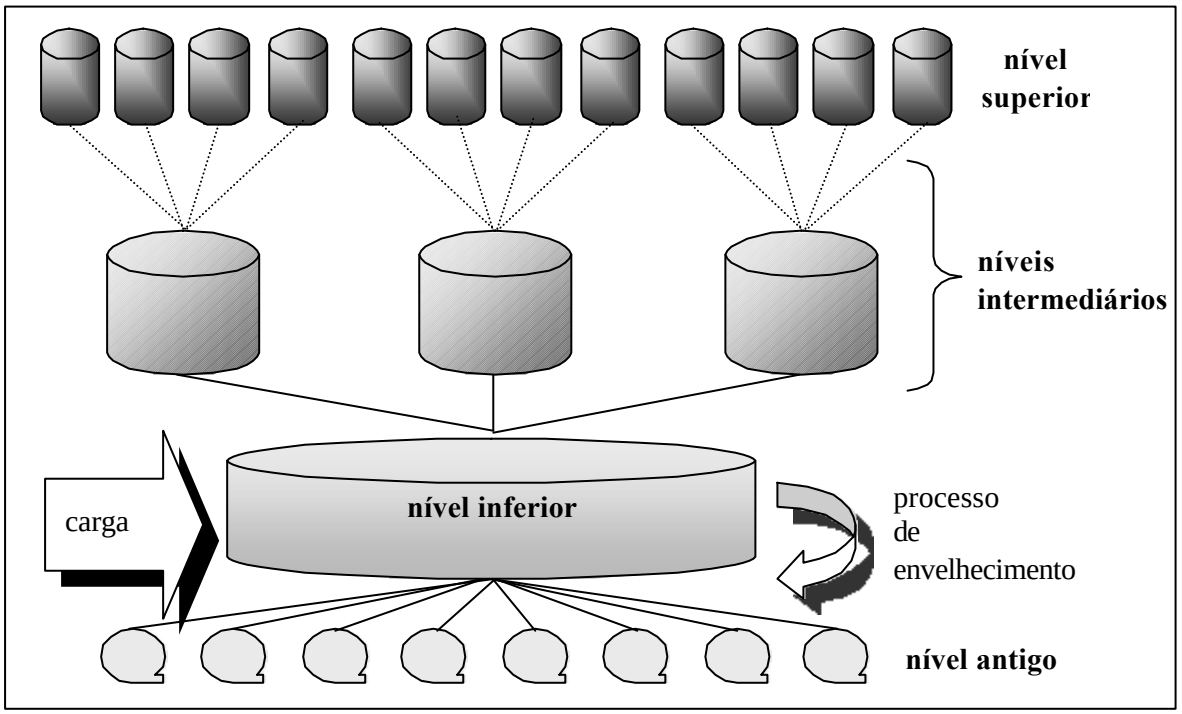

Figura 2.1 Estrutura de um *data warehouse*.

Por outro lado, o nível superior da hierarquia de agregação possui dados altamente resumidos. Entre os níveis inferior e superior podem existir vários níveis intermediários, representando graus de agregação crescente. Os dados armazenados em um nível n correspondem a alguma forma de agregação dos dados armazenados em um nível $n-1$, sendo que o nível intermediário mais inferior utiliza os dados do nível inferior como base para suas agregações. Dentro deste contexto, os dados detalhados do *data warehouse* (nível inferior da hierarquia de agregação) podem ser vistos como um conjunto de visões materializadas definidas sobre as relações base armazenadas nos provedores de informação operacionais. Já cada um dos demais níveis de agregação (dados agregados) pode ser visto como um conjunto de visões materializadas definidas sobre os dados do nível imediatamente subjacente.

Dados resumidos são compactos, armazenados em disco, altamente indexados, facilitam o processo de reestruturação e permitem a visualização de panoramas específicos e globais da empresa, de acordo com o grau de agregação. Usuários de SSD devem ser estimulados a acessar os dados dos níveis superiores, os quais demandam menor esforço no uso de recursos computacionais.

Desta forma, usuários de SSD podem iniciar suas análises no nível superior para obter uma visão geral do negócio, podendo percorrer a hierarquia de agregação até o nível inferior à medida que dados específicos são necessários. Como exemplo, em uma aplicação de *data warehousing* para uma cadeia de supermercados, as seguintes consultas poderiam ser solicitadas: (i) vendas anuais dos produtos da marca `M` em todas as filiais; (ii) vendas mensais no ano de `1998` dos produtos da marca `M` fora de promoção nas filiais `1` e `2`; e (iii) vendas diárias no mês de `outubro de 1998` do produto `P` da marca `M` fora de promoção na filial `1`.

A granularidade dos dados armazenados no nível inferior do *data warehouse* é uma questão de projeto muito importante, uma vez que determina a dimensionalidade do banco de dados. Granularidade refere-se ao nível de detalhe em que as unidades de dados são mantidas. Quanto maior o nível de detalhe dos dados, menor o nível de granularidade, e vice-versa.

A escolha da granularidade dos dados do nível inferior causa um impacto profundo no volume de dados armazenado no *data warehouse*, além de afetar os tipos de consulta que podem ser respondidas pelo sistema. Quando se tem um nível de granularidade muito pequeno, o tamanho do *data warehouse* é muito grande, porém praticamente qualquer consulta pode ser respondida. Por outro lado, quando o nível de granularidade é muito alto, ambos o tamanho do *data warehouse* e o número de questões que podem ser manipuladas são reduzidos. Outros impactos causados pela granularidade dos dados são relacionados às estruturas de indexação necessárias e à quantidade de recursos computacionais utilizados no processamento da consulta.

No exemplo da cadeia de supermercados, o qual armazena as vendas de produtos com relação a promoções e filiais ao longo dos anos, uma granularidade apropriada para o nível inferior é o movimento diário de produtos, ou seja, *produto por filial por promoção por dia* [Kim96]. Caso os dados armazenados representassem o movimento anual de produtos (*produto por filial por promoção por ano*), consultas importantes para a tomada de decisão neste ramo, tal como "Desde o início deste ano, as vendas do produto `P` cresceram ou diminuíram?" não poderiam ser respondidas. Por outro lado, o armazenamento de cada produto comprado em cada compra por cada consumidor implicaria em um *data warehouse* extremamente volumoso.

Com relação às informações a serem armazenadas nos demais níveis de agregação, já existem vários trabalhos realizados no sentido de se determinar o conjunto de agregações a serem armazenadas de forma que o custo total de armazenamento seja mínimo. Estes trabalhos são abordados na seção 2.5.1.

Como pode ser observado, um sistema de *data warehousing* não armazena somente dados resumidos, mas também dados primitivos coletados diretamente do ambiente operacional. Isto garante maior flexibilidade ao sistema, uma vez que este pode responder tanto a consultas e análises previamente antecipadas quanto a consultas e análises *ad-hoc*.

2.2 Arquitetura Típica de um Ambiente de *Data Warehousing*

A Figura 2.2 ilustra a arquitetura genérica utilizada para criar, manter e consultar um *data warehouse*.

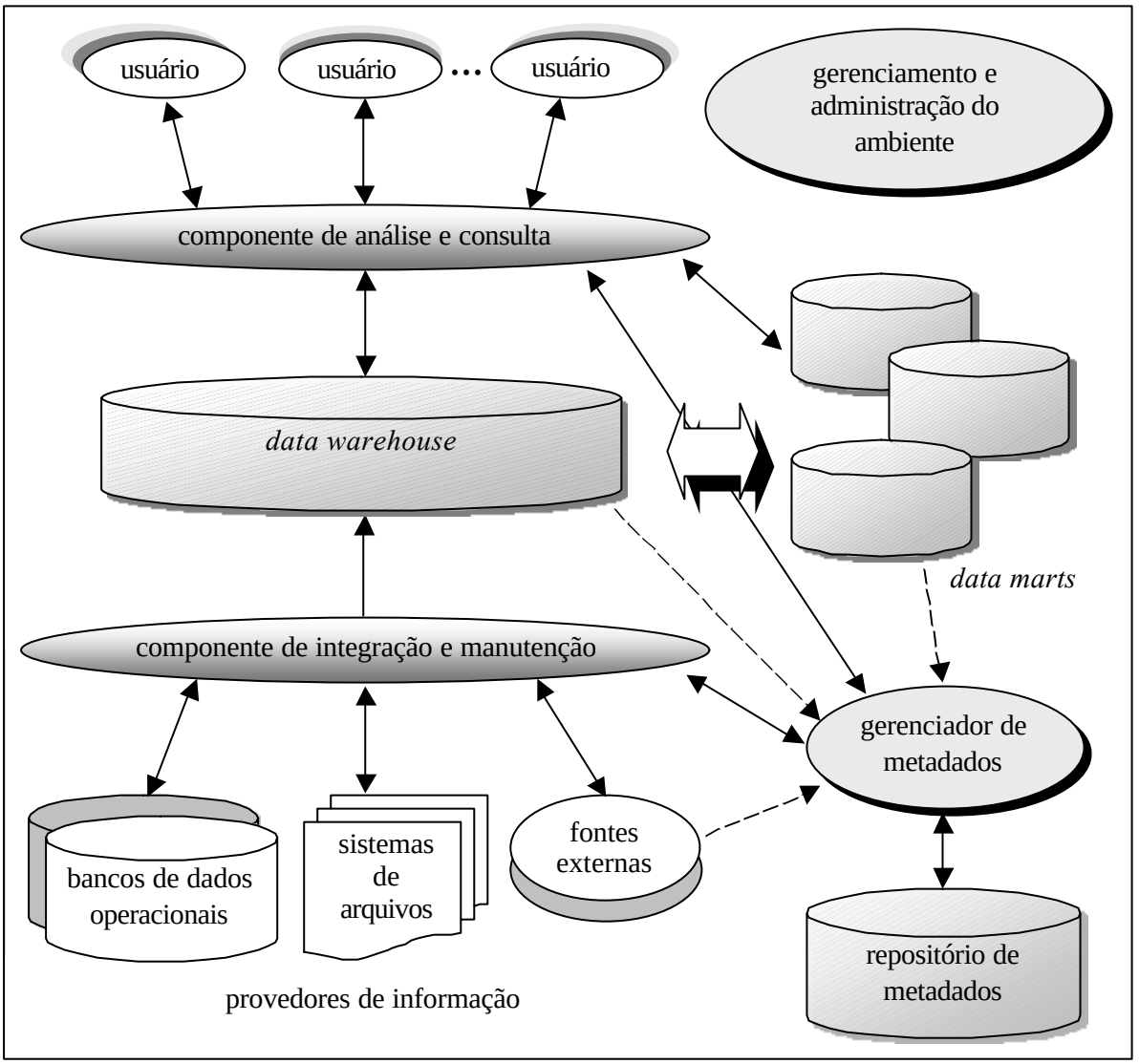

Figura 2.2 Arquitetura típica de um ambiente de *data warehousing*.

Os **provedores de informação** contêm dados operacionais armazenados segundo diferentes modelos e formatos. Dados relevantes destes provedores são extraídos, traduzidos, filtrados, integrados e armazenados no *data warehouse* pelo **componente de integração e manutenção**, o qual também é responsável por atualizar o *data warehouse* periodicamente para refletir as alterações nos dados dos provedores de informação e realizar a expiração dos dados do *data warehouse*.

Em adição ao *data warehouse* principal, podem existir diversos ***data marts*** departamentais, contendo réplicas de porções do *data warehouse*. Ambos os dados do *data warehouse* e dos *data marts* são acessados pelos usuários de SSD através do **componente de análise e consulta**, o qual provê visões multidimensionais destes dados. Este componente e o de integração e manutenção não são independentes. Por exemplo, a determinação de quais informações devem ser armazenadas no *data warehouse* depende das necessidades dos usuários de SSD.

Todas as informações estruturais e semânticas dos provedores de informação e do *data warehouse*, além de quaisquer outros dados importantes para o ambiente são armazenadas em um **repositório de metadados** e manipuladas por um módulo **gerenciador de metadados**. Apesar da Figura 2.2 representar apenas um repositório de metadados, White [Whi99] argumenta

que em um futuro próximo a arquitetura incluirá vários repositórios de metadados compartilhados.

Finalmente, a arquitetura engloba ferramentas para o **gerenciamento e administração do ambiente**. Estas ferramentas são responsáveis pelo monitoramento do sistema, realizando tarefas importantes tais como o gerenciamento de segurança, testes de qualidade dos dados, o gerenciamento e a atualização dos metadados e cópias de reserva (*backup*), além de auditoria e relato da utilização do *data warehouse* para gerenciamento do tempo de resposta e do uso dos recursos.

A seção 2.2.1 contextualiza os componentes da arquitetura com relação às fases de projeto, de criação, de manutenção e de acesso do ambiente. As seções 2.2.2 e 2.2.3 descrevem detalhadamente as funcionalidades oferecidas, respectivamente, pelo componente de integração e manutenção e pelo componente de análise e consulta. A seção 2.2.4 aprofunda a discussão sobre *data marts*, ao passo que a seção 2.2.5 discute as principais conceitos relacionados a metadados.

2.2.1 Fases de Projeto, Criação, Manutenção e Acesso

A fase de projeto de um ambiente de *data warehousing* inicia-se com a definição da arquitetura a ser utilizada. Nesta etapa deve-se realizar o planejamento da capacidade do ambiente e selecionar *hardware* e *software* apropriados. Dentre as principais atividades envolvidas, pode-se citar:

- selecionar servidores de armazenamento, servidores OLAP e de banco de dados, além de ferramentas clientes;
- integrar os servidores e as ferramentas clientes;
- identificar os provedores dos dados relevantes e integrar estes provedores ao ambiente;
- identificar os dados a serem armazenados no *data warehouse*, tanto no nível inferior quanto nos demais níveis da hierarquia de agregação; e
- definir a organização física do *data warehouse*, incluindo a escolha de métodos de acesso.

Após a definição e a especificação do projeto do ambiente, o componente de integração e manutenção deve realizar o carregamento dos dados dos provedores de informação no *data warehouse*. Este carregamento ocorre tanto durante a fase de criação, na qual o *data warehouse* é povoado pela primeira vez, quanto durante a fase de manutenção, na qual alterações nos dados dos provedores são refletidas no *data warehouse*. Outras atividades da fase de manutenção são descritas na seção 2.2.2. Finalmente, o componente de análise e consulta oferece funcionalidades relacionadas ao acesso aos dados, ou seja, como os dados corretamente armazenados no ambiente de *data warehousing* podem ser utilizados por usuários de SSD.

2.2.2 Componente de Integração e Manutenção

O componente de integração e manutenção oferece as seguintes funcionalidades: carregamento dos dados dos provedores de informação no *data warehouse*, atualização periódica desta base de dados e expiração de seus dados. Essa seção descreve cada uma destas funcionalidades em maior nível de detalhe.

2.2.2.1 Carregamento dos Dados

O carregamento dos dados dos provedores de informação, o qual engloba os processos de extração, de tradução, de limpeza, de integração e de armazenamento, representa a atividade mais complexa, cara e demorada, sendo essencial ao bom funcionamento do *data warehousing*. Assim sendo, antes dessa atividade ser iniciada, deve ser desenvolvido um plano de carregamento dos dados para se determinar a melhor forma de migrar os dados dos provedores para o *data warehouse*. Este plano deve ponderar diversos fatores, tais como os recursos disponíveis, os volumes de dados dos provedores, o número de esquemas distintos dos provedores e o número e os tipos de métodos de acesso e plataformas diferentes, além do projeto do *data warehouse*.

Existem diversas ferramentas que auxiliam a atividade de carregamento dos dados, as quais podem ser específicas, enfocando apenas um dos processos, ou integradas, enfocando dois ou mais dos processos envolvidos. Exemplos dessas ferramentas, incluindo as suas respectivas características, podem ser encontrados em [Orl96, BS97, Wil97, Gre].

As seções 2.2.2.1.1 a 2.2.2.1.5 discutem respectivamente particularidades dos processos de extração, de tradução, de limpeza, de integração e de armazenamento dos dados, ao passo que a seção 2.2.2.1.6 salienta a importância do processo de recuperação de falhas.

2.2.2.1.1 Extração dos Dados

O processo de extração dos dados dos provedores de informação que participam do ambiente de *data warehousing* é usualmente implementado de três formas básicas: através de uma interface de comandos padronizados (interface comum), através de um protocolo comum de acesso a dados e através de um conversor de comandos de manipulação de dados e de formatos de dados (*gateway*) [IH94, MST97]. Cada uma destas três abordagens enfatiza um elemento distinto que é comum entre o cliente e o servidor. Considerando-se o processo de extração, uma aplicação que realiza a extração dos dados pode ser considerada um cliente, ao passo que cada provedor de informação é um servidor de dados.

A primeira abordagem usa a interface (tanto do cliente quanto do servidor) como um elemento comum para o acesso aos dados gerenciados por diferentes produtos baseados na linguagem SQL. O princípio por trás desta abordagem consiste na especificação de programas de aplicação clientes de acordo com uma interface genérica (API – *Application Programming Interface*), a qual é independente do tipo do servidor. Em tempo de execução, a aplicação identifica qual a fonte de dados a ser acessada e um módulo específico (*driver*) converte os formatos de dados e os comandos padronizados para os formatos compreendidos pelo servidor alvo. Como resultado, o código da aplicação cliente não precisa ser alterado quando esta for redirecionada para outro tipo de servidor. Exemplos de padrões de interface comum incluem o ODBC (*Open Database Connectivity*) e o BDE (*Borland Database Engine*).

Por outro lado, a abordagem de protocolo comum utiliza um protocolo bem definido para conectar aplicações clientes aos vários tipos de servidores. Intuitivamente, cada interface cliente recebe requisições de aplicações clientes e as traduz em um protocolo comum apropriado. De forma similar, cada interface servidora identifica requisições geradas de acordo este protocolo, processa tais requisições e traduz os resultados obtidos de acordo com o protocolo comum. Através desta abordagem, a interoperabilidade entre os tipos de servidores que utilizam o mesmo protocolo para a comunicação é garantida mesmo que diferentes API tenham sido empregadas. Os padrões DRDA (*Distributed Relational Database Architecture*) e RDA (*Remote Data Access*) são exemplos de protocolos comuns.

Já a abordagem de *gateway* é representada pelo uso de tradutores de comandos de acesso a dados (*gateways* para bancos de dados) como elementos comuns entre clientes e servidores. Tais tradutores assumem a funcionalidade de mapeamento de dados e de comandos entre os vários clientes e servidores. Por não serem padronizados, *gateways* reduzem a portabilidade das aplicações desenvolvidas, uma vez que estas utilizam as API por eles disponibilizadas. Como exemplos de produtos de *gateway* pode-se citar o Database Gateway para DB2 e o Informix Enterprise Gateway.

As alternativas acima descritas podem ser utilizadas tanto separadamente quanto conjuntamente, de acordo com as necessidades da organização. A abordagem de *gateway* usualmente complementa ou estende as outras duas abordagens. Quando desenvolvida com alguma combinação de interface comum ou protocolo comum, adiciona flexibilidade à arquitetura do sistema.

2.2.2.1.2 Tradução dos Dados

O processo de tradução consiste na conversão dos dados do formato nativo dos provedores de informação para o formato e o modelo utilizados pelo ambiente de *data warehousing*. Este processo, assim como o de extração dos dados, é altamente dependente do provedor sendo considerado. Por exemplo, se o provedor de informação armazena seus dados de acordo com o modelo hierárquico e o *data warehouse* utiliza o modelo relacional, rotinas de tradução dos dados do modelo hierárquico para o relacional devem ser especificadas. Uma vez que os provedores de informação que participam do ambiente armazenam seus dados segundo diferentes formatos, diversas rotinas de tradução devem ser desenvolvidas. Tais traduções devem incluir tanto transformações de esquema (alteração da estrutura dos dados) quanto transformações dos dados correspondentes.

Um aspecto muito importante do carregamento dos dados refere-se à característica temporal dos dados do *data warehouse*. Embora a maioria dos provedores de informação não seja histórico, ou seja, armazene somente valores correntes dos dados, o *data warehousing* deve manter informações históricas, uma vez que os usuários de SSD usualmente estão interessados no histórico de como os dados dos provedores evoluíram ao longo do tempo. Desta forma, durante o processo de tradução dos dados, informações temporais devem ser associadas aos dados sendo manipulados. Segundo Yang e Widom [YW98], tais informações podem refletir tanto o momento de atualização dos dados no provedor de informação quanto o momento de armazenamento destes no *data warehouse*, ou ambos.

O problema de tradução de dados não é específico à área de *data warehousing*, tendo sido amplamente pesquisado na área de bancos de dados heterogêneos [EP90, SL90, BHP92, Hsi92]. Dentre os trabalhos existentes, pode-se citar: Sacramento e Laender [SL94] para a tradução do modelo orientado a objetos (modelo de dados do sistema O_2 [Deu+92]) para o modelo relacional, Keim *et al*. [KKM93] para a tradução do modelo de dados relacional para o modelo orientado a objetos, Premerlani e Blaha [PB94] para mapeamentos do modelo relacional para o modelo de dados da metodologia OMT [RBPE91] e Aguiar [Agu95] para mapeamentos entre os modelos ECR [EWH85] e OMT, e entre este último e o modelo de dados do sistema O_2. Por outro lado, os trabalhos de Cluet *et al*. [CDSS98], Siméon e Cluet [SC98] e Abiteboul *et al*. [ACM+99] descrevem o sistema YAT, o qual oferece ferramentas para a especificação e a implementação de conversões de dados entre provedores de dados heterogêneos. Este sistema é composto por um modelo de dados *middleware* para o qual os dados dos provedores são mapeados, uma linguagem declarativa baseada em regras para especificar o processo de tradução dentro deste modelo de dados, um mecanismo de customização e uma interface gráfica. As traduções de um

formato origem para um formato destino são obtidas através da: (i) importação dos dados origem no modelo de dados *middleware*, (ii) tradução para outra representação *middleware* que melhor represente a estrutura destino e (iii) exportação dos dados traduzidos para o sistema destino. Através da linguagem declarativa pode-se descrever as correspondências existentes entre os dados heterogêneos, as quais são utilizadas para derivar automaticamente as regras de tradução entre os formatos. Dentre os formatos disponibilizados pelo sistema, pode-se citar: SGML, HTML, relacional e o formato do sistema O_2.

2.2.2.1.3 Limpeza dos Dados

Desde que o ambiente de *data warehousing* é utilizado para suporte à decisão, é importante que os dados mantidos no ambiente sejam corretos e de qualidade. Em outras palavras, contribuições potenciais dependem da qualidade das decisões tomadas, as quais, por sua vez, dependem da qualidade dos dados utilizados no processo decisório. Em especial, dentre os diversos fatores que podem acarretar o insucesso de um ambiente de *data warehousing*, a falta de atenção com relação à qualidade dos dados representa um fator extremamente comprometedor. Dados duvidosos e/ou incorretos podem gerar informações errôneas que causam um efeito adverso nos lucros da corporação.

A qualidade dos dados está diretamente relacionada às regras de negócio a eles associadas. Uma vez que estas regras são definidas, o dado deve ser testado para verificar a sua consistência e a sua completude com relação a estas regras. Tais regras podem ser inferidas tanto através da identificação das regras de negócio da corporação e a posterior associação destas regras aos dados quanto através da análise dos próprios dados e da identificação de padrões que se aplicam à maioria dos dados analisados. Se, por exemplo, em 95% dos registros analisados a idade das pessoas que compram carros conversíveis com preços mínimos de R$100.000,00 é superior a 40 anos, um registro com valor de idade igual a 18 anos deve ser examinado cautelosamente.

Existem diversas situações nas quais os valores de dados armazenados podem estar incorretos e, conseqüentemente, nas quais a limpeza dos dados torna-se necessária: comprimentos de campos inválidos, dados incompletos ou em branco, duplicações, descrições inconsistentes, violação de restrições de integridade, associações de valores inconsistentes, abreviações de valores não padronizadas e utilização de caracteres ou de tipos de dados não desejados. A atribuição do valor *350* para um campo *idade* é um exemplo de associação de valor inconsistente, ao passo que a existência de uma chave de cliente em um arquivo que não corresponde às chaves do arquivo de clientes representa uma violação da restrição de integridade. Outros exemplos incluem campos com valores de *cor* e *telefone* incorretos, *endereços* parcialmente preenchidos (somente o campo *CEP* é preenchido, os demais campos – *nome da rua*, *número*, *complemento*, *cidade*, *estado*, *país* – são deixados em branco) e *datas* armazenadas erroneamente na forma *ddmmaa* com significado *mmddaa* (010399: 1° de março de 1999 ou 03 de janeiro de 1999).

Por ser uma atividade de extrema importância para o sucesso de um ambiente de *data warehousing*, o processo de limpeza dos dados não deve ser realizado como uma atividade separada, mas sim durante todos os demais processos de extração, de tradução, de integração e de armazenamento dos dados no *data warehouse*.

2.2.2.1.4 Integração dos Dados

Assim como em qualquer ambiente que envolve vários provedores de informação heterogêneos, em um *data warehousing* existe uma alta probabilidade de se existir várias cópias da mesma

informação em diferentes provedores (representadas de forma igual ou não) ou informações correlatas armazenadas em diferentes provedores que são inconsistentes entre si. Esta diversidade de representações está relacionada, em primeiro lugar, ao fato de que as aplicações representadas por estes provedores foram criadas independentemente, de forma não coordenada e sem considerar que um dia seriam integradas. Depois, a modelagem de cada aplicação varia de acordo com diversos fatores, tais como as necessidades, os objetivos e o universo de atuação de cada aplicação. Esses fatores determinam entidades próprias a cada domínio. Por fim, diferentes analistas podem possuir diferentes percepções do mundo real, originando, muitas vezes, modelos distintos para uma mesma aplicação.

Como resultado desta diversidade de representações, o processo de integração depende da identificação de similaridades e de diferenças existentes entre os dados dos provedores de informação que foram previamente traduzidos, além da identificação de conjuntos destes dados que, apesar de serem distintos entre si, são relacionados por alguma propriedade semântica [Agu95]. Estas similaridades e diferenças devem ser detectadas tanto em nível de esquema quanto em nível de instância.

Considerando-se o nível de esquema, os conflitos existentes entre os dados a serem integrados podem ser divididos em três grupos: conflitos de nome, conflitos semânticos e conflitos estruturais. O primeiro tipo de conflito, conflito de nome, refere-se aos nomes utilizados para representar os diferentes elementos existentes nos esquemas a serem integrados. Diferentes nomes podem ser aplicados ao mesmo elemento (problema dos sinônimos) ou o mesmo nome pode ser aplicado a diferentes elementos (problema dos homônimos). Um exemplo de sinônimo ocorre quando o nome *cliente* é utilizado para representar, em um esquema, todos os clientes atendidos por uma loja, enquanto que o nome *comprador* é utilizado em outro esquema para representar a mesma situação.

Após a identificação dos conflitos de nome, devem ser identificados os conflitos semânticos. Este tipo de conflito surge quando o mesmo elemento é modelado nos diferentes esquemas, porém representando conjuntos que se sobrepõem. Em outras palavras, o conjunto de instâncias do elemento de um esquema é mais abrangente do que o conjunto de instâncias do elemento do outro esquema. Por exemplo, em uma aplicação de um provedor, o elemento *produto* representa todos os produtos de todas as seções de um supermercado, ao passo que em uma outra aplicação de outro provedor, *produto* representa apenas os produtos da seção de cosméticos.

Finalmente, conflitos estruturais surgem sempre que diferentes construtores estruturais são utilizados para modelar o mesmo conceito representado em diferentes aplicações. Como exemplo, considerando-se o modelo entidade-relacionamento, o mesmo conjunto de objetos do mundo real pode ser representado como um tipo-entidade em um esquema e como um atributo de um tipo-entidade em outro esquema. A diversidade de conflitos estruturais que podem aparecer em um problema de integração depende da semântica do modelo de dados utilizado.

Em um ambiente de *data warehousing*, não somente a integração dos esquemas das diferentes aplicações é importante, mas também a integração das suas instâncias correspondentes. Como um exemplo simples da necessidade de integração de instâncias, os dados correspondentes ao campo *sexo* de diferentes provedores podem ser codificados de forma inconsistente. Em uma primeira aplicação, esse campo é instanciado com os valores "M" (masculino) ou "F" (feminino), enquanto que em outra aplicação os valores permitidos são "0" (masculino) ou "1" (feminino). Já em uma terceira aplicação, os valores assumidos podem ser "H" (homem) ou "M"(mulher). Independentemente do formato final do campo integrado *sexo* a ser armazenado no *data warehouse*, os diferentes valores devem ser corretamente decifrados antes de serem armazenados. Outro exemplo consiste em um mesmo campo *largura* armazenado

em quatro diferentes provedores, porém com valores que representam unidades de medidas distintas: centímetros, metros, polegadas e jardas.

Como resultado do processo de integração, tem-se uma visão integrada dos dados do *data warehouse*, sem duplicatas ou inconsistências, e de alta qualidade.

Assim como o problema de tradução dos dados não é específico ao tema *data warehousing*, técnicas de integração de esquemas e de dados propostas na área de bancos de dados heterogêneos podem ser adaptadas a esse tema. Por exemplo, o trabalho de Silva *et al.* [SSU+00] propõe uma arquitetura para ambientes de *data warehousing* que utiliza o SGBD heterogêneo fortemente acoplado HEROS (*Heterogeneous Object System*) [ULM98, UM99] para solucionar o problema de integração dos dados nesses ambientes. Para tanto, o esquema global do sistema HEROS passa a oferecer um metamodelo orientado a objetos baseado no esquema estrela (seção 2.4.2.1.1), o qual é primeiro instanciado e depois especializado de forma a incorporar a semântica de *data warehousing* à federação. A arquitetura proposta faz uso de um programa extrator que requisita ao HEROS a extração/integração dos dados dos provedores de informação e armazena os resultados obtidos no *data warehouse*, baseado no metamodelo correspondente. A idéia subjacente à proposta de arquiteturas de bancos de dados integrados para SGBD heterogêneos e *data warehouses* é introduzida em [SSSB98], sendo descrita mais detalhadamente em [AOSS00]. Esses dois trabalhos também discutem a possibilidade de utilização de um modelo orientado a objetos como modelo de dados canônico para a definição do esquema do *data warehouse*.

2.2.2.1.5 Armazenamento dos Dados no *Data Warehouse*

Após os processos de extração, de tradução, de limpeza e de integração, ocorre o armazenamento dos dados trabalhados no *data warehouse*. Durante esta etapa, vários processamentos adicionais ainda são realizados, tais como a verificação de restrições de integridade, a ordenação dos dados, a geração de agregações, a construção de índices e a condensação dos dados visando-se diminuir o volume de dados a ser armazenado.

2.2.2.1.6 Recuperação de Falhas

Por ser uma atividade complexa e demorada, o carregamento dos dados está sujeito a falhas. Tais falhas podem ocorrer durante qualquer um dos seus processos de extração, de tradução, de limpeza, de integração e de armazenamento, tanto na fase de criação quanto na fase de manutenção do *data warehouse* [DDJ+98]. A idéia fundamental da recuperação de falhas durante o carregamento de dados consiste em evitar que tanto leituras desnecessárias aos dados dos provedores de informação quanto computações cujos resultados já foram armazenados no *data warehouse* sejam realizadas. Como conseqüência, o carregamento é reiniciado, mas somente os dados que não foram corretamente calculados antes da falha ocorrer são considerados.

Dentro deste contexto, o trabalho de Labio *et al*. [LWGMG00] propõe um algoritmo de recuperação de falhas que explora algumas propriedades semânticas genéricas do fluxo de dados que ocorre durante a atividade de carregamento dos dados. Este fluxo de dados é representado por um grafo direcionado acíclico, cujos nós correspondem aos processos de extração e de armazenamento, além de quaisquer outras computações realizadas nos dados, tais como agregações e cujas seqüências de parâmetros de entrada/saída correspondem aos dados que são passados de um nó para o outro. Frente a falhas, o algoritmo identifica quais dados em cada seqüência de entrada já foram processados, remove-os da entrada, de acordo com as

propriedades do fluxo de dados (por exemplo, se os dados são processados em ordem alfanumérica crescente) e com o conteúdo do *data warehouse* e procede o carregamento sem estes dados. Esse trabalho também contrasta diversas técnicas de recuperação tradicionais que poderiam ser adaptadas à atividade de carregamento dos dados em termos da sobrecarga apresentada durante o processamento normal da operação e da capacidade de manipular fluxos de dados genéricos.

2.2.2.2 Atualização dos Dados do *Data Warehouse*

Após o carregamento inicial, os provedores de informação que participam do ambiente de *data warehousing* podem continuar a alterar os seus dados. Visando a manutenção da consistência dos dados do ambiente com relação aos dados dos provedores, tais alterações devem ser propagadas e incorporadas ao *data warehouse* (tanto aos dados do conjunto de visões que representa o nível inferior quanto aos dados das demais visões derivadas armazenadas – demais níveis da hierarquia de agregação dos dados).

A periodicidade da incorporação das alterações aos dados do *data warehouse* depende das necessidades dos usuários de SSD e do nível de consistência desejado. Sistemas comerciais usualmente atualizam seus dados com periodicidade diária ou semanal. No entanto, caso consultas OLAP requeiram dados correntes, cada simples alteração realizada nos provedores de informação deve ser propagada continuamente. A freqüência de incorporação das alterações é determinada pelo administrador do *data warehouse*, podendo ser diferente para provedores distintos.

Existem duas técnicas para se atualizar o *data warehouse*: recomputação e atualização incremental. Na técnica de recomputação, o conteúdo do *data warehouse* é descartado e os seus dados são novamente carregados a partir dos provedores de informação operacionais. Já na técnica de atualização incremental, apenas as alterações nos dados dos provedores de informação são propagadas ao ambiente, as quais, juntamente com os dados armazenados no *data warehouse*, são utilizadas para determinar o novo conteúdo dessa base de dados. Segundo Gupta e Mumick [GM95], técnicas de atualização incremental baseiam-se na heurística de que somente uma parte da visão altera em resposta a alterações nas relações base. No entanto, se uma relação base for excluída e a visão do *data warehouse* evoluir para uma relação vazia, pode ser mais barato recalcular a visão novamente do que atualizá-la incrementalmente.

O processo de atualização incremental inicia-se com a detecção e a propagação de alterações nos dados dos provedores de informação que participam do ambiente. Rotinas de detecção de alterações devem identificar e extrair somente os dados relevantes ao ambiente, diminuindo assim o volume de dados a ser manipulado.

A detecção de alterações nos dados operacionais é realizada de acordo com as facilidades oferecidas pelos provedores de informação. Assim, rotinas de detecção de alterações devem considerar como o provedor indica que seus dados foram alterados e/ou que novos dados foram adicionados. Dentre as principais técnicas utilizadas, pode-se citar: (i) varredura de marcadores de tempo (*timestamps*); (ii) utilização de um arquivo *delta*; (iii) utilização do arquivo de *log*; (iv) alteração do código fonte; (v) comparação de versões sucessivas de instantâneos (*snapshots*); e (vi) uso de gatilhos [Inm96, LGM96].

A primeira técnica consiste na varredura dos marcadores de tempo associados aos dados. Nessa técnica, dados com data anterior à ultima atualização do *data warehouse* são desconsiderados. Já as segunda e terceira técnicas limitam os dados a serem extraídos através da varredura de um arquivo *delta* e do arquivo de *log*, respectivamente. Ambos arquivo *delta* e

arquivo de *log* possuem basicamente as mesmas informações, são criados pelas respectivas aplicações/sistemas e refletem somente as alterações que ocorreram nos dados. A quarta técnica, por sua vez, consiste na alteração do código fonte das aplicações para incorporar o envio de modificações ao ambiente de *data warehousing*. Na técnica de comparação de instantâneos, dois instantâneos sucessivos (um representando os dados operacionais no momento da extração de dados anterior e outro representando os dados no momento da extração atual) são comparados visando-se a identificação de diferenças específicas entre os seus dados. Em especial, o trabalho de Labio *et al*. [LGM96] apresenta algoritmos para a comparação de arquivos contendo conjuntos de registros que possuam chave e permitam as operações de inserção, de remoção e de atualização, ao passo que o trabalho de Chawathe *et al*. [CRGMW96] estuda o problema de detectar e de representar alterações significativas entre duas versões de dados estruturados hierarquicamente. Esse último trabalho também oferece suporte às operações para mover e copiar, em adição às de inserção, de remoção e de atualização. Por fim, SGBD relacionais que incorporam características de bancos de dados ativos podem utilizar gatilhos tanto para a detecção quanto para a notificação automática de alterações nos seus dados.

As técnicas acima discutidas possuem limitações e desvantagens, o que faz com que o processo de detecção de alterações nos dados dos provedores de informação represente um grande desafio para o ambiente de *data warehousing*. Por exemplo, somente poucos dados operacionais possuem marcadores de tempo a ele associados, somente poucas aplicações utilizam arquivos *delta* e somente poucos provedores oferecem recursos de bancos de dados ativos. Com relação ao uso do arquivo de *log*, muitas barreiras devem ser enfrentadas. Geralmente esse arquivo somente pode ser acessado utilizando-se o privilégio de administrador de banco de dados. Ademais, seu formato interno é construído para propósitos específicos do sistema, podendo ser difícil de ser entendido e conter informações adicionais desnecessárias. Por fim, o *log* é considerado um componente essencial no processo de recuperação de falhas, sendo protegido pelo sistema. Já o problema da técnica de comparação de versões sucessivas de instantâneos está relacionado ao fato de que, à medida que o volume de dados do provedor de informação cresce, comparações cada vez maiores precisam ser realizadas.

Uma vez detectada e propagadas as alterações relevantes, estas devem ser incrementalmente carregadas no *data warehouse*, de acordo com as especificações do ambiente. A seção 2.5.2 aborda vários trabalhos e otimizações neste sentido, além de definir o conceito de visão materializada e detalhar o relacionamento deste conceito com os dados armazenados no *data warehouse*.

2.2.2.3 Expiração dos Dados do *Data Warehouse*

Como discutido na seção 2.1.1, o volume de dados armazenado em um *data warehouse* é geralmente muito grande, variando de *gigabytes* a *terabytes* de tamanho. Da mesma forma que as rotinas de extração dos dados dos provedores de informação enfocam a limitação dos dados a serem extraídos destes provedores, o processo de expiração está relacionado com a remoção de dados do *data warehouse*, visando uma diminuição do volume de dados armazenado nessa base de dados. Como conseqüência, o ciclo de vida do dado, incluindo sua completa eliminação ou arquivamento final, é uma parte ativa e importante da fase de manutenção do ambiente de *data warehousing*.

De maneira geral, dados são expirados do *data warehouse* quando atingem o limite de tempo no qual tornam-se inválidos (dados são considerados "velhos"), quando não são mais relevantes ou necessários para o ambiente, ou quando o espaço de armazenamento é insuficiente. Como pode ser observado através destes fatores, a expiração dos dados representa um processo

completamente diferente do de atualização dos dados do *data warehouse*, no qual ocorre a propagação de remoções dos dados dos provedores de informação.

Nem sempre os dados expirados são realmente eliminados do ambiente. Muitas vezes esses dados são mantidos em níveis de agregação superiores ou transferidos para outros meios de armazenamento, tal como fita. Quatro diferentes políticas de expiração são apresentadas no trabalho de Wu e Buchmann [WB97]: eliminação completa, eliminação seletiva, arquivamento e eliminação seletiva com arquivamento.

A primeira política, como o próprio nome diz, está relacionada com a eliminação completa dos dados expirados, tanto do nível inferior quanto dos demais níveis de agregação. Os metadados relacionados a estes dados, por outro lado, permanecem armazenados no ambiente de *data warehousing*, permitindo a identificação dos dados operacionais correspondentes caso tais dados precisem ser referenciados novamente. Esta política não é adequada em situações nas quais os dados estão sendo expirados por falta de espaço de armazenamento ou nas quais consultas dos usuários de SSD esperam que níveis com alto grau de agregação englobem informações antigas.

Na política de eliminação seletiva, os dados expirados são eliminados como uma função da capacidade de armazenamento, da freqüência de acesso e das necessidades de desempenho. Por exemplo, apenas os dados expirados do nível inferior da hierarquia de agregação são mantidos, enquanto que os dados relativos aos demais níveis são completamente eliminados, uma vez que os níveis intermediários e superiores são determinados a partir de agregações do nível inferior. No entanto, de acordo com as consultas dos usuários de SSD e o volume de dados armazenado em cada um dos níveis, pode ser mais interessante armazenar somente os dados relativos aos níveis mais superiores, enquanto que os de níveis mais inferiores são eliminados do ambiente. Neste caso, um modelo matemático deve ser utilizado para se determinar quais dados devem ser eliminados ou mantidos no sistema.

A terceira técnica consiste no arquivamento do dado expirado em algum outro dispositivo de armazenamento, antes desse ser completamente eliminado do ambiente. Basicamente, apenas os dados mais detalhados relativos ao nível inferior são armazenados, desde que as agregações podem ser derivadas destes. Esta política pode ser utilizada quando os provedores de informação não são capazes de oferecer dados históricos e quando os dados históricos serão acessados apenas esporadicamente. Um dado deve ser arquivado somente se possuir alguma probabilidade de acesso, mesmo que esta seja pequena. Quando sua probabilidade de acesso aproximar-se a zero, então este deve ser realmente removido.

Por fim, a técnica de eliminação seletiva com arquivamento representa uma combinação das duas técnicas anteriores, e é indicada para casos nos quais os provedores de informação não são históricos e a freqüência de acesso aos dados expirados é relativamente alta. Neste caso, os dados detalhados são arquivados antes de serem eliminados e alguns dados agregados correspondentes a estes dados arquivados continuam presentes no *data warehouse*.

Uma variação destas políticas é apresentada no trabalho de Garcia-Molina *et al*. [GMLY98], que propõe um método para a eliminação de dados expirados do nível inferior, sem que os demais níveis de agregação sejam afetados. Este método é baseado na premissa de que alguns dados podem ser removidos sem problemas, ao passo que outros devem ser mantidos no sistema porque podem ser indispensáveis ao processo de atualização do *data warehouse*. No entanto, os dados expirados mantidos no sistema tornam-se inacessíveis a consultas dos usuários de SSD. Como conseqüência, as agregações permanecem inalteradas pela expiração dos dados do nível inferior e, em adição, após a expiração, ainda existe informação suficiente para que estas agregações permaneçam consistentes com relação a alterações nos provedores.

2.2.3 Componente de Análise e Consulta

O componente de análise e consulta garante o acesso às informações armazenadas no *data warehouse* aos usuários de SSD e aos programas aplicativos que participam do ambiente de *data warehousing*. Oferece, desta forma, funcionalidades relacionadas à consulta e à análise dos dados armazenados, incluindo a habilidade de se determinar a origem dos dados sendo examinados. Essa seção descreve cada uma destas funcionalidades em maior nível de detalhe.

2.2.3.1 Consultando e Analisando o *Data Warehouse* através de Ferramentas

O principal propósito de um ambiente de *data warehousing* consiste em disponibilizar informação integrada aos usuários de SSD para a tomada de decisão estratégica. Tais usuários interagem com o ambiente através de ferramentas dedicadas à análise e consulta dos dados, as quais devem oferecer facilidades de navegação e de visualização. Em especial, estas ferramentas devem permitir que informações relevantes ao contexto de tomada de decisão sejam derivadas a partir da detecção de análise de tendências, da monitoração de problemas e de análises competitiva e comparativa.

Dentre os principais tipos de ferramentas de acesso existentes, pode-se citar: ferramentas de consulta gerenciáveis e geradores de relatório, ferramentas para sistemas de informações executivas, ferramentas OLAP e ferramentas de mineração de dados [CR97].

Ferramentas de consulta gerenciáveis e geradores de relatório são os tipos mais simples de ferramentas e, em geral, não são voltadas especificamente ao ambiente de *data warehousing*. Geradores de relatório, como o próprio nome diz, têm como principal objetivo produzir relatórios periódicos. Já ferramentas de consulta gerenciáveis oferecem aos usuários visões de negócio específicas ao domínio dos dados armazenados e permitem que estes usuários realizem consultas independentemente da estrutura e/ou da linguagem de consulta oferecida pelo banco de dados. Por exemplo, uma ferramenta deste tipo poderia permitir a criação de comandos SQL através da utilização do *mouse*. A saída destas ferramentas é geralmente na forma de um relatório.

Ferramentas para sistemas de informações executivas permitem que aplicações de suporte à decisão gráficas e customizadas sejam desenvolvidas, oferecendo aos usuários de SSD uma visão de alto nível do negócio. Esse tipo de ferramenta é caracterizado por utilizar uma visualização gráfica simplificada, representando exceções a atividades normais de negócio ou a regras através de diferentes cores. Tais ferramentas geralmente apresentam capacidades analíticas limitadas, uma vez que são projetadas visando-se o oferecimento de facilidade de uso tanto para a consulta quanto para a análise das informações.

Por outro lado, ferramentas OLAP são caracterizadas por permitir que usuários de SSD sofisticados analisem os dados usando visões multidimensionais complexas e elaboradas, e por oferecer navegação facilitada através dessas visões. Assim, os usuários de SSD podem analisar os dados sob diferentes perspectivas e/ou determinar tendências através da navegação entre diferentes níveis de hierarquias de agregação. Tais ferramentas apresentam os dados de acordo com o modelo multidimensional (seção 2.4.1), independentemente da forma na qual eles estão realmente armazenados. Aplicações típicas de negócio para essas ferramentas incluem o desempenho de vendas de um determinado produto e a sua lucratividade, a efetividade de um programa de vendas ou a campanha de *marketing* e o planejamento de vendas.

De maneira geral, o componente de análise e consulta possui duas funcionalidades básicas: facilitar o acesso aos dados do *data warehouse* e permitir que informações, padrões e tendências de negócio "escondidas" nestes dados sejam descobertas. Em oposição aos tipos de ferramentas acima descritos, os quais enfocam a primeira funcionalidade, ferramentas de mineração de dados permitem que os usuários de SSD explorem e infiram informação útil a partir dos dados, identificando relacionamentos desconhecidos. Mineração dos dados [AMS+96, Fre98] é uma área que se encontra em plena fase de crescimento, podendo ser considerada uma interseção de diversas outras áreas, como banco de dados, inteligência artificial e estatística. Segundo [Fay98], mineração de dados refere-se à aplicação de algoritmos específicos para a extração de padrões dos dados, a qual consiste em uma das etapas do processo de descobrimento de informação (conhecido como KDD – *Knowledge Discovery in Databases*). A qualidade do conhecimento descoberto por um algoritmo é altamente dependente da aplicação e tem um aspecto subjetivo inerente.

Independentemente da ferramenta de acesso aos dados utilizada, um fator primordial a ser considerado refere-se à visualização dos resultados obtidos. Técnicas de visualização dos dados devem determinar a melhor forma de se exibir relacionamentos e padrões complexos em um monitor bidimensional, de modo que o problema inteiro e/ou a solução sejam claramente visíveis. Por exemplo, padrões podem ser muito mais facilmente detectados se forem expressos graficamente, melhor do que através de simples tabelas. Em especial, técnicas de visualização devem oferecer interação com os usuários de SSD, os quais devem ser capazes de alterar tanto o tipo de informação sendo analisada quanto o método de apresentação sendo utilizado (como histogramas, mapas hierárquicos e gráficos de dispersão).

Uma lista de ferramentas voltadas à análise e consulta dos dados do *data warehouse* pode ser encontrada em [Gre]. Por outro lado, Pendse e Creeth [PC] apresentam informações sobre aproximadamente 30 produtos OLAP, além de relatórios atualizados contendo novidades, dados sobre o mercado e estudos de caso relacionados a esse tema.

2.2.3.2 Identificando a Origem dos Dados Analisados

Uma questão de grande interesse em aplicações OLAP e de mineração dos dados é a habilidade de se determinar o conjunto de dados origem que produz pedaços específicos da informação. Por exemplo, um usuário de SSD pode desejar identificar as origens dos dados suspeitos ou incorretos gerados em uma visão de alto nível, com o intuito de verificar a integridade e a confiabilidade dos provedores de informação ou até mesmo de corrigir os dados correspondentes desses provedores.

O problema de identificar o conjunto exato de itens de dado das relações base que produzem um determinado item de dado de uma visão materializada e o processo pelo qual ele foi produzido é conhecido como problema de linhagem dos dados. Segundo Cui e Widom [CW00], alguns sistemas de *data warehousing* comerciais disponibilizam a linhagem de dados em nível de esquema, ou oferecem facilidades específicas para a manipulação das visões multidimensionais (tal como *drill-down*). Em contrapartida, o trabalho de Cui e Widom, descrito em maior nível de detalhamento em [CWW97], propõe algoritmos para a determinação da linhagem de dados em nível de instância.

Os algoritmos consideram que tanto as visões materializadas quanto os itens de dado base estão armazenados de acordo com o modelo relacional [EN00]. Em adição, as visões materializadas podem ser complexas, definidas em termos dos operadores da álgebra relacional de seleção, de projeção, de junção, de diferença entre conjuntos e de união de conjuntos, além de

um operador adicional de agregação. Essas visões podem tanto armazenar valores de instâncias duplicados quanto possuir uma semântica de conjunto (sem duplicações). Baseado na definição da visão e nos dados base, além de algumas informações auxiliares adicionais, os algoritmos determinam os dados que derivaram o dado da visão materializada.

A definição da visão é expressa na forma de uma árvore de consulta, na qual os nós folhas representam as relações base e os nós internos representam os operadores. Inicialmente, os algoritmos transformam a árvore de consulta da visão em uma forma canônica composta de seqüências de segmentos. Tais segmentos podem ser de dois tipos: segmentos AUSPJ (descritos em termos dos operadores de agregação, de união, de projeção, de seleção e de junção) e segmentos D (descritos em termos do operador diferença). Se a visão é definida como um único segmento, a linhagem dos dados definidos por este segmento é determinada através da execução de consultas relacionais baseadas na forma canônica sobre as relações dos provedores, chamadas de consultas de descoberta, as quais são parametrizadas pelos dados sendo analisados. Neste caso, resultados intermediários não são necessários. Em contrapartida, caso a visão seja especificada por vários segmentos, os algoritmos dividem estes segmentos, definem uma visão intermediária para cada segmento e percorrem recursivamente a hierarquia de visões intermediárias de acordo com a abordagem *top-down*. Em cada nível da hierarquia de visões intermediárias, consultas de descoberta para um segmento único são utilizadas para determinar a linhagem dos dados correntes sendo analisados em relação às visões dos níveis subjacentes até atingir as relações base. Pode-se observar que, neste caso, visões intermediárias são necessárias.

Em geral, as visões intermediárias podem ser tanto armazenadas quanto calculadas a partir das tabelas base quando necessário. Dentro do contexto de *data warehousing*, os algoritmos acima descritos optam pela materialização dessas visões auxiliares, uma vez que os mecanismos de atualização incremental geralmente materializam visões intermediárias, as quais são as mesmas necessárias ao problema de linhagem dos dados. Sob este aspecto, o problema de linhagem de dados está diretamente relacionado ao processo de atualização dos dados do *data warehouse*, uma vez que a consistência das visões intermediárias com relação aos dados dos provedores deve ser garantida. Visando-se um aumento de desempenho, os algoritmos propostos também armazenam cópias das tabelas dos provedores.

2.2.4 *Data Marts*

Um *data mart* consiste na implementação de um *data warehouse* no qual o escopo do dado é limitado, quando comparado ao *data warehouse* propriamente dito. Entretanto, os dados armazenados em *data marts* compartilham as mesmas características que os dados do *data warehouse*, ou seja, são orientados a assunto, integrados, não-voláteis e históricos, além de serem organizados segundo diferentes níveis de agregação.

Um *data mart* pode ser tanto definido como um subconjunto dos dados do *data warehouse* quanto considerado uma política no projeto de construção de um *data warehouse* corporativo.

Data marts podem ser considerados subconjuntos dos dados do *data warehouse* uma vez que possuem cópias replicadas de porções de dados dessa base de dados e são projetados para atender às necessidades específicas de grupos de usuários de SSD dedicados. Por exemplo, um *data mart* departamental pode conter dados departamentais altamente agregados, os quais são utilizados pelos usuários de SSD do departamento em questão para a tomada de decisão local. Tais *data marts* são chamados de *data marts* dependentes, pois independentemente da tecnologia utilizada em suas implementações e do número de *data marts* existentes, grupos de usuários distintos acessam visões de informação derivadas da mesma visão integrada dos dados. Segundo

Moeller [Moe01], a existência de *data warehouses* dependentes em uma corporação representa uma conseqüência natural do volume crescente de informações armazenadas no *data warehouse*.

A criação de *data marts* como subconjuntos do *data warehouse* está relacionada a questões de desempenho, de simplicidade de entendimento e de descentralização de acesso. Como o número de usuários de SSD concorrentes e o volume de dados armazenado em um *data mart* são usualmente menores do que o número de usuários de SSD concorrentes e o volume de dados armazenados no *data warehouse*, o tempo de resposta deste componente tende a ser menor. Em adição, o escopo limitado do *data mart* simplifica o entendimento de seu projeto e, conseqüentemente, a sua manutenção. Por fim, uma vez que grupos de usuários são capazes de manipular de forma autônoma as suas porções de dados de interesse, ocorre uma descentralização de acesso aos dados do *data warehouse*.

Por outro lado, em uma grande corporação, *data marts* tendem a ser utilizados como uma política de construção evolucionária do *data warehouse*. Uma vez que o processo de construção de um *data warehouse* sobre toda a organização é longo e complexo e os custos envolvidos são altos, *data marts* são construídos paulatinamente e, à medida que estes se consolidam, inicia-se a construção do *data warehouse* global. De maneira geral, tais *data marts* representam soluções fragmentadas de porções de negócio da empresa, e são chamados de *data marts* independentes. Diferentemente dos *data marts* dependentes, os quais usam o *data warehouse* como fonte de dados, *data marts* independentes obtêm seus dados diretamente a partir dos provedores de informação.

A utilização de *data marts* independentes tende, inicialmente, a reduzir problemas financeiros, uma vez que a construção desses *data marts* exige recursos monetários inferiores do que os despendidos com a construção de um *data warehouse* corporativo e que usuários de SSD são capazes de reconhecer o valor e a potencialidade da solução de *data warehousing* em um período menor de tempo. Entretanto, em longo prazo, a criação de *data marts* independentes pode conduzir a problemas de integração e de escalabilidade, caso um modelo de negócio completo não seja desenvolvido. Cada *data mart* independente pode assumir formas diferentes de consolidar seus dados. Como conseqüência, os dados através dos diversos *data marts* podem ser inconsistentes. Problemas de escalabilidade, por sua vez, ocorrem em situações nas quais o *data mart* inicial de tamanho pequeno e projeto simplificado tende a crescer, tanto no volume de dados armazenado quanto no número de usuários concorrentes que o utilizam.

2.2.5 Metadados

Dados de nível mais baixo podem ser descritos por dados de nível mais alto graças ao conceito de metadados. Metadados consistem em uma abstração dos dados, e permitem que dados armazenados nos mais diferentes formatos tenham significado. Por exemplo, metadados associados a uma seqüência de 0's e 1's devem indicar se tais caracteres representam palavras, ou números, ou dados estatísticos sobre vendas da empresa, ou ainda informações sobre a distribuição populacional do país [APT96].

O armazenamento de metadados no ambiente de *data warehousing* possui um nível de importância elevado, uma vez que os usuários de SSD necessitam conhecer a estrutura e o significado dos dados no processo de busca por fatos não usuais sobre o negócio. Em outras palavras, metadados constituem-se no principal recurso para a administração dos dados nesse ambiente, sendo utilizados durante a criação, a manutenção, o acesso e o gerenciamento do *data warehouse*. Metadados possuem característica temporal, desde que refletem tanto dados

históricos mantidos no *data warehouse* quanto alterações estruturais do *data warehouse* ao longo do tempo.

Em geral, uma grande variedade de metadados precisa ser armazenada, visando a utilização efetiva do ambiente de *data warehousing*. Wu e Buchmann [WB97] dividem os metadados em três categorias:

- **metadados administrativos**: contêm informações relacionadas à construção e à utilização do *data warehousing*, tais como os esquemas dos provedores de informação e do *data warehouse*, além dos mapeamentos existentes entre os diversos esquemas; regras de extração, de tradução, de limpeza e de atualização dos dados, em adição às regras de mapeamento utilizadas para a solução de problemas de heterogeneidade existentes entre os dados dos diversos provedores de informação que participam do ambiente; especificações sobre grupos de usuários e privilégios a eles associados, incluindo políticas de controle de acesso, autorização e perfis; ferramentas de integração e manutenção, e regras associadas aos processos envolvidos; ferramentas de análise e consulta; consultas, agregações e relatórios pré-definidos;
- **metadados específicos da aplicação:** incluem um conjunto de terminologias específicas ao domínio da aplicação, além de restrições da aplicação e outras políticas; e
- **metadados de auditoria:** mantêm informações relacionadas à linhagem dos dados, à geração de relatórios de erros, às ferramentas de auditoria empregadas e às estatísticas de utilização do ambiente de *data warehousing*, incluindo dados sobre a freqüência das consultas, os custos para se processar uma determinada consulta, o tipo de acesso aos dados e o desempenho do sistema.

Já Campos e Rocha Filho [CR97] argumentam que existem, de um modo geral, três camadas de metadados em um ambiente de *data warehousing*. A primeira camada, denominada de **metadados operacionais**, define a estrutura dos dados mantidos pelos provedores de informação operacionais. **Metadados centrais ao *data warehouse*** representam a segunda camada, e englobam mapeamentos de como os dados dos provedores de informação são transformados, definições de agregados e relações existentes entre os diferentes níveis de agregação, dentre outros. A última camada, **metadados do nível do usuário**, mantém metadados que mapeiam os metadados do *data warehouse* para conceitos que sejam familiares aos usuários de SSD.

Por outro lado, Campos [Cam99] e Vaduva e Dittrich [VD01] também distinguem metadados de acordo com o seu uso em **metadados de negócio**, requeridos principalmente por usuários de SSD e **metadados técnicos**, produzidos e utilizados pelos administradores do banco de dados ou pelos componentes de *software* que participam do ambiente de *data warehousing*. A categoria de **metadados de negócio** contém documentação específica dos usuários de SSD, dicionários, conceitos e terminologias do negócio, detalhes sobre consultas predefinidas e relatórios, dentre outras informações. Em contrapartida, **metadados técnicos** incluem: definições de esquema e especificações de configuração; informações sobre o armazenamento físico; direitos de acesso; e especificações executáveis tais como regras de transformação e de integração.

Ainda outra distinção refere-se a **metadados descritivos** e **metadados transformacionais** [VD01]. Enquanto que **metadados descritivos** incluem informações relacionadas à estrutura dos provedores de informação, do *data warehouse* e dos *data marts* que compõem o ambiente de *data warehousing*, **metadados transformacionais** incluem informações associadas ao processamento de dados, como exemplo as regras empregadas durante os processos de extração, de tradução, de limpeza, de integração e de agregação dos dados.

Independentemente da taxonomia utilizada, Kimball [Kim98a] explora de forma exaustiva as informações que devem ser armazenadas sobre essa enorme variedade de metadados, considerando desde o processo de leitura dos dados dos provedores de informação até o armazenamento desses dados no *data warehouse*. Por exemplo, informações sobre as agregações incluem as suas definições, os *logs* de modificação dessas agregações e as suas estatísticas de utilização, além de quais agregações são potenciais.

Em um ambiente de *data warehousing*, os metadados são armazenados em um ou mais repositórios de metadados e manipulados por módulos gerenciadores de metadados. Mais detalhadamente, um repositório consiste em um banco de dados voltado ao armazenamento e à recuperação dos metadados. Repositórios representam os componentes integradores da arquitetura do ambiente, e são (logicamente) independentes do *data warehouse*, ainda que a mesma plataforma de SGBD seja utilizada. Já os módulos gerenciadores devem coordenar a utilização dos repositórios de metadados, incluindo o acesso, a forma de armazenamento dos dados e a sua manutenção. Outras funcionalidades importantes a serem oferecidas por módulos gerenciadores de metadados incluem, por exemplo: (i) prover mecanismos de controle de versões que suportem a característica temporal dos metadados; (ii) prover mecanismos de notificação que propaguem todas as alterações nos metadados a ferramentas e usuários interessados; (iii) possibilitar o estabelecimento de relacionamentos entre as informações armazenadas visando-se análise de impacto; e (iv) facilitar o fluxo de metadados entre diferentes repositórios.

Existe atualmente um grande número de ferramentas comerciais especificamente voltadas ao gerenciamento de metadados, além de diversas outras funcionalidades relacionadas à utilização desse repositório incorporadas em ferramentas que auxiliam outros processos do carregamento dos dados. Em [APT96], diversas ferramentas existentes são classificadas com relação à sua principal funcionalidade, ao passo que em [Gre] uma lista de ferramentas *stand-alone* voltadas ao gerenciamento de metadados pode ser encontrada.

O gerenciamento de metadados representa uma área de pesquisa muito pouco desenvolvida, podendo até mesmo ser considerada imatura [Cam99]. Por um lado, existe um grande espectro de metadados, os quais devem ser capturados, armazenados e gerenciados consistentemente visando-se tanto minimizar esforços no desenvolvimento, na manutenção e na administração do *data warehousing* quanto melhorar a extração efetiva da informação a partir do dado. Por outro lado, existe uma grande quantidade de produtos comerciais que alegam gerenciar metadados, mas que possuem características diferentes entre si e, acima de tudo, mantêm seus próprios metadados em formatos proprietários. Assim, um desafio de extrema importância relacionado ao gerenciamento de metadados refere-se ao compartilhamento e à troca de metadados de forma que estes possam ser uniformemente acessíveis pelos usuários de SSD e pelas ferramentas do ambiente de *data warehousing*.

Na prática, o principal problema de integração é a falta de um padrão, o qual produtos comerciais de diferentes fabricantes possam seguir. Uma iniciativa que tem sido adotada atualmente neste sentido diz respeito à definição, à especificação e à implementação de formatos de padrão de troca de metadados e de seus mecanismos de suporte por grupos de vendedores. Dois grupos têm merecido grande destaque: Metadata Coalition (MDC) e Object Management Group (OMG). MDC, atualmente liderado pela Microsoft, é o criador do padrão *Metadata Interchange Specification* (MDIS). Como resultado dessa liderança, MDIS tem sido integrado com o padrão *Open Information Model* (OIM). Mais detalhadamente, OIM enfoca o compartilhamento e o reuso de metadados através do oferecimento de um modelo formal de descrição de tipos de metadados. OIM utiliza como base outros padrões de mercado: UML (*Unified Modeling Language*) como a linguagem formal de especificação, XML (*eXtensible Markup Language*) como o formato para a troca de informações entre repositórios baseados em

OIM, e SQL como a linguagem de extração. Por outro lado, OMG possui como membros especialistas tais como IBM, Oracle e Unisys, e trabalha na especificação do padrão para intercâmbio *Common Warehouse Metadata* (CWM). Assim como OIM, CWM também é fundamentado em outros padrões de mercado, a saber: UML, MOF (*Meta Object Facility*), e XMI (*XML Metadata Interchange*), um padrão baseado em XML para a troca de metadados originado pelo OMG. Uma análise comparativa entre os padrões OIM e CWM pode ser encontrada em [VVS00]. Já em [MO98], [SM00] e [Cam00] podem ser encontradas descrições dos padrões MDIS, OIM e MOF, respectivamente.

Como pode ser observado, a necessidade da criação de um padrão para a representação e a troca de metadados em ambientes de *data warehousing* e a existência de diferentes grupos de especialistas propondo seus próprios padrões representam metas antagônicas. Neste sentido, MDC e OMG juntaram-se recentemente através de um relacionamento corporativo para definir um consenso em padrões [Met99b]. A proposta consiste na unificação dos padrões OIM e CWM. No entanto, segundo Vetterli *et al*. [VVS00], apesar das similaridades existentes entre esses dois padrões, existem diferenças significativas que fazem com que a unificação destes seja uma tarefa difícil.

2.3 *Data Warehousing* Virtual

A principal diferença existente entre um ambiente de *data warehousing* convencional (seção 2.2) e um ambiente de *data warehousing* virtual é a presença física ou não do *data warehouse*. Em um ambiente de *data warehousing* convencional, dados de interesse dos provedores de informação são extraídos, traduzidos, filtrados quando necessário, integrados e finalmente armazenados fisicamente no *data warehouse*. Em contrapartida, um ambiente de *data warehousing* virtual apenas oferece aos usuários de SSD uma visão lógica ou virtual dos dados armazenados nos provedores de informação [CR97, Ken00]. Mais especificamente, nesse último ambiente não existe um *data warehouse* físico que armazena os dados obtidos do ambiente operacional.

Assim, usuários de SSD do ambiente de *data warehousing* virtual podem acessar os dados desejados diretamente a partir dos provedores de informação. Isto é geralmente realizado através do uso de ferramentas de análise e consulta, as quais pesquisam os dados operacionais e exibem os resultados aos usuários como se estes resultados tivessem sido obtidos a partir de um *data warehouse* consolidado. Entretanto, as consultas submetidas ao *data warehousing* virtual são constantemente executadas em dados não trabalhados e não estruturados, os quais não foram eficientemente projetados para consulta ou análise. Em especial, essas ferramentas escondem de seus usuários a complexidade do ambiente operacional e, de acordo com as suas funcionalidades, podem até oferecer visões multidimensionais dos dados.

A Figura 2.3 ilustra a arquitetura básica de um ambiente de *data warehousing* virtual. Nesse ambiente, alguns componentes importantes de um ambiente de *data warehousing* convencional não estão presentes:

- não existem bases de dados específicas (ou seja, *data warehouse* e/ou *data marts*) que integram os dados extraídos, traduzidos e filtrados oriundos do ambiente operacional;
- não existe um componente de integração e manutenção que centraliza os processos executados durante a atividade de carregamento dos dados; e
- não existe um repositório de metadados específico que armazena os metadados referentes ao ambiente de *data warehousing*.

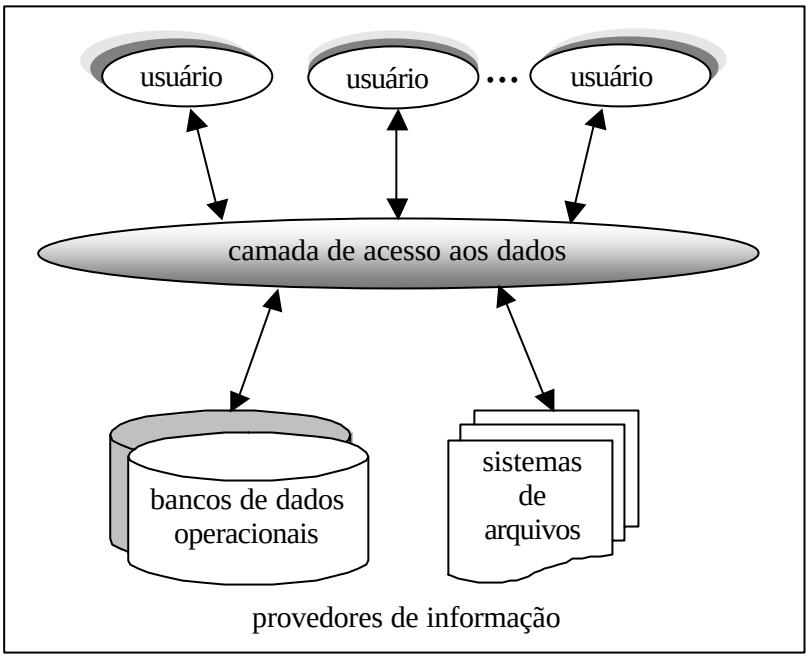

Figura 2.3 Arquitetura básica de um ambiente de *data warehousing* virtual.

Desde que ambientes de *data warehousing* virtuais utilizam os provedores de informação como servidores de banco de dados, pouco investimento monetário precisa ser despendido em *software* e *hardware* adicionais. Em particular, o principal investimento monetário nestes ambientes refere-se às ferramentas de análise e consulta que garantem o acesso aos dados operacionais. Ambientes de *data warehousing* virtuais também são caracterizados por sua simplicidade. Em geral, esses ambientes não oferecem rotinas complexas voltadas à extração, à tradução, à limpeza e à integração dos dados. Além disto, uma vez que o *data warehouse* não está presente na arquitetura desses ambientes, não existe a necessidade de se realizar a atualização periódica dos dados dessa base de dados. Conseqüentemente, o tempo gasto na implementação de um ambiente de *data warehousing* virtual é menor, quando comparado com o tempo de implementação de um ambiente de *data warehousing* convencional [Mim99].

Entretanto, ambientes de *data warehousing* virtuais apresentam diversas desvantagens, as quais contribuem para que o uso desses ambientes seja indicado somente em poucas situações. A seção 2.3.1 destaca as desvantagens de ambientes de *data warehousing* virtuais, com base nas funcionalidades oferecidas por ambientes de *data warehousing* convencionais. Já a seção 2.3.2 identifica situações nas quais pode ser interessante utilizar um *data warehousing* virtual.

2.3.1 *Data Warehousing* Virtual *versus Data Warehousing* Convencional

Ambientes de *data warehousing* virtuais são muito menos robustos do que ambientes de *data warehousing* convencionais. Essa seção compara essas duas abordagens, destacando as desvantagens e as limitações apresentadas por *data warehousing* virtuais. Tal comparação é realizada em termos: (i) das funcionalidades analíticas oferecidas; (ii) da qualidade dos dados manipulados; (iii) da centralização do processo de integração dos dados operacionais; (iv) do desempenho no suporte a consultas OLAP; e (v) da separação existente entre os ambientes operacional e informacional da corporação.

Em primeiro lugar, as funcionalidades analíticas geralmente oferecidas por ambientes de *data warehousing* virtuais são muito mais simples e limitadas do que as funcionalidades analíticas oferecidas por ambientes de *data warehousing* convencionais. Uma vez que os dados manipulados em um ambiente de *data warehousing* virtual são obtidos diretamente a partir do ambiente operacional, estes dados são altamente voláteis, além de não serem orientados a assunto. Ademais, um *data warehousing* virtual raramente manipula dados históricos [Tha99]. Como resultado, análises comparativas complexas e análises de tendência são fortemente prejudicadas, quando não impossíveis de serem realizadas.

Outra desvantagem apresentada por ambientes de *data warehousing* virtuais, quando comparados a ambientes de *data warehousing* convencionais, refere-se à qualidade dos dados. Por um lado, o *data warehouse* mantém os dados sobre o negócio com alta qualidade, principalmente devido ao processo criterioso de limpeza aplicado aos dados durante a atividade de carregamento. Por outro lado, a qualidade dos dados de *data warehousing* virtuais é altamente dependente da qualidade dos dados dos provedores de informação. No entanto, provedores de informação podem armazenar dados duvidosos e/ou incorretos. Em especial, funcionalidades relacionadas à limpeza dos dados são geralmente ausentes em ambientes de *data warehousing* virtuais [Gri01]. Isto significa que a exatidão dos resultados produzidos por esses ambientes em resposta às consultas dos usuários de SSD pode ser questionada.

Não somente a qualidade dos dados é crítica em ambientes de *data warehousing* virtuais, mas também as funcionalidades relacionadas ao processo de integração dos dados operacionais. Como discutido na seção 2.2.2, em *data warehousing* convencionais, estas funcionalidades são oferecidas pelo componente de integração e manutenção. Desde que este componente não é presente na arquitetura de um ambiente de *data warehousing* virtual, ferramentas de análise e consulta desse ambiente enfrentam o desafio de manipular os conflitos de nome, semânticos e estruturais existentes entre os dados dos provedores de informação. É evidente que tais conflitos podem ser resolvidos individualmente por cada uma das ferramentas envolvidas, porém isto gera redundância de esforços e possíveis incompatibilidades.

Em ambientes de *data warehousing* convencionais, os dados do *data warehouse* são organizados visando-se um processamento eficiente de consultas OLAP. Ou seja, um *data warehouse* armazena tanto dados detalhados quanto dados agregados. Em contrapartida, ambientes de *data warehousing* virtuais manipulam apenas dados detalhados, os quais são acessados diretamente a partir dos provedores de informação. Como destacado no início da seção 2.3, os dados operacionais não são estruturados visando-se um bom desempenho no processamento de consultas OLAP. Conseqüentemente, em ambientes de *data warehousing* virtuais, os custos de entrada/saída e de processamento são significativamente maiores, conduzindo a tempos de reposta menos satisfatórios aos usuários de SSD.

Além de comprometerem o desempenho no suporte a consultas OLAP, *data warehousing* virtuais também causam um impacto negativo no desempenho do ambiente operacional. Tal situação é decorrente do fato que em *data warehousing* virtuais não existe uma separação entre os ambientes operacional e informacional da corporação, como ocorre em *data warehousing* convencionais. Mais detalhadamente, consultas OLAP competem com transações OLTP pelos mesmos recursos em *data warehousing* virtuais. Assim, o ambiente operacional também passa a oferecer tempos de reposta menos satisfatórios a seus usuários. Em particular, quanto maior a quantidade de dados que uma corporação manipula, maior a necessidade de se realizar uma separação entre os ambientes operacional e informacional da corporação.

2.3.2 Aplicabilidade do *Data Warehousing* Virtual

Um ambiente de *data warehousing* virtual não oferece as mesmas funcionalidades que um ambiente de *data warehousing* convencional. Isto é decorrente do fato que na arquitetura de um *data warehousing* virtual não estão presentes componentes importantes tais como o *data warehouse* propriamente dito, o repositório de metadados e o componente de integração e manutenção. Assim, *data warehousing* virtuais devem ser utilizados apenas em situações particulares.

A utilização de um ambiente de *data warehousing* virtual pode ser interessante em situações nas quais existe uma demanda pouco freqüente pelos dados. Em especial, esta demanda deve ser composta principalmente por consultas simples, que não afetem de forma demasiadamente negativa o desempenho do ambiente operacional. É importante destacar, entretanto, que a demanda de consultas submetidas a um ambiente de *data warehousing* tende a crescer à medida que os usuários de SSD conscientizam-se da importância desse ambiente. Além disto, usuários de SSD geralmente estão interessados em análises complexas e elaboradas dos dados.

Um ambiente de *data warehousing* virtual também pode ser utilizado como uma estratégia preliminar até que um *data warehousing* convencional seja desenvolvido [Alu96]. Esta situação, em particular, representa a situação mais indicada para o uso deste ambiente. Uma vez que o tempo de implementação de um ambiente de *data warehousing* virtual é reduzido, e o investimento monetário requerido tende a ser pequeno, esse ambiente pode ser utilizado pela corporação com o objetivo de se identificar *se* e *como* os usuários de SSD utilizarão os dados para a tomada de decisão estratégica. Pode-se determinar, por exemplo, qual o público alvo do ambiente de *data warehousing*, qual o escopo dos dados a serem armazenados no *data warehouse* e quais as necessidades de consultas desse público alvo.

Logo após a fase de monitoramento do ambiente de *data warehousing* virtual, um ambiente de *data warehousing* convencional deve ser construído. *Data warehousing* virtuais devem, portanto, ser considerados como soluções temporárias de curta duração. Mais especificamente, *data warehousing* virtuais devem ser substituídos por *data warehousing* convencionais tão rápido quanto possível.

2.4 Modelagem Multidimensional

Em um ambiente de *data warehousing*, as análises efetuadas pelos usuários de SSD representam, de maneira geral, requisições multidimensionais aos dados do *data warehouse*, as quais têm por objetivo a visualização dos dados segundo diferentes perspectivas (dimensões). Sejam as seguintes entidades para o exemplo da cadeia de supermercados: filial, produto, tempo e vendas. No processo de melhoria do desempenho do negócio é necessário examinar os dados sobre as vendas segundo diversas perspectivas. Assim, pode-se examinar o volume de vendas por filial, o volume de vendas por produto, o volume de vendas por tempo, o volume de vendas por filial por produto, etc. Tais análises permitem a identificação de problemas e de tendências, garantindo aos executivos da empresa a possibilidade de formular estratégias efetivas.

O termo dado multidimensional normalmente se refere a dados representando objetos ou eventos que podem ser descritos, e portanto, classificados, por dois ou mais de seus atributos [Wel95, CR97]. Dados que mantêm uma correspondência única entre si, como é o caso de cliente e CPF, não são adequados para serem armazenados de acordo com a representação multidimensional [Pil98]. Na verdade, a determinação de quais dados serão multidimensionais ou não em um ambiente de *data warehousing* depende das consultas a serem realizadas pelos

usuários desse ambiente. Se tais consultas envolvem a recuperação de vários valores e a agregação desses valores, então os dados referenciados por estas consultas devem ser armazenados multidimensionalmente.

Resumindo, o paradigma multidimensional é adequado para modelar a estrutura natural de problemas de suporte à decisão, uma vez que permite a criação de modelos conceituais de negócios. Isto facilita a investigação, o resumo e a organização de dados voltados à análise destes tipos de problemas. Em particular, a visão do negócio do ponto de vista dos usuários de SSD enfoca conjuntos de dados multidimensionais e agregações estatísticas sobre as dimensões destes conjuntos de dados.

A seguir são discutidos os principais aspectos relacionados à modelagem multidimensional. Em particular, são enfocados: os aspectos estáticos e dinâmicos do modelo de dados multidimensional (seção 2.4.1), o armazenamento dos dados do *data warehouse* em sistemas relacionais e multidimensionais proprietários (seção 2.4.2), e alguns trabalhos existentes na literatura voltados à modelagem conceitual dos dados multidimensionais (seção 2.4.3).

2.4.1 Modelo de Dados Multidimensional

A utilização do modelo de dados multidimensional para a modelagem de aplicações de negócio influencia diretamente as ferramentas de análise e consulta que acessam os dados e o projeto do *data warehouse*. Essa seção discute tanto os aspectos estáticos (seção 2.4.1.1) quanto os aspectos dinâmicos (seção 2.4.1.2) do modelo de dados multidimensional, além de introduzir o conceito do cubo de dados (seção 2.4.1.3). Os aspectos estáticos enfocam a modelagem dos dados, enquanto que os aspectos dinâmicos englobam um conjunto de operações básicas que atuam sobre estes dados.

2.4.1.1 Aspectos Estáticos

Os aspectos estáticos do processamento analítico incluem um conjunto de **medidas numéricas**, que são os objetos de análise relevantes ao negócio, e um conjunto de **dimensões**, as quais determinam o contexto para a medida. Uma medida numérica pode ser definida como uma função de suas dimensões correspondentes, representando, desta forma, um valor no espaço multidimensional. Como exemplo, as medidas numéricas unidades-vendidas e número-clientes no *data warehouse* da cadeia de supermercados podem ser determinadas pelas dimensões produto, promoção, filial e tempo.

Medidas numéricas podem ser classificadas em aditivas, semi-aditivas e não aditivas. Uma medida numérica é **aditiva** quando pode ser somada através de todas as suas dimensões. A medida numérica unidades-vendidas é aditiva, uma vez que através de cada combinação de suas dimensões ela pode ser aritmeticamente somada. Em outras palavras, a agregação de unidades-vendidas diárias para unidades-vendidas mensais é realizada somando-se os valores de unidades-vendidas para cada um dos dias que formam o mês. O mesmo é válido para a agregação de unidades-vendidas por produto para unidades-vendidas para todos os produtos, de unidades-vendidas por filial para unidades-vendidas por todas as filiais, e assim por diante.

Medidas numéricas **semi-aditivas**, por outro lado, podem ser somadas somente através de algumas de suas dimensões, enquanto que através de outras o processo aditivo não tem significado algum. Número-clientes não é aditiva através da dimensão produto. Por exemplo, para dois produtos P_1 e P_2 vendidos na mesma filial sob a mesma promoção no mesmo dia, a soma de número-clientes do produto P_1 com número-clientes do produto P_2 não é válida, uma vez

que o mesmo cliente pode estar sendo contabilizado duas vezes. Neste caso, qualquer análise sobre os dados deve ser realizada considerando-se produtos individuais: a agregação de número-clientes do produto P_1 por dia para número-clientes do produto P_1 por mês é correta.

Já medidas numéricas **não aditivas** simplesmente não podem ser somadas. Preço é uma medida não aditiva, uma vez que a agregação de preço diário em preço mensal não é realizada somando-se todos os preços correspondentes aos dias que formam o mês. A agregação de preço por produto por filial em preço por produto para todas as filiais também não implica na soma de todos os preços por produto existentes para cada filial. Como conseqüência, esta medida numérica deve ser calculada utilizando-se média aritmética ou algum outro cálculo mais complexo.

Quanto às dimensões, cada uma delas pode ser descrita por um conjunto de **atributos**. A dimensão produto pode possuir três atributos: marca-produto, categoria e indústria. Assim, o produto "bolacha salgada" da marca "Bolachas & Cia" pertence à categoria "produtos alimentícios" e é produzido pela indústria "Bolachas & Bolachas". Atributos para as demais dimensões são: (i) dimensão promoção: tipo, redução-preço; (ii) dimensão filial: endereço, cidade, estado, região e (iii) dimensão tempo (dia): mês, trimestre, quadrimestre, semestre e ano. Durante toda esta seção, a nomenclatura dia ao invés de tempo será utilizada, para indicar a granularidade diária desta dimensão

Os atributos de uma dimensão podem se relacionar através de **hierarquias de relacionamento**, as quais especificam níveis de agregação e, conseqüentemente, granularidade dos itens de dados. No exemplo anterior, a filial é relacionada aos seus atributos cidade, estado e região através de um relacionamento hierárquico. Assim, filial $\rightarrow$ cidade $\rightarrow$ estado $\rightarrow$ região é uma hierarquia de relacionamento de nível quatro na dimensão filial. Similarmente, dia $\rightarrow$ mês $\rightarrow$ trimestre $\rightarrow$ semestre $\rightarrow$ ano é considerada uma hierarquia de relacionamento de nível cinco para o propósito de agregação, uma vez que dia (1 ... 30/31) = mês; mês (janeiro ... março) = 1° trimestre; mês (abril ... junho) = 2° trimestre; mês (julho ... setembro) = 3° trimestre; mês (outubro ... dezembro) = 4° trimestre; trimestre (1°, 2°) = 1° semestre; trimestre (3°, 4°) = 2° semestre; semestre (1°, 2°) = ano. Em especial, podem existir várias hierarquias de relacionamento ao longo de uma dimensão. Outra hierarquia para a dimensão dia é dia $\rightarrow$ mês $\rightarrow$ quadrimestre $\rightarrow$ ano.

Por fim, alguns atributos das dimensões contêm apenas informações adicionais sobre outro atributo da dimensão e, portanto, não podem ser utilizados em agregações. É o caso do atributo endereço (endereço da filial) da dimensão filial: ele tem sentido se relacionado apenas com filial.

2.4.1.2 Aspectos Dinâmicos

Uma característica essencial do modelo de dados para OLAP é a ênfase em agregações de medidas por uma ou mais dimensões, representando as requisições multidimensionais dos usuários de SSD e estabelecendo a organização dos dados do *data warehouse* segundo diferentes níveis de agregação (seção 2.1.3). Operações analíticas típicas incluem *drill-down*, *roll-up*, *slice and dice*, *pivot* e *drill-across*.

Drill-down consiste no processo de analisar os dados em níveis de agregação progressivamente mais detalhados, ou de menor granularidade. Como exemplo, um usuário pode iniciar sua análise em um alto nível de agregação (unidades-vendidas por marca-produto por ano) e sucessivamente detalhar sua análise através de atributos mais específicos das hierarquias de relacionamento das dimensões (unidades-vendidas por marca-produto por semestre, a seguir, unidades-vendidas por marca-produto por trimestre, a seguir, unidades-vendidas por produto por

mês, a seguir, unidades-vendidas por produto por dia). Esta operação é extremamente útil, uma vez que os usuários de SSD geralmente iniciam as suas análises em níveis altos de agregação e procuram detalhes mais específicos à medida que a identificação da origem de problemas encontrados é necessária.

Por outro lado, a operação *roll-up* representa o processo inverso da operação *drill-down*, possibilitando que a análise dos dados seja realizada em níveis de agregação progressivamente menos detalhados, ou de maior granularidade. Assim, usuários de SSD poderiam requisitar as seguintes visões: unidades-vendidas por produto por dia, a seguir, unidades-vendidas por produto por mês, a seguir, unidades-vendidas por marca-produto por trimestre, a seguir, unidades-vendidas por marca-produto por semestre, e finalmente, unidades-vendidas por marca-produto por ano.

Já a operação *slice and dice* permite que os usuários de SSD restrinjam os dados sendo analisados a um subconjunto destes dados. *Slice* refere-se ao "corte" através de uma ou mais dimensões para um valor fixo, enquanto que *dice* está relacionado à seleção de algumas faixas de valores para as dimensões remanescentes. Assim, a partir da visão multidimensional unidades-vendidas por produto por filial por dia, usuários podem aplicar a operação *slice and dice* para visualizar o subconjunto dos dados contendo as unidades-vendidas para o produto P_3 (*slice*) nas filiais F_1 e F_2 nos dias de D_4 a D_8 (*dice*).

Diferentes perspectivas dos mesmos dados podem ser obtidas pela operação *pivot*, a qual reorienta a visão multidimensional dos dados. Esta operação altera a ordem das dimensões, modificando, conseqüentemente, a orientação sob a qual os dados são visualizados. Através da utilização desta operação, usuários de SSD podem gerar qualquer combinação das dimensões de sua visão multidimensional, investigando desta forma diferentes relacionamentos que podem existir entre os dados. Aplicando-se a operação *pivot* à visão multidimensional unidades-vendidas por produto por filial por dia descrita acima, o usuário pode visualizar os seus dados de interesse através de diferentes combinações equivalentes: unidades-vendidas por produto por dia por filial, unidades-vendidas por dia por produto por filial, e assim por diante. Em relatórios bidimensionais, por exemplo, a operação *pivot* permite que as dimensões representadas em cabeçalhos de colunas e linhas sejam trocadas entre si em combinações arbitrárias, ou ainda permite mover uma das dimensões da linha em uma dimensão da coluna (Figura 2.4).

A operação *drill-across* compara medidas numéricas distintas que são relacionadas entre si através de pelo menos uma dimensão em comum. Dimensões em comum devem ser exatamente as mesmas, ou seja, devem possuir o mesmo significado e devem ter as mesmas restrições definidas para os seus atributos. Este princípio de projeto é obviamente satisfeito quando as dimensões em comum são, na verdade, compartilhadas entre as medidas numéricas sendo consideradas. Por exemplo, as três dimensões em comum entre a medida numérica unidades-vendidas determinada pelas dimensões produto, filial, dia e promoção e entre a medida numérica quantidade-enviada determinada pelas dimensões dia, produto, filial e fabricante podem ser exatamente as mesmas. Conseqüentemente, essas duas medidas numéricas podem ser combinadas e/ou comparadas entre si através da operação *drill-across*. Mesmo que as dimensões comuns difiram entre si pelo nível de granularidade, a realização desta operação ainda é totalmente plausível. Desde que os atributos da dimensão de maior granularidade sejam corretamente construídos a partir de agregações da dimensão de menor granularidade, a operação *drill-across* pode ser realizada baseada somente em atributos que existam em ambas as versões das dimensões.

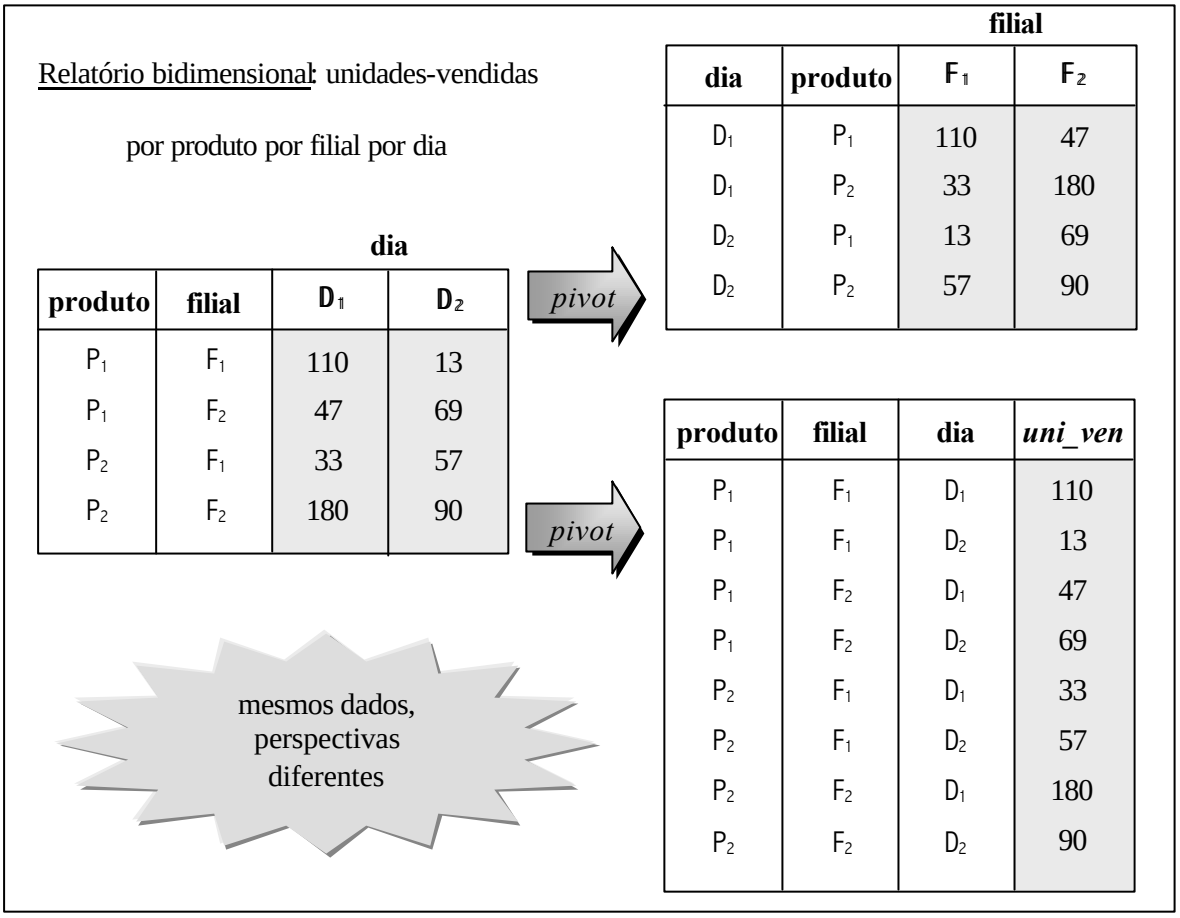

Figura 2.4 Exemplos da operação *pivot* em um relatório bidimensional.

2.4.1.3 Cubo de Dados Multidimensional

Consultas dos usuários de SSD ao ambiente de *data warehousing* são geralmente longas e complexas. Assim sendo, como descrito na seção 2.2.3.1, é importante que tais usuários sejam capazes de analisar os dados do *data warehouse* através de visões multidimensionais elaboradas, podendo navegar através destas visões de forma facilitada. Dentro deste contexto, o cubo de dados consiste em uma representação gráfica de grande utilidade para apresentar as visões dos usuários no espaço multidimensional [Sho97].

A Figura 2.5 mostra um cubo tridimensional, no qual os eixos x, y e z denotam, respectivamente, as dimensões produto, filial e dia. Os valores em cada célula do cubo representam medidas numéricas de interesse. Desta forma, o valor na coordenada (P_2,F_5,D_1) representa o valor agregado da medida numérica quando a dimensão produto tem valor P_2, a dimensão filial tem valor F_5 e a dimensão dia tem valor D_1. Do ponto de vista do usuário de SSD, é muito mais intuitivo visualizar a sua visão multidimensional (no caso, unidades-vendidas por produto por filial por dia) através deste cubo de três dimensões. Cubos que possuem mais do que três dimensões são chamados de hipercubos.

Na verdade, a semântica subjacente ao cubo de dados multidimensional permite não somente a visualização de valores de coordenadas específicas, mas também a identificação das várias agregações que podem ser originadas ao longo de todas as dimensões sendo consideradas. Por exemplo, a Figura 2.6, adaptada de Gray *et al*. [GBLP96], mostra o conceito de agregação (função de agregação = *soma*) para cubos de dados de até três dimensões, em termos de pontos, linhas, planos e cubos. Nessa figura, um cubo 0-dimensional é considerado um ponto, ao passo

que um cubo unidimensional é representado por uma linha com um ponto. Já um cubo bidimensional é especificado por uma tabela cruzada, ou seja, um plano, duas linhas e um ponto. Por fim, um cubo tridimensional pode ser visualizado como um cubo com três tabelas cruzadas que se intersectam. Um diagrama que tem sido comumente utilizado para representar logicamente o cubo de dados multidimensional é o grafo de derivação (seções 2.5.1.1 e 5.1.2).

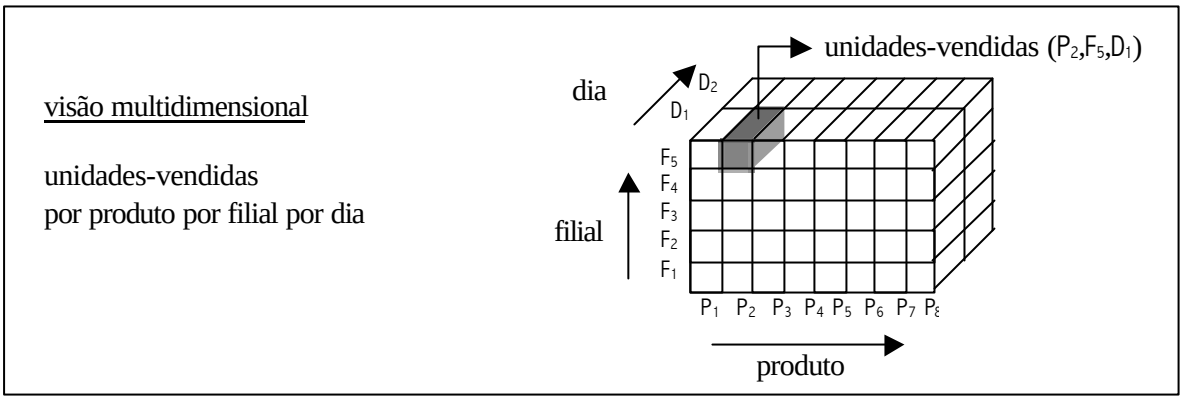

Figura 2.5 Cubo de dados tridimensional.

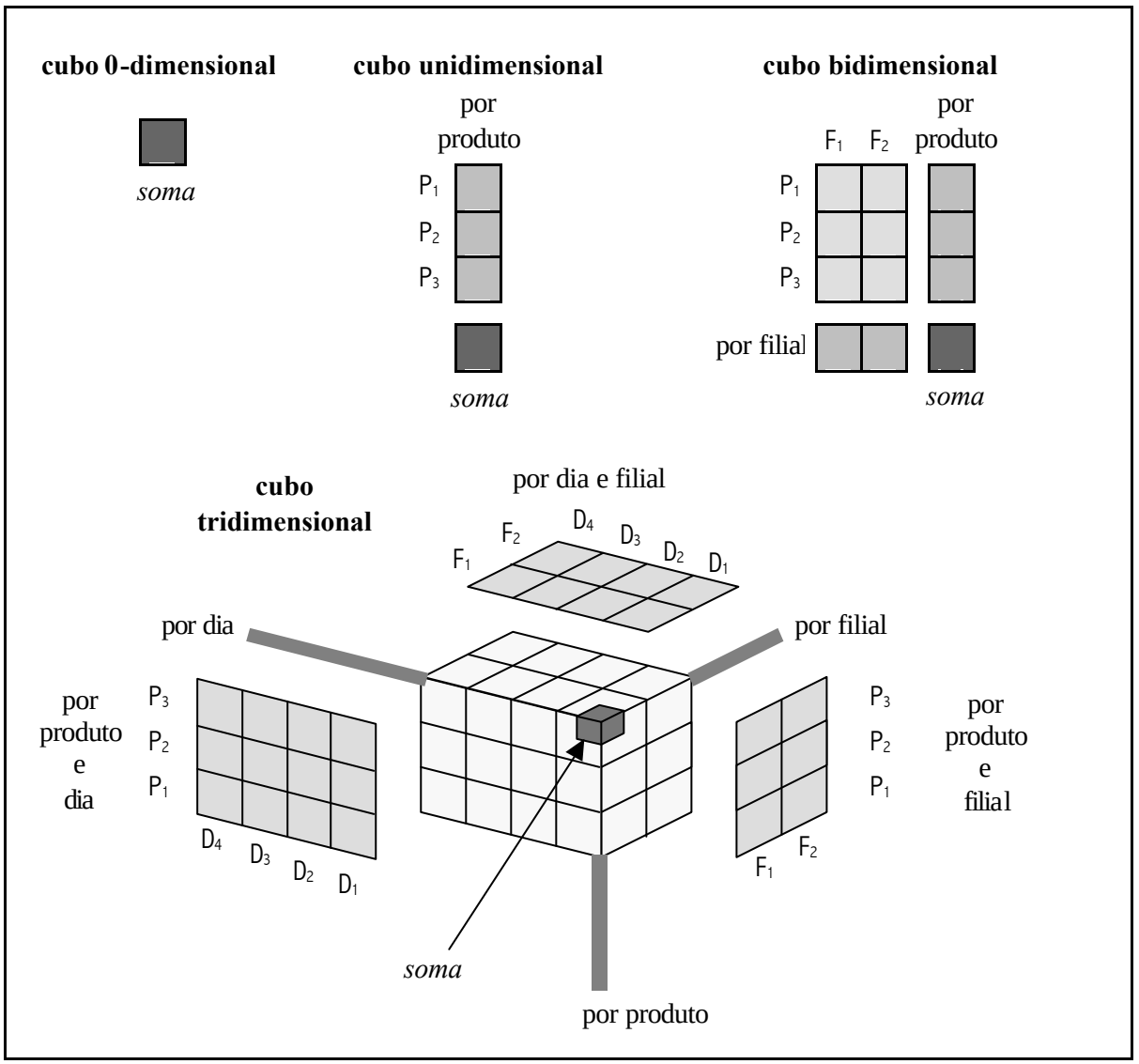

Figura 2.6 Representação do conceito de agregação para cubos de dados.

O cubo de dados também é uma interface natural para representar certas operações analíticas, tais como *pivot* e *slice and dice* (Figura 2.7).

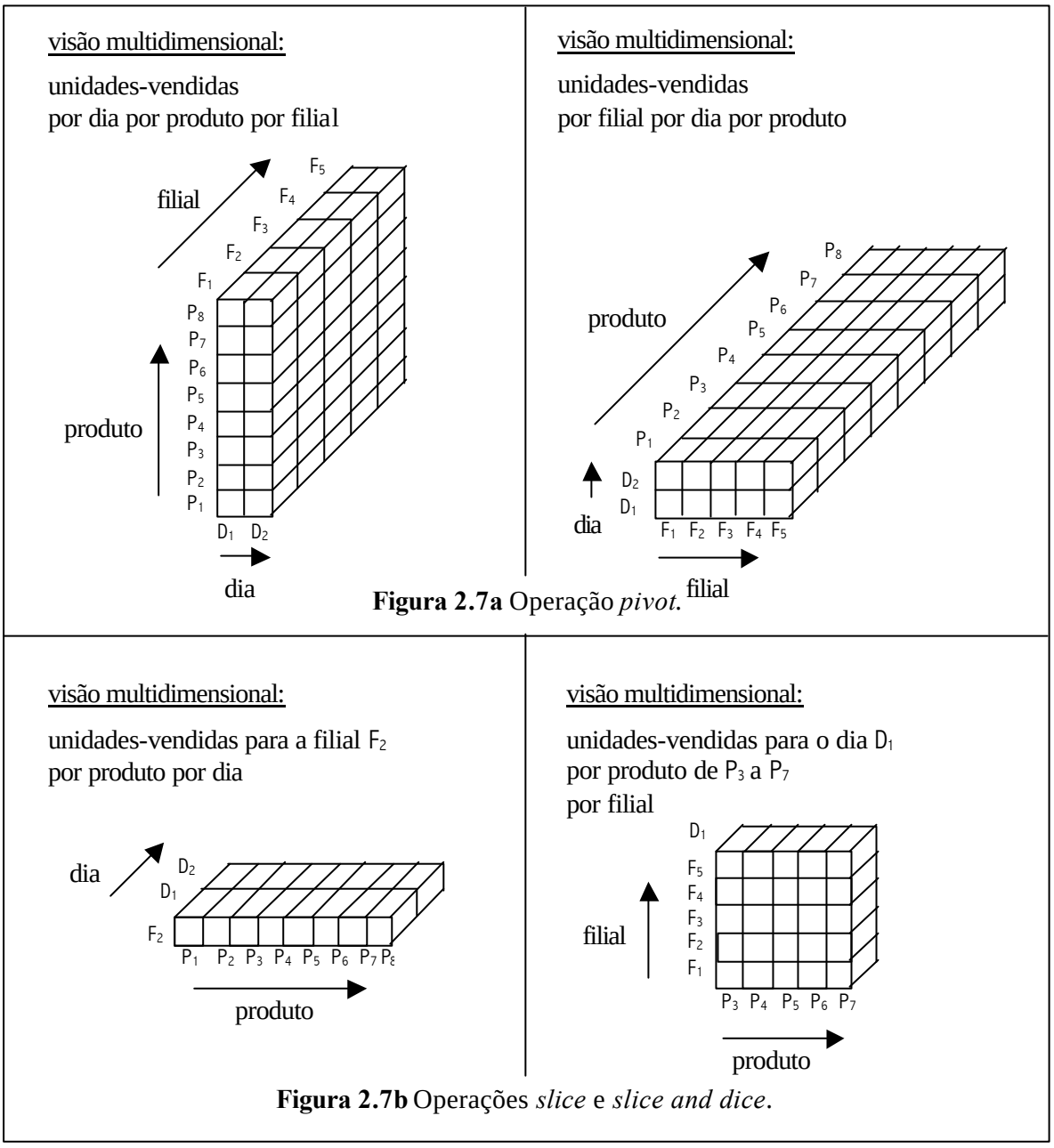

Figura 2.7a Operação *pivot*.

Figura 2.7b Operações *slice* e *slice and dice*.

Figura 2.7 Operações analíticas sobre o cubo de dados da Figura 2.5.

Por permitir que ambas as visões multidimensionais dos usuários de SSD e as operações analíticas sejam facilmente visualizadas através de sua estrutura, a perspectiva multidimensional do cubo de dados é referenciada como modelagem multidimensional. No entanto, o cubo de dados somente pode ser utilizado para expressar os conceitos desta modelagem. Em bancos de dados reais, esta estrutura deve ser mapeada em relações (em sistemas de banco de dados relacionais: seção 2.4.2.1) ou matrizes (em bancos de dados multidimensionais: seção 2.4.2.2).

2.4.2 Representação Lógica do Modelo de Dados Multidimensional

Existem duas principais abordagens correntemente utilizadas para a representação lógica do modelo de dados multidimensional: estruturas relacionais e estruturas de dados especializadas para armazenar o cubo de dados. Sistemas ROLAP (OLAP relacional) armazenam os dados do *data warehouse* em SGBD relacionais, os quais são geralmente estendidos para o suporte eficiente às consultas de tomada de decisão. Por outro lado, sistemas MOLAP (OLAP multidimensional) armazenam os dados multidimensionais em estruturas de dados especializadas e implementam operações analíticas diretamente sobre essas estruturas de dados. Essa seção tem por objetivo discutir as principais características de cada um destes sistemas.

2.4.2.1 Sistemas ROLAP

Sistemas ROLAP são geralmente baseados em dois componentes principais: uma **máquina relacional** (seção 2.4.2.1.1) e uma **máquina ROLAP** (seção 2.4.2.1.2). A máquina relacional diz respeito ao SGBD relacional adaptado para o processamento analítico, o qual é responsável pelo armazenamento dos dados. Já a máquina ROLAP representa a camada de *software* responsável em suprir as limitações da máquina relacional, apresentando os dados que residem no SGBD relacional de forma multidimensional para ferramentas de análise e consulta.

2.4.2.1.1 Máquina Relacional

Em sistemas ROLAP, os dados do *data warehouse* são armazenados em relações, as quais são organizadas especialmente para refletir a visão multidimensional dos dados. Existem dois tipos de esquema utilizados neste sentido: **esquema estrela** e **esquema floco de neve**.

O **esquema estrela** possui uma tabela de fatos dominante localizada visualmente no centro da estrela e um conjunto de tabelas de dimensão nas extremidades. A tabela de fatos armazena as medidas numéricas relevantes ao negócio (fatos), além dos valores das dimensões descritivas para cada instância, os quais são responsáveis pela ligação dos fatos às diversas dimensões. Em outras palavras, a chave primária da tabela de fatos é uma combinação das chaves primárias das tabelas de dimensão. Em geral, tabelas de fatos são longas e finas, ou seja, possuem um grande número de tuplas (bilhões de tuplas) e uma quantidade reduzida de colunas (chave primária e medidas numéricas). Por outro lado, cada dimensão do modelo de dados multidimensional é descrita por sua própria tabela de dimensão, a qual armazena os atributos alfanuméricos da dimensão e possui uma chave primária para cada uma de suas instâncias. Em geral, tabelas de dimensão são curtas e largas, ou seja, possuem um número pequeno de tuplas e uma grande quantidade de colunas (freqüentemente superior a 100 colunas). Em alguns casos, estruturas mais complexas, chamadas de **constelações de fatos**, podem ser empregadas, nas quais várias tabelas de fato compartilham tabelas de dimensão comuns.

O **esquema floco de neve** consiste em uma extensão do esquema estrela, na qual cada extremidade do esquema estrela passa a ser o centro de outras estrelas. Enquanto que as tabelas de dimensão do esquema estrela apresentam uma estrutura desnormalizada, visando obter um bom desempenho na execução de consultas altamente complexas, as tabelas de dimensão do esquema floco de neve são normalizadas de acordo com as hierarquias de relacionamento de atributos. Assim, esta característica do modelo de dados multidimensional é explicitamente representada neste esquema. Entretanto, o uso do esquema floco de neve deve ser ponderado

entre os ganhos em termos de espaço de armazenamento e os custos de complexidade para o usuário. Ademais, a utilização de uma estrutura normalizada implica em uma sobrecarga muito acentuada no processamento das junções das tabelas durante a execução da consulta [Col96].

Além de oferecer uma perspectiva diferente na forma de organização das relações, SGBD relacionais têm que ser estendidos com diversas funcionalidades adicionais para garantir suporte eficiente ao processamento de consultas complexas típicas de ambientes OLAP. A eficiência no processamento de consultas é um fator dominante para o desempenho geral do sistema e para a sua escalabilidade. Do ponto de vista da máquina relacional, quatro otimizações podem ser realizadas: (i) estender a linguagem SQL; (ii) oferecer estruturas de indexação especializadas; (iii) otimizar o processamento de consultas complexas e (iv) explorar o paralelismo.

A primeira otimização consiste em **estender a linguagem SQL** para oferecer operadores OLAP. A linguagem SQL padrão oferece somente cinco funções de agregação: *sum*, *count*, *avg*, *max* e *min*. No entanto, tais funções são limitadas quando comparadas com o espectro e a complexidade das consultas que podem ser requisitadas por usuários de SSD. A falta de funções analíticas adequadas faz com que mesmo a computação de consultas simples resulte em baixo desempenho.

Um operador de extrema importância proposto com este objetivo é o operador *cube* [GBLP96], o qual generaliza os construtores relacionais de agregação para permitir análise multidimensional. Este operador produz todas as agregações possíveis sobre as dimensões especificadas em uma visão multidimensional. Assim, se a visão engloba k dimensões, o operador *cube* produzirá 2^k subconjuntos destas dimensões. Por exemplo, para a visão multidimensional unidades-vendidas por produto por filial por dia, o operador *cube* gerará os seguintes subconjuntos: unidades-vendidas por produto por filial, unidades-vendidas por produto por dia, unidades-vendidas por filial por dia, unidades-vendidas por produto, unidades-vendidas por filial, unidades-vendidas por dia e total de unidades-vendidas. Cada um destes 2^k subconjuntos das dimensões é chamado de *cuboid*. Intuitivamente, este operador calcula para o sistema relacional todos os dados que podem ser visualizados através do cubo de dados multidimensional, como por exemplo, os ilustrados na Figura 2.6.

Diversas técnicas têm sido propostas para calcular eficientemente o operador *cube* na máquina relacional. Basicamente, estas técnicas exploram o relacionamento existente entre os diversos *cuboids*, ao invés de calcular cada *cuboid* independentemente. Em outras palavras, estas técnicas baseiam-se na premissa de que, a partir do momento que um *cuboid* pode ser calculado a partir de outro, a computação simultânea de diversos *cuboids* relacionados entre si pode ser realizada mais eficientemente do que a abordagem de gerar cada um dos 2^k *cuboids* independentemente. Uma descrição detalhada de várias técnicas com este propósito, incluindo as características e as limitações de cada uma, pode ser encontrada em [RS97]. Outro trabalho que pode ser citado é o de Souza e Sampaio [SS99], o qual otimiza o cálculo do operador *cube* privilegiando o critério de baixo custo no processamento de consultas (tempo de resposta) comumente realizadas pelos usuários de SSD, as quais são associadas aos diversos *cuboids*. Em especial, a rápida aceitação da importância do operador *cube* tem conduzido a uma variação do mesmo sendo proposta para o SQL padrão. Por fim, o trabalho mais recente de Han *et al.* [HPDW01] enfoca o cálculo eficiente do operador *iceberg-cube*, o qual produz apenas as agregações do cubo de dados cujos valores estão acima de um determinado limite.

Alguns SGBD também adicionam à linguagem SQL funções estatísticas, físicas e de análise financeira, dentre outras, além de permitir que os usuários de SSD introduzam novas funções de agregação. Por exemplo, o sistema de *data warehousing* Informix Red Brick oferece a linguagem RISQL (Red Brick Intelligent SQL), a qual consiste de um conjunto de extensões

para SQL que inclui funções apropriadas para aplicações de análise dos dados e suporte à decisão. Dentre estas funções, pode-se citar: *rank* ("quais os 50 produtos mais vendidos, baseado nas vendas do ano passado?"), *ratio-to-report* ("quais meses obtiveram o maior lucro nas vendas de um determinado produto durante o terceiro e quarto trimestres de 1998?"), *cume* ("quais as vendas mensais do produto P_1?"), *moving-avg* ("qual a média de movimentação de seis meses das vendas do produto P_1 nos últimos dois anos?") e *create-macro* ("liste as vendas anuais para este ano e compare com os resultados obtidos em cada um dos últimos cinco anos), a qual permite que consultas comuns sejam parametrizadas e escritas somente uma vez, e sejam compartilhadas por muitos usuários de SSD [Inf].

Já o trabalho de Agrawal *et al*. [AGS97] propõe um conjunto de operações algébricas, próximas da álgebra relacional que, conseqüentemente, podem ser facilmente traduzidas para a linguagem SQL. Para tanto, estendem a linguagem SQL de forma que esta suporte funções na cláusula GROUP BY e funções de agregação definidas pelos usuários que poderiam retornar conjuntos em cláusulas SELECT. Os operadores propostos, baseados em um modelo de dados cúbico que trata dimensões e medidas numéricas simetricamente são: (i) *push* e *pull*, para tratamento dessa simetria; (ii) *restrict*, que opera em uma dimensão do cubo e remove os valores da dimensão que não satisfazem à condição exigida; (iii) *destroy*, que reduz a dimensionalidade do cubo em um; (iv) *join*, que relaciona os dados de dois cubos baseado em uma função que define como os elementos do cubo resultante devem ser obtidos (por exemplo, divida os dados do primeiro cubo pelos dados correspondentes do segundo cubo); e (v) *merge*, que é uma operação que agrega as dimensões de acordo com as suas hierarquias de atributos. As operações *restrict*, *destroy*, *join* e *merge* são descritas em maior nível de detalhe na seção 3.4.1.

Outra otimização que pode ser realizada pela máquina relacional para aumentar o desempenho no processamento de consultas OLAP é a utilização de **estruturas de indexação especializadas**. Uma vez que consultas realizadas pelos usuários de SSD têm um alto fator de seletividade (recuperam grandes conjuntos de tuplas como resposta a uma consulta) e enfatizam a agregação de medidas por uma ou mais de suas dimensões, métodos de indexação para estes ambientes devem oferecer funcionalidades adicionais, tais como: garantir suporte simétrico a todas as dimensões do cubo, permitir que vários valores chave sejam buscados simultaneamente e oferecer suporte eficiente aos índices associados aos vários níveis de agregação.

Uma abordagem promissora consiste no uso de índices do tipo mapa de *bits* (*bitmap*). Na sua forma mais simples, um índice do tipo mapa de *bits* em um atributo indexado consiste de um vetor de *bits* que armazena os valores 0 ou 1, dependendo do valor do predicado sendo considerado. Um índice do tipo mapa de *bits* B lista todas as linhas com um determinado predicado de tal forma que, para cada linha de número i cujo atributo indexado satisfaz ao predicado, o i-ésimo bit em B é igual a 1. Além de apresentarem tamanho compacto, principalmente para atributos de baixa cardinalidade, índices do tipo mapa de *bits* permitem que operações tais como *and*, *or*, *xor* ou *not* sejam realizadas eficientemente.

Vários índices do tipo mapa de *bits* (por exemplo, *value-list index*, *projection index*, *bit-sliced index* e *groupset index*) têm sido projetados para diferentes tipos de consultas, incluindo consultas de agregação, consultas do tipo *range queries* e consultas do tipo OLAP [OQ97]. Mais detalhadamente, consultas de agregação são consultas que utilizam as funções *sum*, *count*, *avg*, *max* e *min*, enquanto consultas do tipo *range queries* são consultas cujo predicado inclui faixas de valores. Faixas de valores podem ser contínuas, ou seja, "anos de 1993 a 1998" ou discretas, por exemplo "primeiro mês de cada ano" e "produto IN {sapato, bolsa, cinto}". Já consultas do tipo OLAP envolvem funções de agregação e de agrupamento.

Uma vez que diferentes índices do tipo mapa de *bits* apresentam desempenhos distintos de acordo com o tipo de consulta, pode ser interessante manter-se vários desses tipos de índice para um mesmo atributo em um ambiente de *data warehousing*. A utilização de vários índices para um mesmo atributo permite que o melhor índice possa ser escolhido durante o processamento da consulta. Além disto, desde que o ambiente de *data warehousing* é caracterizado por ser basicamente do tipo somente para leitura, a utilização de vários índices não introduz uma alta complexidade ao ambiente. No entanto, para se obter um bom projeto do *data warehouse*, deve-se realizar um estudo detalhado contrastando o espaço necessário para o armazenamento dos vários índices e o desempenho na execução da consulta [CI98].

Uma discussão de outras técnicas de indexação para OLAP, incluindo métodos hierárquicos e a aplicabilidade de métodos multidimensionais originalmente projetados para dados espaciais pode ser encontrada em [Sar97].

A terceira funcionalidade que deve ser oferecida pela máquina relacional para o suporte eficiente ao ambiente de *data warehousing* consiste em **otimizar o processamento de consultas complexas**. Como discutido através de várias seções deste trabalho, o ambiente de *data warehousing* é caracterizado por consultas que manipulam grandes volumes de dados, pela existência de visões com alto grau de agregação e pelo pré-armazenamento de vários níveis de agregação. Escolher um plano de execução que seja ruim pode implicar não somente em baixo desempenho em termos de tempo de resposta, mas também em uma grande quantidade de espaço adicional necessário para executar a consulta. Em especial, a existência de diversos níveis de agregação pré-armazenados incrementa o número de representações relacionais possíveis para responder a uma mesma consulta, à medida que diferentes níveis de agregação mais inferiores podem ser utilizados para responder consultas de níveis de agregação mais superiores. Outro fator que afeta o otimizador de consultas complexas é o tipo da estrutura de indexação utilizada. A adição de novos índices especializados, como o índice do tipo mapa de *bits* descrito anteriormente, introduz novos métodos de acesso que devem ser considerados durante o processamento da consulta.

O problema de encontrar técnicas eficientes para processar consultas complexas tem sido de grande interesse na área acadêmica, sendo que algumas destas técnicas já estão sendo implementadas atualmente em máquinas relacionais. Os trabalhos de Chaudhuri e Dayal [CD97] e Harinarayan [Har97] listam e classificam uma série de artigos relacionados a este tópico, incluindo otimização de consultas SQL complexas aninhadas e consultas de agregação. Já o trabalho de Fang *et al*. [FSGM+98] apresenta estratégias de execução eficientes para consultas *iceberg*. Consultas *iceberg* são consultas do tipo OLAP nas quais valores de agregados que estão abaixo de uma determinada especificação são eliminados.

Aliado às otimizações acima destacadas, a **execução paralela de consultas complexas** através do particionamento dos dados do *data warehouse* entre um conjunto de processadores pode aumentar significativamente o desempenho do processamento da consulta. Para tanto, devem ser considerados aspectos tais como localização paralela dos dados do *data warehouse* e junção paralela [SMKK98].

2.4.2.1.2 Máquina ROLAP

A máquina ROLAP interpreta as consultas dos usuários de SSD e converte-as em consultas SQL complexas que acessam os dados armazenados na máquina relacional. Esta máquina implementa, assim, a transformação de uma consulta realizada de acordo com o modelo multidimensional em uma consulta relacional. Considerando-se o processo inverso, o servidor ROLAP oferece visões multidimensionais a partir dos dados que residem na máquina relacional para ferramentas de

análise e consulta, as quais formatam os dados de acordo com as necessidades dos usuários de SSD.

A máquina ROLAP complementa a máquina relacional tanto em termos de funcionalidades oferecidas quanto em termos de desempenho. Quanto mais eficiente a capacidade de processamento de consultas complexas da máquina relacional e mais próximas da máquina ROLAP as funcionalidades oferecidas pelo SGBD relacional, menor o suporte que deve ser oferecido pela máquina ROLAP, e vice-versa. Em geral, a máquina ROLAP explora a escalabilidade e as características transacionais dos sistemas relacionais.

Uma funcionalidade adicional de extrema importância que deve ser oferecida pela máquina ROLAP é a determinação de quais visões correspondentes aos diferentes níveis de agregação devem ser materializadas. De fato, a materialização de visões consiste na técnica mais efetiva para melhorar o desempenho no processamento de consultas multidimensionais. Quanto maior o número de visões materializadas, melhor o desempenho do sistema, porém pior a escalabilidade do sistema e maior o tempo necessário para a atualização dos dados do *data warehouse*. Assim sendo, um estudo detalhado considerando-se estes três fatores (desempenho *versus* escalabilidade *versus* tempo de atualização) deve ser realizado para um bom projeto do ambiente de *data warehousing*. Quando a máquina ROLAP materializa um conjunto de visões correlatas, ela deve endereçar outro problema importante: a reformulação das consultas dos usuários de SSD em termos de visões materializadas apropriadas de forma transparente. O tópico visão materializada é tratado em maior nível de detalhe na seção 2.5.

Uma vez determinadas quais visões correspondentes a quais níveis de agregação devem ser materializadas, estas são geralmente armazenadas em tabelas relacionais de pelo menos duas formas diferentes. Por um lado, são criadas tanto tabelas de fatos distintas para as diferentes agregações pré-computadas quanto tabelas de dimensão "reduzidas" correspondentes às dimensões que foram agregadas, as quais devem conter somente os atributos condizentes com o nível de agregação. As demais tabelas de dimensão são compartilhadas entre as diversas tabelas de fatos. Por outro lado, os dados agregados são armazenados nas mesmas tabelas de fatos e dimensão já existentes, sem adição de novas tabelas. Em contrapartida, Roussopoulos e Kotidis [RK98, KR98] propõem uma estrutura alternativa para o armazenamento das visões agregadas materializadas em sistemas ROLAP: *cubetree storage organization* (CSO). Tal estrutura consiste de uma coleção de árvores-R e provém ambos o armazenamento e a indexação em uma única organização. Segundo os autores, CSO apresenta melhor desempenho no processamento de consultas e garante melhor utilização do espaço de armazenamento quando comparado com tabelas relacionais e índices convencionais.

2.4.2.2 Sistemas MOLAP

Sistemas MOLAP surgiram como uma segunda alternativa para a representação e o armazenamento do cubo de dados multidimensional. Tais sistemas armazenam diretamente os dados multidimensionais em estruturas de dados especializadas (máquina de armazenamento multidimensional) proprietárias e implementam as operações OLAP diretamente sobre estas estruturas. Do ponto de vista lógico, o cubo de dados pode ser visto como uma matriz multidimensional, na qual os índices, os quais representam as dimensões, variam sobre os segmentos contínuos dos números naturais e as células, as quais representam as medidas numéricas, são formadas pela interseção de todas as dimensões.

Uma vez que as medidas numéricas da estrutura descrita acima são formadas pela interseção de todas as dimensões, nem todas as células possuem valores. No exemplo da cadeia de

supermercados, nem todos os produtos são vendidos em todas as filiais sob todas as promoções em todos os dias. Assim, a matriz multidimensional é caracterizada por ser esparsa. Em outras palavras, uma grande porcentagem das células do cubo multidimensional geralmente é vazia. Como conseqüência, muitos sistemas MOLAP adotam uma representação de armazenamento em dois níveis para adaptar os conjuntos de dados esparsos, além de utilizarem técnicas de compressão dos dados [CD97, DSHB98].

Em uma representação de armazenamento de dois níveis, matrizes mais ou menos densas são indexadas por uma estrutura que contém combinações esparsas dos dados. Mais especificadamente, o nível inferior armazena as dimensões mais ou menos densas em forma de matrizes. Já o nível superior armazena as combinações das dimensões esparsas como uma estrutura de indexação que contém ponteiros para as matrizes do nível inferior, as quais possuem a informação desejada. Em especial, técnicas de compressão dos dados podem ser aplicadas às matrizes do nível inferior, uma vez que estas podem não ser totalmente densas. A determinação do conjunto de dimensões densas e esparsas é realizada através do uso de ferramentas de projeto ou pela entrada do usuário, ao passo que o endereçamento do nível inferior é geralmente realizado através da utilização de técnicas de indexação tradicionais, tais como árvore-B, árvore-B+ e *hash* [FZ92].

Segundo Sarawagi [Sar97], o sistema Hyperion Essbase [Hyp] utiliza a técnica de indexação árvore-B para representar a estrutura de nível superior. Neste sistema, o conjunto de dimensões densas e esparsas é identificado pelo usuário. Uma árvore de índice é construída na combinação dos valores das dimensões esparsas, na qual cada entrada em uma de suas folhas aponta para uma matriz multidimensional formada pelas dimensões densas. Os campos de dimensão são mapeados para inteiros contínuos que determinam as posições das matrizes multidimensionais, as quais armazenam em suas células todos os valores das medidas numéricas e são compactadas quando necessário. No exemplo da cadeia de supermercados, as dimensões produto e filial podem ser dimensões densas, ao passo que as dimensões dia e promoção podem ser dimensões esparsas. Nesse caso, a árvore-B é construída em cada par produto-filial e, para cada par de valores no qual o dado é presente, uma matriz bidimensional de dia por promoção é armazenada.

Apesar das técnicas de armazenamento utilizadas por sistemas MOLAP serem variações da estrutura de dois níveis descrita acima, Dinter *et al*. [DSHB98] ressaltam que, em contraste com sistemas ROLAP, estes sistemas não possuem uma tecnologia uniforme e comumente aceita para armazenar os seus dados. No entanto, tais sistemas consideram o armazenamento físico dos dados multidimensionais um fator decisivo, uma vez que este influencia diretamente o desempenho, os requisitos de memória e o volume máximo de dados. Dinter *et al*. também realizam um levantamento de diversos fatores que deveriam ser considerados por técnicas de armazenamento de dados oferecidas por sistemas MOLAP.

Por outro lado, a determinação de quais visões correspondentes aos diferentes níveis de agregação devem ser materializadas é tão importante para sistemas MOLAP quanto para sistemas ROLAP. Assim sendo, os estudos realizados neste sentido podem ser aplicados a ambos os tipos de sistemas. No entanto, no caso de sistemas MOLAP, tais agregações pré-computadas também são armazenadas de acordo com a estrutura de matrizes.

Assim como Gray *et al*. [GBLP96] propuseram o operador *cube* para sistemas ROLAP, o trabalho de Zhao *et al*. [ZDN97] apresenta um algoritmo eficiente para calcular esse operador para sistemas MOLAP. Como discutido anteriormente, o operador *cube* produz todas as agregações possíveis sobre as dimensões especificadas em uma visão multidimensional. O algoritmo proposto, baseado no armazenamento dos dados em matrizes esparsas, calcula todas estas agregações minimizando tanto a quantidade de memória principal utilizada quanto o custo

de entrada/saída. Segundo os seus autores, o algoritmo proposto pode ser eficientemente aplicado a sistemas ROLAP para o mesmo propósito, através de três passos: (i) leitura da tabela relacional e armazenamento dos seus dados na estrutura de matriz; (ii) cálculo do operador *cube* na matriz resultante; e (iii) armazenamento da matriz resultante em tabelas relacionais.

Uma característica peculiar de sistemas MOLAP é que muitas vezes eles são construídos no topo de sistemas relacionais [Wie97]. Como exemplo, o produto de *data warehousing* Pilot Decision Support da Pilot Software [Pil98] efetua o armazenamento dos dados de menor nível de granularidade (maior nível de detalhe) em um banco de dados relacional.

2.4.2.3 Impacto da Representação Lógica dos Dados Multidimensionais

A nomenclatura ROLAP/MOLAP está relacionada à classificação de sistemas de *data warehousing* com relação ao paradigma utilizado para o armazenamento dos dados. Assim sendo, não somente alguns conceitos comuns são tratados de maneira diferente, mas também causam diferentes impactos em ambos os sistemas.

Enquanto que sistemas MOLAP utilizam técnicas de indexação e de compressão para o armazenamento de dados esparsos, esta característica do cubo de dados multidimensional é manipulada implicitamente em sistemas relacionais. O esquema estrela simplesmente não armazena as tuplas referentes às combinações de chaves primárias (correspondentes às dimensões) que originam valores não válidos.

A variação do número de dimensões, por sua vez, causa um impacto diferenciado em sistemas ROLAP e MOLAP. Devido à presença das perspectivas embutidas diretamente na estrutura de armazenamento de dados, adicionar dimensões em sistemas MOLAP implica que a forma da matriz que armazena os dados será alterada (por exemplo, de uma matriz bidimensional para uma tridimensional). Já adicionar dimensões em uma tabela relacional implica em adicionar novos atributos às tuplas da tabela (adicionar colunas).

No entanto, grande parte do debate envolvendo sistemas ROLAP e MOLAP não é baseado apenas no critério relacionado à forma de armazenamento dos dados, mas também nos critérios de capacidade de armazenamento, de desempenho e de funcionalidades oferecidas.

Com relação à capacidade de armazenamento, a abordagem relacional é muito mais escalável e pode manipular *data warehouses* de tamanhos grandes. Por outro lado, em sistemas MOLAP a capacidade de armazenamento é limitada pelo número de células permitidas. No entanto, esses sistemas requerem menor espaço de armazenamento, uma vez que os valores das medidas numéricas são compactados em matrizes multidimensionais e os valores das dimensões não precisam ser explicitamente armazenados.

Sistemas MOLAP implementam as operações OLAP diretamente sobre as suas estruturas de dados especializadas proprietárias. Uma vez que a aritmética de matrizes é mais eficiente do que a manipulação de dados relacionais, o desempenho no suporte às consultas destes sistemas é melhor do que o desempenho no suporte às consultas de sistemas ROLAP. Em adição a esta característica, a utilização de técnicas de indexação eficazes embutidas na estrutura de armazenamento destes sistemas permite que os dados sejam recuperados mais rapidamente. Em contrapartida, sistemas MOLAP não oferecem suporte eficiente a consultas *ad hoc*, pois são otimizados para operações multidimensionais específicas.

As funcionalidades adicionais oferecidas por sistemas ROLAP tiram proveito do fato que a tecnologia relacional encontra-se mais consolidada. Assim, o uso de SGBD relacionais permite a incorporação de novas funcionalidades, tais como paralelismo, distribuição e tratamento de

versões. A utilização desses sistemas também garante maior integração com dados não-numéricos, os quais são freqüentemente relevantes para aplicações de *data warehousing*. Por fim, sistemas ROLAP trabalham com uma maior variedade de plataformas servidoras.

Ambas as abordagens MOLAP e ROLAP apresentam pontos positivos e negativos. Com a finalidade de restringir os aspectos negativos, diversas melhorias têm sido propostas, as quais foram analisadas ao longo das seções 2.4.2.1 e 2.4.2.2. Por exemplo, o aumento no desempenho do processamento de consultas e a economia de espaço de armazenamento podem ser obtidos por meio do uso de técnicas avançadas de indexação. Outra melhoria refere-se à materialização de somente um subconjunto das visões correspondentes aos diversos níveis de agregação. Tal melhoria garante melhor desempenho para sistemas ROLAP, uma vez os resultados de consultas comumente realizadas são pré-computadas, e melhor escalabilidade para sistemas MOLAP, uma vez que o armazenamento de todos os níveis de agregação nestes sistemas degrada a capacidade de armazenamento dos mesmos. Como conseqüência, estas abordagens estão cada vez mais competitivas, sendo cada vez mais utilizadas através de todos os tipos de negócio.

2.4.2.4 Armazenamento dos Dados Multidimensionais em SGBD Objeto-Relacionais

Outra abordagem que tem sido estudada mais recentemente para o armazenamento dos dados multidimensionais é a utilização da tecnologia objeto-relacional. SGBD objeto-relacionais utilizam como base a tecnologia relacional e estendem-na para o suporte eficiente aos conceitos de orientação a objetos, tais como objeto complexo, polimorfismo, herança e encapsulamento [EN00]. Oferecem, desta forma, todas as vantagens de SGBD relacionais, acopladas à capacidade de manipulação de tipos de dados complexos como se fossem nativos do próprio sistema. Dentro deste contexto, o trabalho de Klein *et al*. [KCT99] discute a viabilidade do emprego desta tecnologia em ambientes de *data warehousing* em termos de forma de representação dos dados do cubo de dados e da arquitetura do ambiente.

SGBD objeto-relacionais estendem o esquema estrela convencional incorporando as abstrações orientadas a objetos na representação das medidas numéricas, das dimensões e dos métodos. Solucionam, conseqüentemente, algumas limitações apresentadas por esse esquema. Por exemplo, o esquema estrela não representa explicitamente hierarquias de relacionamento de atributos. Em SGBD objeto-relacionais, tal hierarquia pode ser representada pelo uso do conceito de generalização/especialização. Isto é realizado através do polimorfismo disponibilizado pelas tabelas tipadas, baseadas em hierarquias de tipos, com o uso de um diagrama de classes que identifica claramente a hierarquia. Outra limitação do esquema estrela refere-se ao fato de que a representação dos relacionamentos existentes entre dimensões implica na inclusão de novas tabelas ao esquema. Utilizando-se a tecnologia objeto-relacional, os relacionamentos podem ser expressos como um atributo das próprias dimensões, através do emprego da abstração de agregação por composição. Outra extensão inclui o uso de subtabelas tipadas para representar uma mesma dimensão que é usada várias vezes em uma tabela de fatos.

Além de solucionar as limitações acima apresentadas, SGBD objeto-relacionais também oferecem funcionalidades adicionais, como o uso de objetos complexos como atributos de dimensões e a incorporação de métodos e de rotinas disponibilizados em extensões ou criados pelos usuários. Como um exemplo dessa última funcionalidade, medidas numéricas derivadas, as quais são geradas a partir de outras medidas numéricas, podem ser implementadas facilmente através de métodos associados à classe fato.

Com relação à arquitetura do ambiente de *data warehousing*, a utilização da tecnologia objeto-relacional permite expandir o papel atual do SGBD para suporte à carga dos dados e à representação de informações. Para tanto, dois novos componentes são introduzidos: uma interface virtual e bibliotecas de classe ou extensões. A interface virtual permite a criação de tabelas virtuais de dados externos, facilitando o desenvolvimento de métodos de acesso através de *gateways* e permitindo que a carga de dados seja feita diretamente a partir do servidor que contém a base de dados. Já bibliotecas ou extensões permitem a criação e a integração de novos tipos, transformando séries temporais, imagens, mapas, vídeos, documentos, páginas HTML, dentre outros, em tipos manipulados pelo SGBD. É importante destacar que a máquina ROLAP, e muitas vezes as ferramentas de análise e consulta, devem ser modificadas uma vez que os dados estão representados em sistemas objeto-relacionais, melhor do que em relacionais.

Apesar da tecnologia objeto-relacional suprir algumas limitações do esquema estrela, Klein *et al*. argumentam que a criação de um esquema utilizando esta tecnologia apresenta complexidade elevada, quando comparado à tecnologia relacional. Outras desvantagens são o tempo de carga dos dados relativamente alto, a ineficiência na execução de algumas operações de agregação e a necessidade de adaptação de componentes da arquitetura de *data warehousing*.

2.4.3 Modelagem Conceitual dos Dados Multidimensionais

Outra área que tem sido de grande interesse por parte da comunidade acadêmica é a proposta de um modelo de dados conceitual multidimensional que englobe ambos os aspectos estáticos e dinâmicos do modelo de dados multidimensional [AGS97, GMR98, CT98]. O objetivo principal destas abordagens consiste em oferecer uma descrição conceitual dos dados independente da forma na qual estes estão armazenados (se em sistemas ROLAP ou MOLAP), similarmente ao que ocorre com a tecnologia relacional. Em especial, a utilização de um modelo conceitual permite que o projeto global do *data warehouse* corresponda melhor às necessidades dos usuários de SSD, além de garantir melhor documentação ao sistema. Assim sendo, trabalhos nesta área preocupam-se não somente com a proposta de uma modelagem conceitual dos dados e uma linguagem de consulta associada, mas também com a definição de mapeamentos dos conceitos oferecidos para os conceitos disponibilizados por ambos os sistemas ROLAP e MOLAP.

Agrawal *et al*. [AGS97] propõem um modelo conceitual no qual o aspecto estático é representado por dados organizados em um ou mais cubos, enquanto que o aspecto dinâmico é composto por um conjunto mínimo de operadores próximos da álgebra relacional. As principais características desse trabalho são o tratamento simétrico de dimensões e medidas, o suporte a várias hierarquias de relacionamento de atributos ao longo de cada dimensão e a possibilidade de realização de consultas *ad hoc* através da livre combinação dos operadores propostos. Maiores detalhes sobre esses operadores podem ser encontrados na seção 3.4.1. No entanto, o enfoque do trabalho de Agrawal *et al*. é mais voltado ao desenvolvimento de uma linguagem de consulta do que à modelagem de dados.

Por outro lado, a principal ênfase do trabalho de Golfarelli *et al*. [GMR98] é a especificação gráfica de um modelo conceitual estruturado hierarquicamente na forma de árvores. A modelagem dos dados (aspecto estático) consiste de um conjunto de esquemas de fatos. Os componentes básicos desses esquemas são fatos, dimensões e hierarquias ao longo destas dimensões. Em especial, um fato representa um foco de interesse, podendo possuir um ou mais atributos, ou seja, uma ou mais medidas numéricas. Por exemplo, os atributos do fato de interesse vendas são unidades-vendidas e lucro-obtido. Cada esquema de fato possui como raiz um fato e como folhas as dimensões que determinam aquele fato. Cada folha que representa uma dimensão pode ser a raiz de uma ou mais subárvores, as quais correspondem às várias

hierarquias de relacionamento de atributos da dimensão. As folhas mais inferiores representam os atributos de granularidade maior. Além de permitir a representação explícita de várias hierarquias de relacionamento para cada dimensão, o modelo proposto também diferencia a representação tanto de medidas numéricas aditivas, semi-aditivas e não aditivas, quanto de atributos que não podem ser agregados. Com relação aos aspectos dinâmicos, a representação das consultas é realizada graficamente através de um conjunto de marcadores localizados nos atributos das dimensões, os quais indicam implicitamente o nível de agregação dos fatos considerados na consulta. No entanto, a linguagem de consulta não possui fundamentação teórica. Esse trabalho também apresenta uma metodologia semi-automatizada para a derivação do modelo de dados proposto a partir de esquemas entidade-relacionamento.

A modelagem de dados apresentada por Cabbibo e Torlone [CT98] é baseada em dois construtores principais: dimensões e *f-table*. Cada dimensão é organizada em uma hierarquia de níveis, correspondendo aos domínios dos dados de diferentes granularidades (hierarquia de relacionamento de atributos). Dentro de uma dimensão, valores de diferentes níveis são relacionados através de funções *roll up*, de acordo com a hierarquia definida sobre eles. Assim, uma função *roll up* associa um valor v_1 de um certo nível com um valor v_2 de um nível mais superior da hierarquia. Ou seja, funções *roll up* descrevem intencionalmente como valores de diferentes níveis são relacionados entre si. Um nível pode possuir descrições associadas a ele, as quais oferecem informação adicional sobre o mesmo (atributos que não podem ser utilizados em agregações). Já *f-tables* são funções de coordenadas simbólicas, definidas com respeito a combinações particulares de níveis, para medidas: elas são utilizadas para representar os fatos. A Figura 2.8 mostra a modelagem de dados conceitual para o exemplo da cadeia de supermercados, considerando-se as dimensões produto, filial e dia, introduzidas na seção 2.4.1.1, e os fatos de interesse vendas e custo do produto, com os atributos unidades-vendidas e custo.

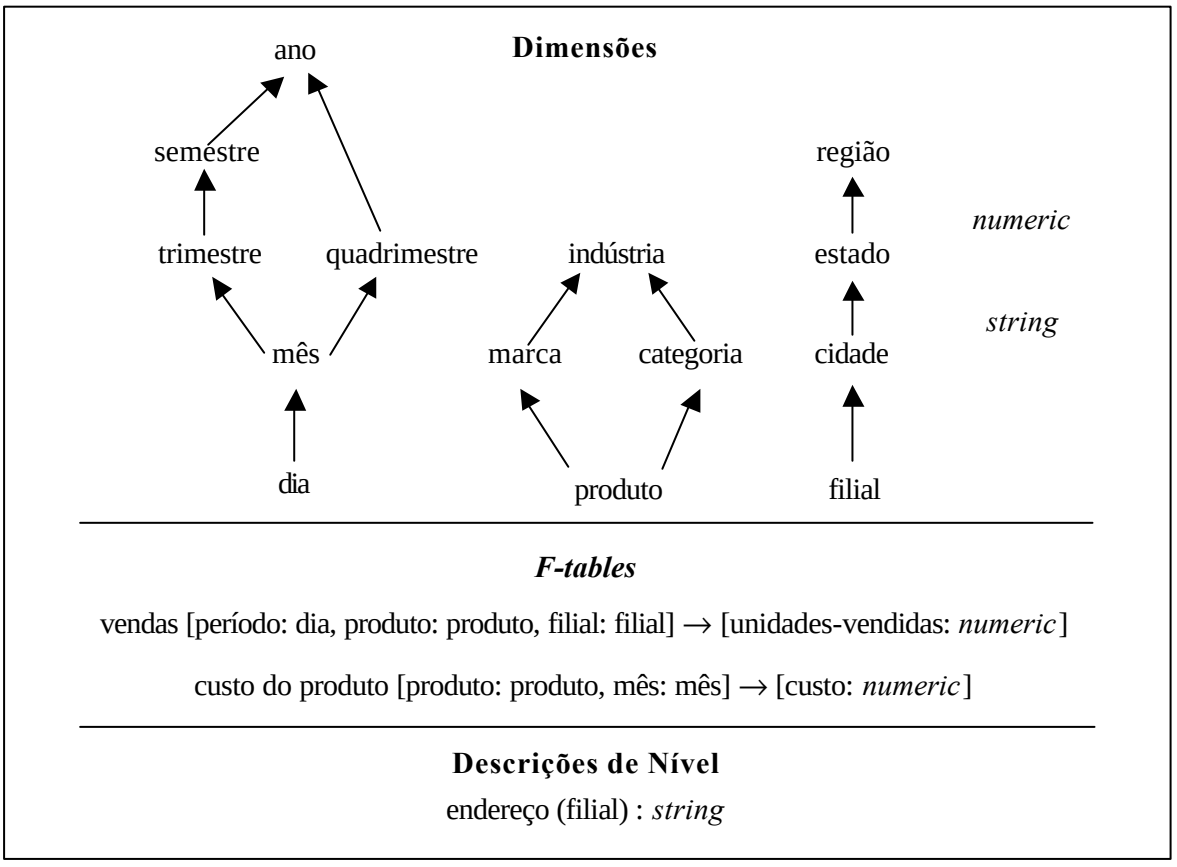

Figura 2.8 Modelagem conceitual para o exemplo da cadeia de supermercados.

Três tipos de linguagem de consulta utilizando três paradigmas diferentes são associados aos aspectos dinâmicos do modelo de dados conceitual em consideração: (i) uma linguagem gráfica, para que usuários de SSD iniciantes representem as suas operações analíticas graficamente através da modelagem conceitual; (ii) uma linguagem textual declarativa baseada em cálculo, para que usuários de SSD sofisticados possam expressar consultas mais complexas; e (iii) uma linguagem algébrica procedural, envolvendo um conjunto de operações atômicas, para que otimizadores de consulta possam executar a consulta eficientemente. Uma vez que a linguagem algébrica é equivalente ao cálculo, e mais poderosa do que a linguagem gráfica, é possível traduzir, automaticamente, as linguagens gráfica e de cálculo em expressões algébricas. Detalhes de cada uma das linguagens oferecidas, além de mapeamentos do modelo de dados proposto para a implementação lógica em sistemas ROLAP e MOLAP podem ser encontrados em [CT00]. Dentre as principais características do modelo de dados proposto, pode-se citar: suporte a várias hierarquias de relacionamento de atributos ao longo de cada dimensão, possibilidade de realização de consultas *ad hoc*, representação explícita de atributos que não podem ser agregados e o tratamento simétrico de dimensões e medidas. Por fim, Cabbibo e Torlone [CT98] também apresentam uma metodologia similar à de Golfarelli *et al*. [GMR98] para o projeto conceitual do *data warehouse* a partir de esquemas entidade-relacionamento.

2.4.3.1 Abrangência do Tema Modelagem Conceitual dos Dados Multidimensionais

Além dos trabalhos descritos na seção 2.4.3, diversos outros trabalhos acadêmicos têm sido voltados à proposta de um modelo conceitual dos dados multidimensionais. Como exemplo, pode-se citar os trabalhos de Pedersen e Jensen [PJ99] e Tryfona *et al*. [TBC99]. Resumidamente, Pedersen e Jensen preocupam-se com a identificação de um conjunto de requisitos que qualquer modelo de dados multidimensional deveria satisfazer, e propõem um modelo de dados estendido e uma álgebra sobre os objetos multidimensionais desse modelo que atendem a esses requisitos. Tryfona *et al*., por outro lado, apresentam uma abordagem menos formal, estendendo o modelo entidade-relacionamento para o suporte à modelagem conceitual de *data warehouses*. O modelo proposto, denominado starER, combina os elementos do modelo entidade-relacionamento com a estrutura do esquema estrela (fatos sobre o negócio são centrais, ao passo que os demais dados são definidos em torno desses fatos).

Outro tópico sendo atualmente investigado pela área acadêmica refere-se à incorporação de conceitos do paradigma de orientação a objetos na modelagem conceitual dos dados multidimensionais. Neste sentido, o trabalho de Ravat *et al*. [RTZ99] caracteriza o esquema de uma aplicação de *data warehousing* por um conjunto de classes e um conjunto de restrições que configuram os objetos do ambiente de *data warehousing*. Em particular, as classes consideradas pelo modelo proposto podem ser tanto classes de objetos genéricos quanto classes de ligação entre classes de objetos genéricos. Enquanto as classes de objetos genéricos modelam as entidades (ou seja, os objetos) extraídas dos provedores de informação, as classes de ligação definem os relacionamentos existentes entre essas entidades. Estes tipos de classes e as suas respectivas instâncias são representados por Ravat *et al*. através da notação de diagrama de classes e de diagrama de objetos da linguagem UML.

Já o trabalho de Abelló *et al*. [ASS00] tem por objetivo destacar as vantagens introduzidas pela utilização de conceitos do paradigma de orientação a objetos na modelagem conceitual de *data warehouses*. Isto é realizado através da análise de seis dimensões orientadas a objetos, cada uma das quais representando diferentes relacionamentos entre os dados. As dimensões definidas pelos autores são: (i) classificação/instanciação, incluindo classificação dinâmica e classificação

múltipla; (ii) generalização/especialização, englobando herança simples e herança múltipla; (iii) agregação/decomposição, incluindo os relacionamentos de agregação simples e de composição; (iv) comportamental; (v) derivacional, para representar o tratamento dos dados derivados; e (vi) dinamicidade, para representar alterações nos objetos, nas classes e nas metaclasses ao longo do tempo. Esta última dimensão inclui o tratamento da versão dos objetos e da evolução dos esquemas, além do suporte a um modelo de dados extensível. Para cada uma destas dimensões, os autores aplicam os conceitos abordados na modelagem das características de ambientes de *data warehousing*.

O trabalho de Abelló *et al*. [ASS00] descrito no parágrafo anterior é expandido de modo mais formal em dois outros trabalhos dos mesmos autores: um investigando a representação de *dimensões* [ASS01a] e outro discutindo conceitos relacionados a *medidas numéricas* [ASS01b]. É importante reforçar o fato que todos esses trabalhos enfocam a modelagem dos dados no nível conceitual. Em [ASS01a], Abelló *et al*. exploram os conceitos de composição e de agregação simples (dimensão agregação/decomposição) para mostrar os relacionamentos existentes nas hierarquias de relacionamento de atributos de uma dimensão. Esses autores utilizam os conceitos de especialização e de agregação (dimensão generalização/especialização e dimensão agregação /decomposição, respectivamente) para analisar os relacionamentos existentes entre diferentes dimensões de um mesmo cubo de dados ou de diferentes cubos de dados. Além disto, na terminologia orientada a objetos, os níveis de agregação são representados como classes e as suas instâncias, como objetos (dimensão classificação/instanciação). Já em [ASS01b], *medidas numéricas* podem ser agrupadas em *células*, as quais, por sua vez, podem ser agrupadas em diferentes *classes*. *Classes* são representadas por *cubos* n-dimensionais. Finalmente, diferentes *cubos* podem ser agrupados em um *Fato*. Abelló *et al*. usam os conceitos de especialização (dimensão generalização/especialização), de agregação (dimensão agregação/ decomposição) e de derivação (dimensão derivacional) para explicar e exemplificar como diferentes *cubos* e *Fatos* podem ser relacionados.

2.5 Visões Materializadas

Na sua forma mais simples, uma visão é uma relação derivada definida em termos de relações básicas armazenadas. Em outras palavras, uma visão não materializada (visão virtual) é uma especificação de programa (definição da visão: intenção) que gera dados cada vez que é referenciada. Muitas vezes pode ser interessante armazenar os dados obtidos na execução de uma visão como uma extensão desta visão. Assim, além de sua intenção, a visão materializada também possui dados, podendo desta forma ser utilizada como base para a execução de outras visões ou ser coletivamente agrupada para construir supervisões [GM95, Rou98].

Visões materializadas podem ser adaptadas a qualquer problema de domínio que precise de rápido acesso aos dados derivados ou no qual recalcular a visão a partir das relações base pode ser caro ou inviável (por exemplo, quando os dados subjacentes são remotos). Esta técnica se torna extremamente importante para o ambiente de *data warehousing*, desde que este é caracterizado por armazenar grandes volumes de dados e por oferecer suporte a consultas complexas de longa duração sobre estes dados. Assim, uma primeira razão para a utilização de visões materializadas neste ambiente é o aumento no desempenho do processamento de consultas OLAP, garantindo tempos de resposta mais satisfatórios aos usuários de SSD. O custo de consultas escritas em termos de visões materializadas é significativamente menor do que o custo de consultas escritas em termos de visões virtuais. Tal afirmação é decorrente do fato que a extensão da visão é armazenada ao invés de ser calculada em tempo de execução e que, na maioria das vezes, as consultas podem ser respondidas sem acessar as relações base remotas. Em

um ambiente de *data warehousing*, visões também podem ser materializadas almejando-se diminuir os custos relacionados à atualização de outras visões materializadas.

Essas visões materializadas tornam-se inconsistentes à medida que as relações base correspondentes aos diferentes provedores de informação que participam do ambiente de *data warehousing* são alteradas. Desta forma, manter a consistência dos dados do *data warehouse* através da atualização incremental das visões materializadas representa um grande desafio ao ambiente (seção 2.5.2). Outra tarefa crítica é a determinação do conjunto de visões a serem materializadas considerando-se diversos requisitos conflitantes (seção 2.5.1). Por exemplo, quanto mais visões são materializadas, melhor o tempo de resposta às consultas e mais baixo o custo de entrada/saída. Em contrapartida, maior o espaço de armazenamento necessário e maior o custo relacionado à atualização incremental destas visões materializadas. Finalmente, se várias visões materializadas são mantidas no *data warehouse*, o processador de consultas do ambiente deve utilizar estas visões para otimizar consultas *ad hoc* dos usuários de SSD de forma transparente (seção 2.5.3).

2.5.1 Identificação de Visões a serem Materializadas

Como discutido na seção 2.1.3, o relacionamento entre visão e os dados armazenados no *data warehouse* pode ser observado sob dois enfoques. Por um lado, o nível inferior da hierarquia de agregação dos dados pode ser visto como um conjunto de visões nas quais as relações base residem nos provedores de informação que participam do ambiente de *data warehousing*. Tais visões podem ser tanto uma cópia das relações base quanto uma relação derivada destas. Por outro lado, cada um dos demais níveis da hierarquia de agregação pode ser considerado como um conjunto de visões do nível imediatamente subjacente. Estas visões associadas aos diversos níveis de agregação podem ser virtuais ou materializadas.

A determinação de quais visões correspondentes aos diferentes níveis da hierarquia de agregação devem ser materializadas é de suma importância para o processamento de consultas eficiente em ambientes de *data warehousing*. De maneira genérica, os trabalhos discutidos nas seções 2.4.2.1 e 2.4.2.2 para o cálculo do operador *cube* [GBLP96, RS97, ZDN97, SS99] podem ser aplicados quando todas as visões devem ser materializadas. No entanto, muitas vezes a materialização de todas as visões pode ser inviabilizada por certas restrições, tais como o espaço de armazenamento disponível e os custos relacionados à atualização dessas visões.

O requisito processamento de consulta eficiente é conflitante tanto com o tamanho do *data warehouse* quanto com o tempo gasto para se manter a consistência das diversas visões materializadas. Em um extremo, a materialização completa de todas as visões garante um excelente tempo de resposta às consultas submetidas ao sistema. No entanto, pré-calcular e armazenar cada visão não é uma solução adequada principalmente quando o espaço consumido torna-se excessivo, ou quando o tempo de manutenção é proibitivo. O outro extremo, quando nenhuma visão com exceção da correspondente ao nível inferior é materializada, implica em baixo desempenho no processamento das consultas OLAP. Em contrapartida, esta solução não implica em espaço de armazenamento adicional ou tempo gasto com a atualização dos demais níveis de agregação. Através da materialização no *data warehouse* de um conjunto selecionado apropriado de visões, os custos relacionados ao processamento de consultas e à manutenção das visões podem ser mantidos em níveis aceitáveis, sem se desprezar as limitações de espaço de armazenamento.

Escolher um conjunto certo de visões a serem materializadas é uma tarefa não trivial, uma vez que uma visão materializada pode ser utilizada para o cálculo rápido de outras visões. Desta

forma, pode-se materializar uma visão correspondente a uma consulta relativamente pouco realizada se esta ajudar a responder a várias outras consultas rapidamente. Como pode ser observado, decidir quais visões materializar é equivalente a decidir quais consultas freqüentemente realizadas materializar. No entanto, ao longo desta seção, o termo consulta será utilizado para referenciar as consultas representativas dos usuários de SSD, enquanto que o termo visão corresponderá aos diferentes níveis da hierarquia de agregação.

Dentro deste contexto, trabalhos nesta área [HRU96, BPT97, GHRU97, TS97, SDN98, KR99] enfocam o oferecimento de algoritmos baseados em heurística que determinam quais visões devem ser materializadas, quando o sistema não permite a materialização de todas as visões. Dado a restrição de espaço, e se disponível um conjunto de consultas comumente realizadas pelos usuários de SSD, estes trabalhos selecionam um conjunto apropriado de visões a serem materializadas de forma a garantir um bom desempenho no processamento das consultas. Genericamente, a principal diferença apresentada por estes trabalhos está relacionada ao enfoque adotado: (i) se os algoritmos propostos visam otimizar os custos associados ao cálculo da consulta (seção 2.5.1.1); ou (ii) se os algoritmos propostos visam otimizar ambos o cálculo da consulta e a manutenção da visão (seção 2.5.1.2). Outro enfoque refere-se à materialização dinâmica das visões (seção 2.5.1.3), em oposição ao pré-selecionamento estático dessas visões.

2.5.1.1 Materializando Visões com Relação ao Espaço de Armazenamento

O conflito desempenho no processamento da consulta *versus* espaço de armazenamento é abordado no trabalho de Harinarayan *et al*. [HRU96]. Esse trabalho propõe um algoritmo que identifica quais visões devem ser materializadas de forma a minimizar o custo associado ao cálculo da consulta. Considera, para tanto, o modelo de custo linear, no qual o custo para responder a uma consulta é proporcional ao número de linhas examinadas (tamanho da visão). Em outras palavras, o tempo para responder a uma consulta é igual ao espaço ocupado pela visão materializada que responde a esta consulta. O algoritmo proposto, denominado *greedy*, baseia-se em um *lattice* de visões, o qual captura as dependências existentes entre as visões. Uma visão v é dependente de outra visão u ($v \preceq u$) se e somente se v pode ser determinada usando somente os resultados de u.

Uma maneira apropriada de se representar um *lattice* de visões é através de um grafo, conhecido como grafo de *lattice* de visões ou grafo de derivação. A Figura 2.9 mostra o grafo de derivação para o exemplo da cadeia de supermercados, considerando-se as dimensões produto (p), filial (f) e tempo (t). Cada vértice do grafo representa uma visão, a qual agrega dados sobre as dimensões presentes naquele vértice e é nomeada de acordo com estas dimensões. Por exemplo, a visão pf representa uma visão agregada sobre o agrupamento das dimensões p e f. Já as dependências entre as visões adjacentes são representadas através de arestas direcionadas. Neste caso, a visão p pode ser calculada a partir da visão pf. O grafo de derivação possui dois vértices especiais: um (visão pft) que pode ser usado para derivar qualquer outra visão e outro (visão $vazio$) que pode ser calculado a partir de qualquer outra visão, e representa a visão completamente agregada.

Em situações nas quais não somente as dimensões são consideradas nas agregações, mas também os atributos destas dimensões, o grafo de derivação apresenta complexidade adicional. Essa complexidade advém do fato que as hierarquias de relacionamento de atributos introduzem novas dependências entre as visões, as quais devem ser levadas em consideração quando da decisão de quais visões materializar. Uma definição mais completa e formal de grafo de derivação é realizada na seção 5.1.2.

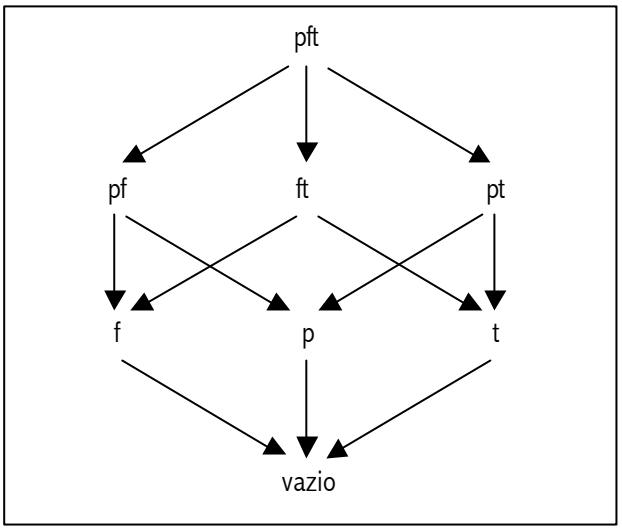

Figura 2.9 Grafo de derivação para as dimensões produto (p), filial (f), e tempo (t).

Para um determinado grafo de derivação com custos associados aos vértices, o algoritmo *greedy* determina o conjunto S de visões a serem materializadas da forma a seguir. Em um primeiro passo, o algoritmo sempre opta pela materialização da visão correspondente ao vértice superior do grafo, uma vez que esta pode ser utilizada para calcular qualquer outra visão. Assim, inicialmente S contém esta visão. No segundo passo, para cada uma das visões do grafo de derivação que ainda não pertence ao conjunto S, o algoritmo determina o benefício total desta visão e, dentre estas, seleciona a visão que maximiza o benefício e adiciona-a ao conjunto S. Este segundo passo é repetido até que um determinado número de visões materializadas seja selecionado ou que o limite de espaço de armazenamento disponível seja atingido. O benefício de uma visão v é determinado considerando-se como essa visão pode melhorar o custo para calcular outras visões, incluindo ela mesma. Assim, para cada visão w descendente de v, o algoritmo verifica se calcular w a partir de v apresenta menor custo do que calcular w a partir de qualquer outra visão no conjunto S. Se o custo de v é menor do que o custo de seu competidor, então a diferença entre estes custos, sempre positiva, representa parte do benefício (benefício parcial) de selecionar v como uma visão materializada. O benefício total de v consiste na soma de todos os benefícios parciais com relação a todas as visões w. Quando uma quantidade fixa de espaço de armazenamento é alocado, o algoritmo considera o benefício por unidade de espaço. Por outro lado, quando alguma probabilidade de acesso é associada às visões, indicando a freqüência na qual elas são requisitadas pelas consultas, então o algoritmo também considera estas probabilidades no cálculo do benefício.

Dois outros trabalhos, um de Shukla *et al*. [SDN98] e outro de Gupta *et al*. [GHRU97], estendem o trabalho de Harinarayan *et al*. [HRU96]. Tais trabalhos atacam o problema desempenho no processamento da consulta *versus* espaço de armazenamento, utilizam o mesmo modelo de custo linear e são baseados no grafo de derivação.

Enquanto que o algoritmo de Harinarayan *et al*. determina apenas um conjunto de visões a serem materializadas dado a restrição de espaço de armazenamento, os algoritmos propostos em [GHRU97] objetivam identificar tanto um conjunto de visões quanto um conjunto de índices associados a estas visões que devem ser materializados, de acordo com essa mesma restrição. O problema enfrentado por esta abordagem é que, quando existe a possibilidade de se associar um ou mais índices a uma determinada visão para aumentar a velocidade de processamento de certas consultas, todo o benefício de se materializar esta visão pode residir nos índices posteriormente associados a ela.

Assim, Gupta *et al*. oferecem uma família de algoritmos, chamados de *r-greedy*, que integram a seleção de visões e índices em um único passo. Similarmente ao algoritmo *greedy*, os algoritmos *r-greedy* executam em vários estágios, selecionando a cada estágio um conjunto C que tenha no máximo r estruturas. Tal conjunto C consiste tanto de uma visão e alguns de seus índices quanto de um único índice cuja visão já tenha sido selecionada em algum estágio anterior. Em cada estágio, o conjunto C que possua o máximo benefício por unidade de espaço com relação ao conjunto de estruturas selecionadas nos estágios anteriores é escolhido. Por exemplo, o algoritmo *1-greedy* seleciona, a cada estágio, *uma* visão ou *um* índice associado a uma visão já escolhida que ofereça o maior benefício em termos do número de linhas processadas (modelo de custo linear). Embora o valor de r possa ser determinado de maneira arbitrária, variando de 1 ao número de estruturas que se deseja materializar, testes de desempenho realizados pelos autores comprovaram que, na prática, os algoritmos *r-greedy* não apresentam um aumento de desempenho significativo para valores de r maiores do que 4. Por fim, um algoritmo mais genérico, denominado *inner-level greedy*, é proposto. A principal diferença deste algoritmo é que ele não restringe o máximo de estruturas que podem ser selecionadas em cada estágio.

Por outro lado, o principal objetivo do trabalho de Shukla *et al*. é apresentar algoritmos que sejam mais simples (algoritmos PBS e PBS-U), porém mais rápidos do que o algoritmo *greedy*, mas que apresentem a mesma funcionalidade que este. Enquanto o tempo de execução do algoritmo *greedy* é de $O(k.n^2)$, onde k é o número de visões selecionadas e n é o número de vértices do grafo de derivação, o tempo de execução dos algoritmos propostos é de $O(n \log n)$. Ademais, para grafos de derivação que apresentam ordenação entre os tamanhos de duas visões adjacentes, o benefício do conjunto de visões selecionadas para serem materializadas por estes algoritmos é o mesmo que o garantido pelo algoritmo de Harinarayan *et al*.: pelo menos $(0{,}63 - f)$ vezes o benefício do conjunto ótimo de visões para a mesma quantidade de espaço, onde f é a fração do espaço disponível ocupado pela maior visão materializada.

Outra contribuição expressiva do trabalho de Shukla *et al*. é a extensão dos algoritmos anteriormente destacados (*greedy* e PBS) com a introdução da possibilidade de materialização de somente uma parte da visão, ao invés de sua materialização completa, como ocorre nestes algoritmos. Assim, as visões a serem materializadas são quebradas em blocos (*chunks*) e estes blocos passam a ser as unidades de materialização. Para tanto, os valores distintos para cada dimensão da visão são divididos em faixas de valores e os blocos são criados baseados nesta divisão. Quando uma consulta é realizada, o conjunto de blocos necessário para respondê-la é determinado. Dependendo do que foi selecionado para materialização, esse conjunto é dividido em duas partições: uma consistindo de blocos que foram materializados e outra consistindo de blocos que podem ser calculados em tempo de execução a partir de outros blocos materializados. Este esquema implica na necessidade de calcular somente blocos que estejam faltando para determinar o resultado de uma consulta, melhor do que calcular a consulta completamente. Os autores argumentam que a utilização de blocos como granularidade de materialização permite o projeto de algoritmos com custos associados ao cálculo da consulta mais baixos do que os algoritmos que assumem que a granularidade da materialização é a visão completa.

2.5.1.2 Materializando Visões com Relação ao Espaço de Armazenamento e à Manutenção

Em contrapartida aos trabalhos de Harinarayan *et al*. [HRU96], Gupta *et al*. [GHRU97] e Shukla *et al*. [SDN98], os trabalhos de Baralis *et al*. [BPT97] e Theodoratos e Sellis [TS97] determinam o conjunto de visões a serem materializadas de forma a minimizar os custos associados a ambos o cálculo da consulta e a manutenção das visões materializadas. Em [BPT97], o modelo de custo

não é completamente especificado, permitindo que uma grande variedade de modelos de custo sejam adaptados. Em especial, o custo da consulta é determinado pela freqüência na qual a consulta é realizada e por uma função que especifica o custo de calcular esta consulta a partir do conjunto de visões materializadas, ao passo que o custo de manutenção é determinado pela freqüência na qual a visão é modificada e por uma função que identifica seu custo de atualização. Em ambos os casos, modelos de custo podem ser adaptados às respectivas funções.

Assim como os algoritmos apresentados em [HRU96, GHRU97, SDN98], o algoritmo proposto por Baralis *et al*. é baseado no grafo de derivação. A principal idéia por trás deste algoritmo consiste em considerar somente as visões do grafo que, quando materializadas, oferecem alguma contribuição para a redução do custo total. De acordo com as consultas relevantes especificadas pelos usuários de SSD, o algoritmo inicialmente identifica um conjunto de visões candidatas, dentre as visões do grafo de derivação. Uma visão é candidata quando ela é associada diretamente a uma consulta ou quando ela pode ser utilizada para calcular duas outras visões candidatas. Uma vez desprezadas as visões não candidatas, o subgrafo de derivação gerado é percorrido de forma que as visões que não contribuam para a solução ótima sejam removidas. A heurística que determina quais visões candidatas devem ser eliminadas do subgrafo de derivação é baseada na estimativa do tamanho da materialização, a qual pode ser realizada seguindo técnicas de estimativas de consultas tradicionais já existentes. De acordo com a estimativa do tamanho das visões, é possível determinar quando a materialização de uma visão com nível de agregação de maior grau (menor nível de detalhe) oferece pouca ajuda para responder a consultas com relação à materialização da visão com menor nível de agregação.

Em [TS97], os autores definem o problema de configuração do *data warehouse* como um problema de otimização no qual o custo operacional total precisa ser minimizado, enquanto que todas as consultas de interesse a serem realizadas ao *data warehouse* devem ser respondidas usando-se exclusivamente o conjunto de visões selecionadas para serem materializadas. Dizer que todas as consultas de interesse devem ser respondidas usando-se exclusivamente as visões materializadas significa que, para cada consulta, deve existir uma reescrita completa desta em função do conjunto de visões a serem materializadas. Assim como em [BPT97], o custo operacional é a combinação do custo da consulta (determinado pela freqüência e pela importância relativa da consulta, além do plano de execução da consulta mais barato identificado por qualquer otimizador de consulta) com o custo de manutenção. No entanto, o custo de manutenção não considera somente a freqüência na qual a visão é modificada e o custo do cálculo incremental das atualizações nas visões, mas também o custo de se transmitir os dados necessários para a atualização.

Diferentemente dos trabalhos destacados anteriormente, os quais apresentam algoritmos baseados no grafo de derivação, o algoritmo proposto em [TS97] é baseado na representação das visões através de um grafo multiconsulta. Um grafo multiconsulta permite a representação compacta de várias consultas, através da junção dos grafos de cada uma. Dentro do escopo do trabalho, o qual considera visões/consultas relacionais que possuam apenas os operadores de seleção e de junção, os vértices do grafo multiconsulta representam o conjunto de todas as relações que aparecem em todas as visões. Já as suas arestas podem ser de dois tipos: arestas de junção e arestas de seleção. Uma aresta de junção entre dois vértices corresponde ao predicado de junção em uma visão que envolva os atributos de relações distintas. Arestas de seleção, por sua vez, são auto-arestas em um vértice, as quais correspondem ou ao predicado de seleção em uma visão envolvendo atributos de uma mesma relação ou ao caso no qual a visão é igual à relação.

Baseado nesta representação das visões usando um grafo multiconsulta, o algoritmo proposto modela o problema de configuração do *data warehouse* como um problema de busca

estado-espaço. Cada estado consiste de um grafo multiconsulta das visões a serem materializadas e da reescrita das consultas de interesse de acordo com estas visões, e possuem um custo operacional associado. As transições de um estado para outro são realizadas através de regras de transformação de estados (corte da aresta de seleção, corte da aresta de junção e junção da visão) e têm por objetivo gerar um novo grafo multiconsulta e reescrever completamente as consultas de acordo com o novo grafo. O algoritmo, o qual é estendido com heurísticas para melhorar o desempenho da busca, determina o estado que apresenta custo operacional mínimo e retorna tanto o conjunto de visões a serem materializadas quanto a reescrita das consultas dos usuários de SSD de acordo com este conjunto de visões.

2.5.1.3 Materializando Visões Dinamicamente

Os trabalhos descritos nas seções 2.5.1.1 e 2.5.1.2 adotam um paradigma estático de pré-selecionamento de um conjunto de visões a serem materializadas de acordo com a especificação de um conjunto de consultas comumente realizadas pelos usuários de SSD. Nenhum destes trabalhos considera a natureza dinâmica do problema de seleção, nem propõe soluções que possam ser adaptadas em tempo de execução à carga dos dados. No entanto, as análises dos usuários de SSD geralmente possuem uma natureza dinâmica. Conseqüentemente, o padrão de consultas a serem realizadas é dificilmente previsível, ficando a cargo do administrador do ambiente de *data warehousing* periodicamente redefinir as visões a serem materializadas. Surge então, a necessidade de se estudar a materialização dinâmica das visões.

Dentro deste contexto, o trabalho de Kotidis e Roussopoulos [KR99] descreve o sistema *DynaMat*. Este sistema materializa dinamicamente informação em vários níveis de agregação de acordo com a carga de trabalho submetida ao ambiente de *data warehousing*. Isto é realizado levando-se em consideração ambos o tempo gasto na atualização das visões e a disponibilidade de espaço de armazenamento. Para tanto, o sistema monitora constantemente as consultas dos usuários de SSD, e materializa o melhor conjunto de visões, adaptando-se rapidamente a novos padrões de consulta. O sistema possui duas fases: (i) *fase on-line*, na qual o sistema responde às consultas dos usuários de SSD e materializa os resultados destas consultas; e (ii) *fase de atualização*, na qual as atualizações recebidas dos provedores de informação são propagadas às visões materializadas. Essa última fase é geralmente realizada *off-line*. Em particular, o sistema *DynaMat* considera que os diversos níveis de agregação a serem materializados são armazenados em um banco de dados de informação dedicado, denominado *view pool*. A única exceção refere-se ao nível inferior, o qual representa o *data warehouse* propriamente dito.

Durante a *fase on-line*, os resultados das consultas dos usuários de SSD, chamados de fragmentos, vão sendo materializados no *view pool* de acordo com o grafo de derivação. Isto é realizado até que o limite de espaço de armazenamento disponível seja atingido. Intuitivamente, tais fragmentos correspondem a porções das visões do grafo de derivação ou às visões propriamente ditas. Quando não existe mais espaço disponível, o sistema adota uma política de substituição para substituir um fragmento anteriormente materializado pelo novo resultado da consulta, caso este último apresente maior benefício do que o primeiro. Esta política de substituição pode variar desde simplesmente não admitir mais resultados materializados no *view pool* até a utilização de técnicas mais elaboradas como LRU, FIFO ou heurísticas para decidir se o resultado é mais benéfico para o sistema. Kotidis e Roussopoulos afirmam que quatro diferentes políticas de substituição apresentaram bons resultados durante os testes do sistema. Estas políticas consideraram: (i) o tempo no qual o fragmento foi acessado pela última vez para responder à consulta, resultando em uma política de substituição LRU (*least recently used*), (ii) a freqüência de acesso do fragmento, resultando em uma política de substituição LFU (*least*

frequently used), (iii) o tamanho do resultado, medido em páginas de disco, referenciado como política de substituição SFF (*smaller fragment first*), e (iv) a razão de penalidade esperada pela recomputação do fragmento caso ele seja substituído, normalizado pelo seu tamanho atual, referenciado como política de substituição SPF (*smaller penalty first*).

Por outro lado, durante a *fase de atualização*, visões materializadas são eliminadas do *view pool* caso não sejam capazes de serem atualizadas dentro da janela de atualização (similar à janela noturna; seção 2.1.2). A decisão de quais fragmentos devem ser descartados também é realizada pela política de substituição, a qual escolhe o fragmento que apresenta o menor benefício para o sistema.

2.5.2 Manutenção Incremental das Visões Materializadas

Visões materializadas tornam-se inconsistentes sempre que as relações base são alteradas. Em um ambiente de *data warehousing*, como discutido na seção 2.2.2.2, após o carregamento inicial dos dados dos provedores de informação no *data warehouse*, estes provedores podem continuar a alterar os seus dados. Tais alterações devem, então, ser detectadas e propagadas ao ambiente, e incorporadas incrementalmente ao *data warehouse*, tanto aos dados do conjunto de visões que representa o nível inferior quanto aos dados das demais visões derivadas armazenadas.

O problema de manutenção incremental de visões tem sido estudado extensivamente, sendo que diversos algoritmos com este propósito têm sido desenvolvidos [GL95, GM95, Qua96]. Griffin e Libkin [GL95] propõem um algoritmo para a manutenção de visões materializadas que podem armazenar valores de instâncias duplicados. Para tanto, usam uma extensão natural das operações da álgebra relacional (*bag algebra*) como linguagem básica, considerando operadores de seleção, de projeção, de união, de subtração modificada (monus; subtrai multiplicidade), de interseção e de produto cartesiano. Resumidamente, para cada primitiva definida, os autores determinam uma expressão (equação) para a propagação de alterações. O conjunto de inserções e de remoções a serem propagadas para a visão materializada é determinado através da aplicação recursiva destas expressões.

Enquanto o trabalho de Griffin e Libkin apresenta as restrições de que a definição da visão pode incluir no máximo um operador de agregação e que a cláusula GROUP BY não é permitida, Quass [Qua96] estende o algoritmo em [GL95] para oferecer manutenção de visões com agregação de maneira genérica. Assim, a definição da visão pode incluir ambos diversos atributos em sua cláusula GROUP BY e diversos operadores de agregação. Similarmente a Griffin e Libkin, o trabalho de Quass é baseado na *bag algebra* e define um conjunto de equações de propagação de alterações para operadores de agregação. No entanto, os algoritmos abordados em [Qua96] apresentam expressões de manutenção para operadores de agregação que retornam resultados como modificações sempre que possível, ao invés de apenas conjuntos de inserção e remoção.

Por fim, o trabalho de Gupta e Mumick [GM95] classifica e compara diversas técnicas de manutenção incremental de visões materializadas propostas até então. Tal classificação é realizada em termos: (i) da informação disponível para a manutenção. Por exemplo, se todas ou apenas algumas relações base podem ser acessadas, ou se a visão materializada está ou não disponível; (ii) do conjunto de operadores que podem ser utilizados para a definição das visões materializadas; (iii) dos tipos de modificações dos dados das relações base que são considerados durante a manutenção. Em outras palavras, se as modificações são manipuladas diretamente ou são modeladas como remoções seguidas de inserções; e (iv) do fato de que os algoritmos propostos podem ser aplicados uniformemente a todas as instâncias do banco de dados e a todas

as instâncias das modificações. Ou seja, se a manutenção das visões depende apenas das instâncias particulares do banco de dados e das modificações que ocorreram nas relações base.

Os trabalhos acima abordados propõem algoritmos que são projetados para ambientes de banco de dados tradicionais, nos quais se assume que a manutenção de visões é realizada por um sistema que tem total controle e conhecimento das definições de visões, das relações base, das alterações e da visão propriamente dita. Em outras palavras, ambos os dados base e as visões materializadas residem no mesmo sistema de banco de dados. No entanto, o problema de manutenção incremental de visões em ambientes de *data warehousing* apresenta complexidade adicional, devido ao desacoplamento existente entre os provedores de informação (base de dados), que são autônomos, e a máquina de definição e de manutenção de visões (componente de integração e manutenção). Por exemplo, durante a manutenção da visão materializada, pode ser necessário que as alterações realizadas nos provedores sejam integradas com dados provenientes de outros provedores, antes de serem armazenadas definitivamente no *data warehouse*. Durante este processamento, outras atualizações podem ter ocorrido, gerando inconsistências nos dados integrados (anomalias de inserção e de remoção).

Essa seção ilustra, inicialmente, exemplos de anomalias de inserção e de remoção plausíveis de ocorrerem em ambientes de *data warehousing* (seção 2.5.2.1). A seguir são descritos trabalhos voltados à manutenção de visões materializadas sob dois aspectos: (i) quando os dados dos provedores de informação são acessados durante a manutenção; seção 2.5.2.2; e (ii) quando a visão pode ser mantida sem a necessidade de acesso aos provedores de informação; seção 2.5.2.3. A abrangência do problema de manutenção incremental de visões materializadas é destacada na seção 2.5.2.4.

2.5.2.1 Anomalias de Inserção e de Remoção em Ambientes de *Data Warehousing*

Os exemplos de anomalias de inserção e de remoção ilustrados nessa seção são baseados nos trabalhos de Zhuge *et al*. [ZGMHW95, ZWGM97, ZGMW98] e Garcia-Molina *et al*. [GMLWZ98]. Como conseqüência, utilizam a mesma notação destes e são descritos em termos do modelo relacional e da álgebra relacional. Apesar de utilizarem definições de visão em álgebra relacional, os exemplos assumem que as visões materializadas podem armazenar valores de instâncias duplicados. Além disto, a notação `insert(r,t)` é utilizada para denotar a inserção da tupla `t` em uma relação `r`. De maneira similar, a notação `delete(r,t)` denota a remoção da tupla `t` de uma relação `r`.

Nos exemplos, a atualização das visões materializadas é realizada da seguinte forma. Cada provedor de informação primeiro executa a operação de atualização (ou seja, a operação de inserção ou de remoção) em suas relações base. Em seguida, esta operação é enviada, através do módulo associado ao provedor responsável pela detecção e propagação de alterações, ao ambiente de *data warehousing*. O ambiente, assim que recebe a notificação da operação, produz uma consulta a ser enviada ao provedor, com a finalidade de calcular o novo resultado da visão materializada. Esta consulta é chamada de consulta de manutenção. O provedor, ao receber esta consulta de manutenção, calcula o seu resultado, retornando-o ao *data warehousing*, o qual finalmente atualiza a visão materializada.

A seção 2.5.2.1.1 exibe a atualização correta da visão materializada. A seção 2.5.2.1.2, por sua vez, ilustra a anomalia de inserção para situações nas quais as relações que fazem parte da visão materializada são provenientes de um mesmo provedor de informação. Já a seção 2.5.2.1.3 ilustra a situação de anomalia na qual uma visão materializada é definida em termos de relações

base que pertencem a provedores de informação distintos. Por fim, a seção 2.5.2.1.4 discute anomalias que consideram várias visões materializadas definidas em termos de relações base comuns.

2.5.2.1.1 Manutenção Incremental Correta da Visão

A Tabela 2.2 exibe duas relações base presentes em um provedor de informação remoto P: r_1 (w, x) e r_2 (x, y). Adicionalmente, para esse provedor de informação, a visão V no *data warehouse* é definida pela seguinte expressão da álgebra relacional: $V = \Pi_w (r_1 \bowtie r_2)$. Inicialmente, a visão materializada VM contém uma única tupla: [1].

r_1	
w	x
1	2

r_2	
x	Y
2	4

VM
w
1

Tabela 2.2 Instâncias das relações r_1, r_2 e VM.

Se a tupla [2,3] for inserida na relação r_2, os seguintes eventos podem ocorrer:

- P executa U_1 = insert(r_2,[2,3]) e notifica o ambiente;
- o ambiente recebe U_1 e, para manter a visão materializada incrementalmente, envia a consulta de manutenção $Q_1 = \Pi_w (r_1 \bowtie [2,3])$ para P;
- P recebe Q_1 e calcula o seu resultado utilizando as relações base correntes, retornando o resultado A_1=([1]) ao ambiente; e
- o ambiente recebe o resultado A_1 e adiciona ([1]) a VM, obtendo ([1],[1]).

Neste exemplo, a visão materializada final é correta, ou seja, é equivalente ao que seria obtido através da utilização de um algoritmo de manutenção incremental convencional aplicado diretamente ao provedor de informação.

2.5.2.1.2 Anomalia de Inserção – um único Provedor de Informação

Considerando a mesma visão materializada do exemplo anterior e a mesma definição das relações r_1 e r_2, porém com instâncias diferentes (Tabela 2.3), se a tupla [2,3] for inserida na relação r_2 e a tupla [4,2] for inserida em r_1, os seguintes eventos são plausíveis de ocorrerem:

r_1	
w	x
1	2

r_2	
x	y

VM
w

Tabela 2.3 Instâncias das relações r_1, r_2 e VM.

- P executa U_1 = insert(r_2,[2,3]) e notifica o ambiente;
- o ambiente recebe U_1 e envia a consulta de manutenção $Q_1 = \Pi_w (r_1 \bowtie [2,3])$ para P;
- P executa U_2 = insert(r_1,[4,2]) e notifica o ambiente;
- o ambiente recebe U_2 e envia a consulta de manutenção $Q_2 = \Pi_w ([4,2] \bowtie$) para P;

- P recebe Q_1 e calcula o seu resultado utilizando as relações base correntes: r_1 = ([1,2],[4,2]) e r_2 = ([2,3]), retornando o resultado A_1 = ([1],[4]) ao ambiente;
- o ambiente recebe o resultado A_1 e adiciona-o a VM, obtendo ([1],[4]);
- P recebe Q_2 e calcula o seu resultado utilizando as relações base correntes: r_1 e r_2, retornando o resultado A_2 = ([4]) ao ambiente; e
- o ambiente recebe o resultado A_2 e adiciona-o a VM, obtendo ([1],[4],[4]).

Nesse exemplo, as relações base do provedor de informação continuaram a ser alteradas antes que U_1 fosse refletida no *data warehouse*. Isto faz com que a resposta A_1 seja calculada utilizando-se um estado diferente daquele que existia no momento no qual U_1 foi realizada. Conseqüentemente, a visão materializada final é incorreta. As mesmas discussões realizadas nessa seção podem ser aplicadas para anomalias de remoção.

2.5.2.1.3 Anomalia de Atualização – uma Visão, vários Provedores de Informação

A Tabela 2.4 mostra as relações base r_1 (w, x), r_2 (x, y) e r_3 (y, z), as quais são armazenadas respectivamente nos provedores de informação P_1, P_2 e P_3. Para esse exemplo, a visão V no *data warehouse* é definida pela seguinte expressão da álgebra relacional: $V = (r_1 \bowtie r_2 \bowtie r_3)$. Inicialmente, a visão materializada VM é vazia ().

r_1	
w	x
1	2

r_2	
x	y

r_3	
y	z
3	4

VM			
w	x	y	z

Tabela 2.4 Instâncias das relações r_1, r_2, r_3 e VM.

Se a tupla [2,3] for inserida na relação r_2 e a tupla [1,2] for removida de r_1, os seguintes eventos podem ocorrer:

- P_2 executa U_1 = insert(r_2,[2,3]) e notifica o ambiente;
- o ambiente recebe U_1 e gera a consulta de manutenção $Q_1 = r_1 \bowtie [2,3] \bowtie r_3$; para calcular Q_1, o ambiente envia a subconsulta $Q_1^1 = r_1 \bowtie [2,3]$ para P_1;
- o ambiente recebe A_1^1 = [1,2,3] de P_1; e envia a subconsulta $Q_1^2 = [1,2,3] \bowtie$ para P_3;
- P_1 executa U_2 = delete(r_1,[1,2]) e notifica o ambiente; uma vez que a relação r_1 passa a ser vazia, o ambiente não executa nenhuma atividade (não gera a consulta de manutenção);
- o ambiente recebe A_1^2 = [1,2,3,4] de P_3, que representa a resposta final para Q_1; e
- o ambiente adiciona A_1 a VM, obtendo ([1,2,3,4]).

O resultado final da visão materializada é incorreto, uma vez que a intercalação da consulta de manutenção Q_1 com as atualizações simultâneas de outros provedores de informação gera inconsistências.

2.5.2.1.4 Anomalia de Atualização – várias Visões

Usando como base as relações r_1 (w, x), r_2 (x, y) e r_3 (y, z), para duas visões no *data warehouse* (Tabela 2.5), definidas como $V_1 = (r_1 \bowtie r_2)$ e $V_2 = (r_2 \bowtie r_3)$, a seguinte seqüência pode ocorrer:

- no tempo t_0, a relação r_2 é vazia. Assim, ambas VM_1 e VM_2 são vazias;
- no tempo t_1, U_1 = insert(r_2,[2,3]) é executado;
- no tempo t_2, U_1 é computada para VM_1, através da junção desta tupla com as tuplas da relação r_1, sendo seu resultado inserido em VM_1; e
- no tempo t_3, U_1 é computada para VM_2, através da junção desta tupla com as tuplas da relação r_3, sendo seu resultado inserido em VM_2.

	r_1		r_2		r_3		VM_1			VM_2		
tempo	w	x	x	y	y	z	w	x	y	x	y	z
t_0	1	2			3	4						
t_1	1	2	2	3	3	4						
t_2	1	2	2	3	3	4	1	2	3			
t_3	1	2	2	3	3	4	1	2	3	2	3	4

Tabela 2.5 Instâncias das relações r_1, r_2, r_3, VM_1 e VM_2.

Para ambas as visões materializadas VM_1 e VM_2 os resultados são corretos. No entanto, após o tempo t_2, VM_1 reflete o novo estado da relação r_2. Em contrapartida, a visão materializada VM_2 não reflete este estado. Assim, estas duas visões são inconsistentes entre si após t_2 e antes de t_3.

2.5.2.2 Mantendo Visões Acessando os Provedores de Informação Originais

Apesar dos exemplos discutidos na seção 2.5.2.1 serem baseados principalmente em visões tradicionais definidas em termos dos operadores de projeção e de junção, a manutenção incremental de visões que possuam qualquer operação que requeira informação adicional de um ou mais provedores de informação está sujeita às mesmas anomalias. Para cada uma das situações de anomalia descritas nas seções 2.5.2.1.2 a 2.5.2.1.4, os trabalhos de Zhuge *et al*. [ZGMHW95, ZWGM97, ZGMW98] apresentam algoritmos que garantem a consistência das visões materializadas, eliminando tais anomalias.

Estes algoritmos, em particular, são implementados no sistema WHIPS (*Warehouse Information Project at Stanford*) [Wid95, HGMW+95, WGL+96, LZW+97]. Este sistema enfoca a implementação do componente de integração e manutenção da arquitetura típica de um ambiente de *data warehousing*, tendo, como um dos seus principais compromissos, a manutenção da consistência dos dados integrados de diferentes provedores de informação. A arquitetura do sistema WHIPS é composta por um conjunto de módulos, os quais possuem funcionalidades específicas. Isto permite que o sistema suporte diferentes provedores de informação, distintos projetos de *data warehouse* e várias formas de gerenciamento de visões. Por exemplo, a cada visão materializada definida no sistema é associado um módulo *gerenciador de visão*, o qual é responsável por manter a consistência dos dados manipulados por aquela visão utilizando qualquer um dos algoritmos propostos em [ZGMHW95, ZGMW98], ao passo que os algoritmos propostos em [ZWGM97] são implementados pelo módulo *processo de junção*. Uma descrição detalhada do sistema WHIPS é realizada na seção 3.7.

Dentro deste contexto, a seção 2.5.2.2.1 descreve o algoritmo ECA, responsável por atualizar visões materializadas definidas em termos de um único provedor de informação. Já a

seção 2.5.2.2.2 descreve os algoritmos *Strobe*, voltados à atualização incremental de visões definidas em termos de provedores de informação distintos. A atualização de várias visões materializadas correlacionadas é abordada na seção 2.5.2.2.3.

2.5.2.2.1 Manutenção Incremental Correta da Visão – um único Provedor de Informação

Visando solucionar os problemas de anomalia em situações nas quais apenas um provedor de informação é considerado (seção 2.5.2.1.2), o trabalho de Zhuge *et al*. [ZGMHW95] propõe o algoritmo ECA (*Eager Compensating Algorithm*). Este algoritmo permite que os provedores de informação continuem a atualizar os dados de suas relações base ao mesmo tempo que a visão materializada correspondente no *data warehouse* é atualizada incrementalmente. Dois tipos de operação de atualização são manipulados pelo ECA: inserção e remoção. Modificações são tratadas como remoções seguidas de inserções. O algoritmo ECA também assume que: (i) a atualização da visão materializada é imediata, ou seja, ocorre após cada atualização dos dados das relações base; (ii) a seqüência na qual as notificações de operação são recebidas pelo ambiente de *data warehousing* é a mesma na qual estas são processadas; e (iii) o provedor de informação manipula ambas as operações de atualização às suas relações base e as consultas realizadas pelo ambiente de forma atômica.

A idéia básica do algoritmo ECA consiste em adicionar consultas de compensação às consultas enviadas pelo ambiente de *data warehousing* ao provedor de informação devido à notificação de uma operação de atualização U nas relações base deste provedor. Estas consultas de compensação têm por objetivo compensar o efeito de outras operações de atualização significativas que porventura tenham sido executadas no provedor após o recebimento da notificação de U pelo *data warehousing*, mas antes que esta operação tenha sido refletida na visão materializada.

Assim, o comportamento do provedor de informação continua a ser o mesmo que o descrito na seção 2.5.2.1: (i) para cada operação de atualização em suas relações base, o provedor primeiro executa esta operação e em seguida notifica o ambiente; e (ii) para cada consulta de manutenção recebida do ambiente, o provedor calcula o seu resultado e retorna-o ao *data warehousing*. Por outro lado, o comportamento do ambiente de *data warehousing* é aprimorado para oferecer suporte a consultas de compensação. Para tanto, o ambiente armazena todas as consultas de manutenção que foram enviadas ao provedor e que ainda não foram respondidas (consultas não respondidas). Quando o ambiente recebe a notificação de uma operação de atualização no provedor, este gera uma consulta de manutenção correspondente e incorpora a esta uma consulta de compensação para cada consulta não respondida. Esta consulta resultante é, então, enviada ao provedor. Em contrapartida, quando o ambiente recebe a resposta às consultas realizadas ao provedor, este armazena estas respostas intermediárias em uma relação temporária até que todos os resultados das consultas de compensação tenham sido obtidos, ou seja, até que não existam mais consultas não respondidas. Somente após esta situação é que ocorre a atualização propriamente dita da visão materializada.

Para mostrar como o algoritmo ECA trabalha, a discussão a seguir é baseada no exemplo introduzido na seção 2.5.2.1.2. Se, da mesma forma que nesse exemplo, a tupla [2,3] for inserida na relação r_2 e a tupla [4,2] for inserida em r_1, os seguintes eventos podem ocorrer:

- P executa $U_1 = \text{insert}(r_2,[2,3])$ e notifica o ambiente;

- o ambiente recebe U_1 e envia a consulta de manutenção $Q_1 = \Pi_w (r_1 \bowtie [2,3])$ para P (nessa situação, Q_1 é a mesma consulta que a do referido exemplo, uma vez que ainda não existem consultas não respondidas);
- P executa $U_2 = \text{insert}(r_1,[4,2])$ e notifica o ambiente;
- o ambiente recebe U_2 e envia a consulta $Q_2 = \Pi_w ([4,2] \bowtie r_2) - \Pi_w ([4,2] \bowtie [2,3])$ para P (em especial, a primeira parte de Q_2 é idêntica à consulta do exemplo em questão, enquanto que a segunda parte de Q_2 corresponde à consulta de compensação, uma vez que existe uma consulta não respondida – Q_1);
- P recebe Q_1 e calcula o seu resultado utilizando as relações base correntes: $r_1 = ([1,2],[4,2])$ e $r_2 = ([2,3])$, retornando o resultado $A_1 = ([1],[4])$ ao ambiente;
- o ambiente recebe o resultado A_1 e armazena-o na relação temporária;
- P recebe Q_2 e calcula o seu resultado utilizando as relações base correntes: r_1 e r_2, retornando o resultado $A_2 = \varnothing$ ao ambiente;
- o ambiente recebe o resultado A_2 e armazena-o na relação temporária;
- como não existem mais consultas não respondidas, o ambiente atualiza VM, obtendo as tuplas corretas ([1],[4]).

Zhuge *et al*. [ZGMHW95] também discutem otimizações que poderiam ser aplicadas ao algoritmo ECA caso a visão incluísse uma chave para cada relação base envolvida. Neste caso, operações de remoção poderiam ser manipuladas diretamente pelo ambiente de *data warehousing*, sem enviar consultas de manutenção ao provedor de informação. Por outro lado, operações de inserção gerariam consultas de manutenção. Entretanto, consultas de compensação não seriam necessárias.

2.5.2.2.2 Manutenção Incremental Correta da Visão – uma Visão, vários Provedores de Informação

A manutenção incremental correta de visões materializadas que são definidas em termos de relações base que pertencem a provedores de informação distintos é garantida pelos algoritmos da família *Strobe* [ZGMW98]. Estes algoritmos, além de permitirem a atualização simultânea dos dados dos provedores de informação e da visão materializada no *data warehouse* e de manipularem somente operações de inserção e de remoção como tipos de operações de atualização (assim como o algoritmo ECA), apresentam as seguintes características adicionais: oferecem diferentes cenários de transação de atualização, oferecem diferentes níveis de consistência e combinam as operações de atualização realizadas nos provedores de informação em ações atômicas a serem aplicadas à visão materializada.

Existem três cenários de transação de atualização oferecidos pelos algoritmos da família *Strobe*. O primeiro e mais simples destes três cenários é chamado de transação de atualização única. Neste cenário, cada operação de inserção ou de remoção executada no provedor de informação representa uma única transação, a qual é notificada ao ambiente de *data warehousing* individualmente. $T = \text{insert}(r_2,[2,3])$ é um exemplo de uma transação de atualização única. Por outro lado, no cenário de transação local ao provedor, uma única transação representa uma seqüência de operações de inserção ou de remoção executadas em um mesmo provedor de informação ($T = <\text{delete}(r_1,[1,2]), \text{insert}(r_1,[3,4])>$). Por fim, o cenário no qual uma transação representa um conjunto de operações de inserção e de remoção executadas em vários provedores de informação ($T = <\text{delete}(r_1,[1,2]), \text{delete}(r_3,[3,4])>$) é chamado de transação global. Em ambos os

cenários de transação local e de transação global, a seqüência de operações que compõem a transação deve ser aplicada de maneira atômica ao *data warehouse*. Além disto, cada provedor possui um escalonamento serializável local de todas as suas transações locais, e existe uma ordem de serialização global das transações globais.

A família de algoritmos *Strobe* oferece quatro níveis de consistência para as visões materializadas do *data warehouse*: convergência, consistência fraca, consistência forte e completude. A convergência representa o nível de consistência mais flexível, e garante que a visão materializada eventualmente estará consistente com os dados dos provedores de informação, mesmo que esta porventura esteja temporariamente inconsistente. Em outras palavras, considerando-se uma execução finita, após a última execução de operação de atualização nos provedores e após o encerramento das atividades correspondentes executadas pelo ambiente de *data warehousing*, a visão materializada é consistente com as relações dos provedores de informação. Por outro lado, a completude é o nível mais rígido, exigindo que a visão materializada esteja sempre consistente com os dados dos provedores. Por fim, os dois níveis de consistência remanescentes consideram que: (i) o estado da visão materializada representa o seu conteúdo, o qual muda sempre que a visão é atualizada; (ii) o estado de um provedor de informação representa o conteúdo das relações base deste provedor, o qual muda sempre que alguma operação de atualização é a ele submetida; e (iii) existe um escalonamento serial global que representa a execução das transações através de todos os provedores. Com base nessas considerações, as consistências fraca e forte garantem que o estado da visão materializada reflete um conjunto de estados válidos dos provedores. No entanto, na consistência fraca cada provedor pode possuir um escalonamento serializável diferente e a visão materializada pode refletir um conjunto diferente de transações consideradas para cada provedor, ao passo que na consistência forte cada estado da visão materializada reflete o mesmo escalonamento serializável global, e a ordem de seus estados é a mesma que a ordem das operações de atualização realizadas nos provedores.

Cada algoritmo da família *Strobe* obtém um nível específico de consistência para um dos três cenários de transação de atualização. O algoritmo *Strobe*, o mais simples, garante consistência forte para transações de atualização única e convergência para transações locais ao provedor e para transações globais. O algoritmo *Transaction-Strobe* (*T-Strobe*) garante consistência forte para transações locais ao provedor e consistência fraca para transações globais, ao passo que o algoritmo *Global-Strobe* (*G-Strobe*) garante consistência forte para transações globais. Por fim, o algoritmo *Complete-Strobe* certifica completude para transações de atualização única e, quando associado a *T-Strobe* (*C-TStrobe*) e *G-Strobe* (*C-GStrobe*), certifica completude para transações locais ao provedor e para transações globais, respectivamente.

No entanto, todos os algoritmos são baseados nos mesmos conceitos, os quais são descritos a seguir. Similarmente ao algoritmo ECA, os algoritmos da família *Strobe* processam as operações de atualização assim que as suas notificações são recebidas pelo ambiente de *data warehousing*, enviando, quando necessário, consultas de manutenção aos provedores de informação. No entanto, tais consultas, ao invés de serem enviadas a apenas um único provedor de informação, são enviadas a vários provedores, os quais armazenam as relações que participam da visão (seção 2.5.2.1.3). Para tanto, os algoritmos *Strobe* processam a consulta de manutenção por partes, obtendo resultados intermediários para cada parte até que a consulta seja totalmente respondida. Para cada parte da consulta de manutenção, os algoritmos determinam qual o próximo provedor de informação a receber parte da consulta e qual a porção da consulta que pode ser executada neste provedor. Outra característica em comum entre o algoritmo ECA e os algoritmos *Strobe* é que estes últimos também mantêm informação de todas as operações de atualização que ocorrem durante o cálculo da consulta de manutenção, e compensam estas

operações posteriormente. No entanto, ao invés de compensar tais operações através de consultas de compensação, os algoritmos *Strobe* mantêm, para cada consulta de manutenção, um conjunto de operações pendentes de atualização que ocorreram enquanto a consulta estava sendo processada. Somente depois do processamento completo da consulta nos provedores de informação o ambiente compensa o resultado obtido baseado no conjunto de operações pendentes para aquela consulta. Finalmente, diferentemente dos algoritmos ECA, os algoritmos *Strobe* assumem que as visões materializadas no *data warehouse* incluem as chaves primárias de todas as relações base envolvidas. Assim, o ambiente apenas gera as consultas de manutenção quando as operações de inserção são a ele notificadas.

Apesar dos algoritmos *Strobe* processarem as operações de atualização assim que as suas notificações são recebidas pelo ambiente, estas operações não são refletidas imediatamente na visão materializada, e sim armazenadas em uma lista de ações a serem realizadas na visão. Somente quando a aplicação de todas as ações desta lista levar a visão materializada de um estado consistente para outro estado consistente (quando não existirem mais consultas não respondidas e quando todas as atualizações recebidas forem processadas) é que a visão materializada é devidamente atualizada, de forma atômica, em uma única transação. Assim, quando o ambiente de *data warehousing* recebe a notificação de uma operação de remoção, ele gera uma ação de remoção para as tuplas correspondentes na visão materializada e adiciona esta ação à lista de ações. Por outro lado, quando uma operação de inserção é notificada, o ambiente gera a consulta de manutenção correspondente e armazena as operações pendentes. Após a resposta a esta consulta ser compensada com as respectivas operações pendentes, uma ação de inserção na visão materializada é gerada e adicionada à lista de ações. A visão materializada é atualizada de acordo com esta lista de ações.

2.5.2.2.3 Manutenção Incremental Correta de Várias Visões

O problema de garantir a consistência dos dados armazenados no *data warehouse* pode ser observado sob dois aspectos diferentes: (i) manter cada visão materializada consistente com os dados dos provedores de informação; e (ii) providenciar que várias visões definidas em termos de relações base comuns sejam mutuamente consistentes (seção 2.5.2.1.4). Enquanto que os algoritmos acima discutidos enfocam o primeiro aspecto, Zhuge *et al*. [ZWGM97] propõem algoritmos que trabalham em conjunto com estes para garantir a consistência de várias visões que acessam as mesmas porções de dados dos provedores.

De maneira geral, em [ZWGM97], Zhuge *et al*. permitem que cada visão materializada no *data warehouse* seja mantida incrementalmente de acordo com os algoritmos ECA ou da família *Strobe*. Em especial, diferentes visões materializadas podem utilizar diferentes algoritmos de manutenção incremental. No entanto, ao invés dos algoritmos ECA ou *Strobe* atualizarem as visões materializadas correspondentes, eles geram listas de ações a serem executadas em cada uma das visões materializadas. Tais listas de ações são armazenadas pelos algoritmos propostos, os quais aguardam até que todas as listas correspondentes às visões que compartilham porções de dados sejam recebidas. Somente quando todas as visões afetadas puderem ser modificadas conjuntamente, as alterações são refletidas no *data warehouse*, em uma única transação, de forma atômica.

No exemplo da seção 2.5.2.1.4, as alterações em VM_1 no tempo t_2 não são diretamente realizadas. Ao contrário, tais alterações são armazenadas até que as alterações correspondentes a VM_2 sejam recebidas no tempo t_3. Somente depois disto ocorre a atualização do *data warehouse* em uma única transação.

2.5.2.3 Mantendo Visões sem Acessar os Provedores de Informação Originais

Como pode ser observado através dos exemplos abordados na seção 2.5.2.1, a manutenção de visões materializadas em um ambiente de *data warehousing* requer, na maioria dos casos (exceto para visões muito simples), acesso aos dados que não estão disponíveis nas visões propriamente ditas, mas sim nas relações base a partir das quais as visões materializadas são definidas. Nestas situações, nas quais os provedores de informação são acessados para obter informações adicionais a serem utilizadas na manutenção, os algoritmos de Zhuge *et al*. [ZGMHW95, ZWGM97, ZGMW98] podem ser utilizados para manter a consistência correta das visões materializadas. No entanto, muitas vezes os provedores podem estar temporariamente indisponíveis, ou pode ser caro ou demorado consultá-los.

Assim, reduzir ao máximo ou simplesmente não realizar o acesso aos provedores de informação que participam do ambiente de *data warehousing* pode ser uma solução viável nestas situações críticas. Uma visão materializada é considerada dotada de automanutenção com relação a uma operação de modificação (inserção, remoção, atualização) se pode ser mantida incrementalmente sem acessar as relações base por ela referenciadas em resposta a todas as instâncias de uma alteração do tipo indicado nas relações base. Mais especificadamente, as alterações a serem refletidas na visão materializada podem ser calculadas usando somente a visão materializada propriamente dita e as alterações realizadas nos provedores de informação. Por exemplo, Gupta e Mumick [GM95] argumentam que visões definidas em termos dos operadores relacionais de seleção, de projeção e de junção não são dotadas de automanutenção com respeito a inserções, mas geralmente são dotadas de automanutenção com respeito a remoções e a modificações, em situações nas quais restrições de chave são disponíveis.

Quando uma visão materializada não é dotada de automanutenção e o acesso aos provedores de informação é inviável, deve-se utilizar informação adicional para atualizar esta visão de forma consistente sem acessar as relações base. Uma solução trivial para este problema consiste na replicação completa no *data warehouse* de todas as relações base referenciadas pela visão materializada. Conseqüentemente, algoritmos tradicionais de manutenção de visões [GL95, GM95, Qua96] poderiam ser utilizados, uma vez que ambas as relações base e a visão materializada estariam armazenadas no mesmo sistema. Esta solução, embora garanta que a visão materializada seja dotada de automanutenção no *data warehouse*, introduz diversos problemas adicionais. Em primeiro lugar, cópias de todas as relações base referenciadas por todas as visões materializadas devem ser armazenadas no *data warehouse*, utilizando considerável espaço de armazenamento adicional. Ademais, nem sempre todas as informações correspondentes a todas as relações base são necessárias para atualizar as visões materializadas. Outro problema refere-se ao fato de que, muitas vezes, a replicação completa dos dados dos provedores de informação não é permitida. Por fim, a redundância em grande escala pode conduzir a problemas de inconsistência.

Dentro deste contexto, Quass *et al*. [QGMW96] propõem um algoritmo que, dado uma visão materializada no *data warehouse* definida em termos dos operadores de seleção, de projeção e de junção, determina um conjunto de visões auxiliares a serem armazenadas de tal forma que ambas a visão materializada e as visões auxiliares sejam dotadas de automanutenção, sem acessar os dados base nos provedores de informação. O algoritmo proposto leva em consideração restrições de chave e de integridade referencial para reduzir o tamanho das visões auxiliares. Não existe, portanto, a necessidade de replicação de todas as relações base na sua totalidade no *data warehouse*. Para tanto, o algoritmo inicialmente constrói um grafo de junção para a visão materializada em questão. Esse grafo direcionado $G(V) = <R,\varepsilon>$ é formado por um conjunto de

vértices, os quais representam as relações base R referenciadas por V, e por um conjunto de arestas, de tal forma que existe uma aresta direcionada $e(R_i, R_j) \in \varepsilon$ de R_i para R_j se V contém uma condição de junção $R_i.B = R_j.A$ e A é a chave de R_j. Utilizando este grafo, o algoritmo identifica o conjunto de relações das quais uma relação R_i depende (isto é, $Dep(R_i,G)$), a cláusula transitiva deste conjunto (ou seja, $Dep+(R_i,G)$), e o conjunto de relações com as quais a relação R_i precisa ser combinada de tal forma que a chave de uma destas relações seja preservada na visão (isto é, $Need(R_i,G)$). Para cada relação $R_i \in R$, o algoritmo verifica se $Dep+(R_i,G)$ inclui todas as demais relações de R e R_i não pertence ao conjunto $Need(R_j,G)$ para qualquer relação $R_j \in R$. Se esta situação é verdadeira, então o algoritmo não armazena qualquer parte de R_i com a finalidade de manter V. Caso contrário, o algoritmo determina a visão auxiliar a ser materializada no *data warehouse*, de acordo com o conjunto $Dep(R_i,G)$ e as demais visões auxiliares já armazenadas.

2.5.2.4 Abrangência do Problema de Manutenção Incremental de Visões Materializadas

Os exemplos e trabalhos discutidos ao longo dessa seção ilustram a abrangência e as dificuldades do problema de manutenção incremental de visões materializadas. Conseqüentemente, esse tem sido um tópico de grande interesse na área acadêmica, sendo que diversos outros trabalhos além dos já citados têm sido propostos com os mais variados objetivos.

Quass e Widom [QW97] propõem um mecanismo de controle de concorrência que permite que ambas as consultas dos usuários de SSD e as transações de atualização sejam executadas de forma concorrente. O algoritmo é chamado de 2VNL (*two-version no locking*), desde que até duas versões de cada instância são simultaneamente disponíveis e não ocorre bloqueio dos dados. Em especial, 2VNL garante que um usuário manipule um estado consistente do *data warehouse* através de toda a sua sessão de consulta, a qual pode incluir, por exemplo, um conjunto seqüencial de consultas que fazem parte de um processo *drill-down*.

Por outro lado, os trabalhos de Yang e Widom [YW98, YW00] permitem que os usuários de SSD realizem consultas temporais ao *data warehouse*, através do oferecimento de visões temporais sobre provedores de informação não históricos. Tais visões temporais são mantidas, de maneira incremental, tanto quando os dados dos provedores de informação são alterados (propagação de alterações) quanto quando o tempo corrente do banco de dados temporal avança. Uma vez que os provedores de informação não são temporais, o conceito de visão dotada de automanutenção é essencial para esse trabalho.

Outro artigo que enfoca a utilização de visões dotadas de automanutenção é o de Huyn [Huy97]. Diferentemente de Quass *et al*. [QGMW96], que se preocupam com a determinação de um conjunto de visões auxiliares tal que essas e a visão materializada sejam dotadas de automanutenção, Huyn endereça dois problemas relacionados: determinar se uma visão é dotada de automanutenção e como manter esta visão. Os algoritmos propostos visam minimizar o acesso externo a provedores de informação e abordam situações nas quais várias visões materializadas são armazenadas no *data warehouse*.

Enfocando uma implementação ROLAP do *data warehouse*, o trabalho de Hurtado *et al*. [HMV99] aborda o problema de manutenção incremental de visões materializadas frente a alterações nas tabelas de dimensão. Tais alterações podem afetar tanto a estrutura das hierarquias de relacionamento de atributos das dimensões quanto os domínios dos atributos destas dimensões. A introdução de um novo nível de agregação em uma hierarquia de relacionamento de atributos ou a definição de um relacionamento entre dois níveis independentes pertencentes à mesma dimensão são exemplos de alterações estruturais, ao passo que alterações de domínio

referem-se à inserção e à remoção de tuplas nas relações. Os algoritmos propostos para a manutenção das visões materializadas são baseados em um conjunto de operadores básicos que permitem a especificação das alterações consideradas. Em especial, a metodologia descrita para a manutenção incremental com relação aos domínios dos atributos é baseada no grafo de derivação, uma vez que utiliza o relacionamento existente entre as diversas visões materializadas para calcular as alterações a serem realizadas nestas visões, além de considerar que as visões são dotadas de automanutenção.

Em contrapartida, dado que trabalhos tais como os de Griffin e Libkin [GL95] e Quass [Qua96] desenvolvem expressões padrão para manter incrementalmente uma grande variedade de visões materializadas, Labio *et al*. [LYGM99] argumentam que podem existir inúmeras alternativas para se realizar a atualização das visões utilizando essas expressões de manutenção padrão. Estas estratégias geralmente implicam em diferentes quantidades de trabalho e conduzem a diferentes janelas de atualização (período no qual as visões materializadas são mantidas). Assim, o trabalho de Labio *et al*. visa o oferecimento de técnicas que permitam a obtenção de estratégias de atualização corretas e eficientes considerando uma única visão ou várias visões materializadas, minimizando o trabalho necessário para executar uma expressão de atualização e diminuindo, conseqüentemente, a janela de atualização.

Mais recentemente, Chan *et al*. [CLS00] enfocam o problema de manutenção de visões materializadas em um ambiente baseado na Web. Os algoritmos propostos visam, em especial, a manutenção incremental de visões definidas em termos dos operadores *sum*, *count*, *avg*, *min* e *max* da linguagem SQL. Uma vez que o ambiente sendo considerado é baseado na Web, os autores dividem o processo de manutenção em dois processos independentes, porém cooperativos. O primeiro processo, chamado de processo de propagação, manipula as alterações relevantes nas tabelas base de forma a determinar quais alterações devem ser aplicadas ao *data warehouse*. No entanto, a aplicação propriamente dita dessas alterações no *data warehouse* somente é realizada pelo segundo processo, o processo de atualização. Tal estratégia objetiva minimizar a carga de trabalho do *data warehouse* durante o processo de manutenção, contribuindo para um aumento da disponibilidade dos dados dessa base de dados.

2.5.3 Reformulação de Consultas Usando Visões Materializadas

A utilização das visões materializadas no *data warehouse* para responder às consultas complexas dos usuários de SSD garante um aumento no desempenho do processamento destas consultas, oferecendo tempos de resposta mais satisfatórios. No entanto, a utilização destas visões materializadas deve ser realizada de forma transparente ao usuário, sem que este tenha que se preocupar com quais agregações devem ser acessadas. Em outras palavras, os usuários de SSD devem especificar as suas consultas somente em termos do nível inferior do *data warehouse*, independentemente de quais visões correspondentes aos demais níveis de agregação são realmente materializadas.

Observando o mesmo problema sob outro ponto de vista, se o ambiente de *data warehousing* mantém diversas visões materializadas, o otimizador de consultas deste ambiente pode e deve utilizar estas visões materializadas durante a otimização de consultas *ad hoc*, mesmo quando estas consultas não mencionam as visões. Por exemplo, seja uma consulta da cadeia de supermercados que deseja calcular o número de produtos vendidos para cada produto. Um otimizador de consultas pode otimizar esta consulta para acessar a visão materializada que armazena o número de produtos vendidos para cada produto por filial, e evitar desta forma o acesso aos dados mais inferiores, os quais são mais volumosos.

Assim, um desafio de grande importância em ambientes de *data warehousing* consiste em reformular (reescrever) as consultas dos usuários de SSD de forma que estas utilizem uma ou mais visões materializadas. Reescrever consultas usando visões é um problema de destaque que abrange diversas aplicações de banco de dados, além de *data warehousing*. Inicialmente, o problema foi considerado com o propósito de otimização de consultas em ambientes tradicionais, nos quais visões são reutilizadas entre vários acessos compartilhados ou não, através do armazenamento de resultados intermediários para garantir um aumento na velocidade de processamento e uma diminuição do custo de execução das consultas [Rou98]. Atualmente, o conceito de reescrita de consultas usando visões tem sido empregado em outras áreas tais como integração da informação e adaptação da visão [BLR97].

O principal objetivo dos trabalhos existentes no sentido de se reformular consultas utilizando visões materializadas consiste em, dado uma consulta Q e um conjunto de visões materializadas, encontrar uma reescrita de Q, chamada de Q', de tal forma que Q' seja equivalente a ou "maior possível contida" (*maximally-contained*) em Q, mas que seja definida em termos das visões materializadas. Duas consultas Q e Q' são equivalentes quando calculam o mesmo conjunto de resposta (incluindo valores duplicados, se for o caso) para qualquer estado do banco de dados. A definição de "maior possível contida" será discutida mais adiante.

Os trabalhos analisados [GHQ95, SDJL96, DG97, CL98, KR99] diferem-se principalmente em termos (i) da linguagem utilizada para definir ambas as consultas dos usuários de SSD e as visões materializadas, incluindo o conjunto de operadores disponibilizados, se valores de instâncias duplicados podem estar presentes ou não e se as visões e/ou consultas são recursivas ou não; e (ii) da abordagem utilizada para resolver o problema, englobando tanto a premissa na qual o trabalho é baseado quanto a forma na qual o problema é atacado.

Os trabalhos de Gupta *et al*. [GHQ95], Srivastava *et al*. [SDJL96] e Chang e Lee [CL98] utilizam a linguagem de consulta SQL tanto para a definição das visões quanto para a especificação das consultas dos usuários de SSD. Ambos os trabalhos apresentam condições para a reescrita de consultas conjuntivas com agrupamento e agregação, e permitem a semântica de multiconjunto (valores de instâncias duplicados). Consultas conjuntivas são consultas definidas em termos dos operadores de seleção, de projeção e de junção. No entanto, consultas/visões em [GHQ95, SDJL96] são da forma *select-from-where-group by-having*, ao passo que consultas/visões em [CL98] são da forma *select-from-where-group by*. Em especial, as condições na cláusula *having* em [SDJL96] são utilizadas para intensificar condições na cláusula *where*, permitindo que visões materializadas que porventura poderiam ser descartadas caso esta intensificação não fosse realizada possam ser utilizadas na reescrita da consulta, sem afetar o resultado obtido. Srivastava *et al*., em especial, discutem o problema de reformular a consulta em três diferentes situações: (i) quando ambas as consultas e as visões são conjuntivas, porém somente as consultas permitem operadores de agrupamento e de agregação, (ii) quando ambas as consultas e as visões são conjuntivas com operadores de agrupamento e de agregação; e (iii) quando somente as visões permitem operadores de agrupamento e de agregação, apesar de ambas as consultas e as visões serem conjuntivas. Em contrapartida, os trabalhos de Gupta *et al*. e Chang e Lee apenas abordam a situação (ii). Em particular, Srivastava *et al*. mostram que na situação (iii) não é possível usar as visões para calcular a consulta, uma vez que a cláusula GROUP BY pode eliminar a multiplicidade das tuplas, inviabilizando a obtenção de resultados corretos frente à semântica de multiconjunto.

Por outro lado, o trabalho de Duschka e Genesereth [DG97] oferece um procedimento para construir um programa *Datalog* que é a melhor reescrita de outro programa *Datalog* dado um conjunto de visões materializadas conjuntivas e consultas recursivas genéricas. Um programa *Datalog* é um conjunto finito de regras de Horn. Dado um programa *Datalog*, pode-se definir um

grafo de dependência para este de forma que os nós correspondam aos nomes dos predicados que aparecem nas regras e que exista uma aresta do nó do predicado p_i para o nó do predicado p se p_i aparece no corpo de uma regra cujo predicado da cabeça é p. Um programa é recursivo se existe um ciclo no grafo de dependência. Por outro lado, uma visão conjuntiva consiste de uma única regra de Horn sem recursividade.

Com relação à abordagem utilizada para resolver o problema de reescrita das consultas utilizando visões materializadas, os três primeiros trabalhos destacados na comparação anterior baseiam-se na premissa de visão potencialmente útil. Dado uma consulta Q e uma visão materializada V, V é potencialmente útil para responder Q se um certo resultado intermediário que é necessário no processo do cálculo de Q também pode ser obtido de V. Intuitivamente, se V é útil para o cálculo de Q, então V deve substituir algumas relações e condições forçadas em Q, enquanto que outras relações e condições de Q devem permanecer na reescrita de Q. Esta premissa essencialmente requer um isomorfismo entre a visão e a porção da consulta: a visão deve possuir todas as tuplas e todos os atributos necessários para calcular a consulta. Em outras palavras, V tanto não deve desconsiderar quaisquer colunas (em suas projeções) quanto deve manter quaisquer tuplas necessárias ao cálculo de Q. Para visões/consultas com agregação e agrupamento, deve-se verificar se as informações de agregação em V são suficientes para calcular as agregações de Q, e se a multiplicidade das tuplas devido à cláusula GROUP BY na visão são corretas ou podem ser calculadas.

Apesar de serem baseados na mesma premissa, o trabalho de Srivastava *et al*. [SDJL96] adota uma abordagem semântica, à medida que detecta quando uma informação existente em uma visão materializada é suficiente para responder a uma consulta através de mapeamentos de coluna da visão materializada para a consulta. Em contrapartida, os trabalhos de Gupta *et al*. [GHQ95] e Chang e Lee [CL98] adotam uma abordagem sintática, realizando transformações sintáticas na árvore de consulta de uma consulta de tal forma que a definição da visão seja equivalente a uma subparte da definição da consulta. Além disto, enquanto que em [GHQ95] ambas a definição da visão materializada e a subparte da definição da consulta devem possuir o mesmo conjunto de relações, Chang e Lee identificam condições nas quais uma visão materializada pode ser utilizada para reformular uma consulta mesmo que esta visão possua relações não mencionadas na consulta.

Já o trabalho de Duschka e Genesereth [DG97] baseia-se no problema de delimitação de consulta (*query containment*). Se V representa um conjunto de visões e Q uma consulta realizada por um usuário de SSD, a consulta reescrita Q' é "maior possível contida" na consulta Q usando V se: (i) os nomes das relações que aparecem em Q' são somente os nomes das visões em V; (ii) Q' está contida na consulta Q; e (iii) não existe uma consulta P que também usa somente as visões V tal que P está contido em Q e Q' seja contido mas não equivalente a P. Baseado nesta premissa, dado um programa *Datalog* correspondente a Q, esse trabalho constrói programas *Datalog* recuperáveis (definidos somente em termos das visões V) "maiores possíveis contidos" em duas etapas. A primeira etapa constrói um programa que pode conter símbolos de função, e tem como resultado um programa lógico recuperável. Uma vez que o programa lógico resultante não é necessariamente um programa *Datalog* e que os símbolos de função são introduzidos de forma controlada, a segunda etapa transforma este programa recuperável em um programa *Datalog* que representa a reescrita da consulta (Q'). Como resultado, o programa *Datalog* recuperável resultante é "maior possível contido" no programa *Datalog* que representa Q. Concluindo, quando existe um programa *Datalog* recuperável equivalente, o conceito de "maior possível contido" implica em equivalência.

Outro trabalho que merece destaque pela forma na qual usa as visões materializadas para responder a consultas dos usuários de SSD é o trabalho de Kotidis e Roussopoulos [KR99].

Como introduzido na seção 2.5.1.3, o sistema *DynaMat* considera que os fragmentos correspondentes às consultas dos usuários de SSD estão armazenados no *view pool*, enquanto que o nível inferior está armazenado no *data warehouse* propriamente dito. A carga de trabalho considerada por esse sistema consiste de visões/consultas que permitem agrupamento e agregação e cujos predicados incluem faixas de valores. Tais visões/consultas correspondem, portanto, a definições da forma *select-from-where-group by* em SQL. No entanto, essas consultas são definidas utilizando-se uma nomenclatura abstrata, de forma que a carga de trabalho seja independente do tipo de esquema utilizado.

Diferentemente dos trabalhos discutidos anteriormente, os quais estão voltados para a reescrita das consultas em termos das visões materializadas, dada uma consulta do usuário, o sistema *DynaMat* oferece uma estrutura que percorre eficientemente o *view pool* para determinar se resultados já materializados podem ser eficientemente utilizados para responder a esta consulta. Esta busca é realizada através de um conjunto de índices conectados através do grafo de derivação, onde cada nó do grafo tem um índice dedicado que é usado para manter informação de todos os fragmentos da visão correspondente que são armazenados no *view pool*. Outra diferença fundamental entre esse trabalho e os anteriores está relacionada ao fato de que, enquanto que em [GHQ95, SDJL96, DG97, CL98] a reescrita da consulta pode ser realizada em termos de uma ou mais visões materializadas e uma visão materializada pode ser utilizada mais do que uma vez na reescrita de uma mesma consulta, em [KR99] a consulta é calculada ou a partir de um único fragmento do *view pool* ou completamente a partir das informações do *data warehouse*. Esta última diferença é determinada pela estrutura do fragmento: o sistema não permite que os fragmentos correspondentes a faixas de valores arbitrárias sejam armazenados. Ou a faixa de valores considerada corresponde ao domínio total da dimensão (como exemplo $1 \leq$ produto ≤ 1000) ou somente a um valor específico (como produto =1000).

Assim sendo, quando uma consulta do usuário de SSD é realizada durante a *fase on-line*, o sistema *DynaMat* percorre o *view pool* buscando fragmentos candidatos que respondem à consulta. Dado um fragmento F e uma consulta Q, F responde a Q se e somente se para cada faixa de valores não vazia r_i de Q, F armazena exatamente a mesma faixa de valores e, para cada faixa vazia $r_i = \varnothing$, a faixa de valores correspondente de F é tanto vazia quanto cobre o domínio total da dimensão. Baseado no conteúdo do *view pool*, existem três possibilidades: (i) quando o fragmento F corresponde exatamente à definição de Q, o sistema recupera F e retorna a resposta ao usuário de SSD; (ii) quando a possibilidade (i) não ocorre, mas existem outros fragmentos já materializados que podem ser eficientemente utilizados para responder à consulta, o sistema seleciona o melhor fragmento F para calcular Q baseado em um modelo de custo; e (iii) quando nenhum fragmento no *view pool* pode responder a Q, a consulta é manipulada pelo *data warehouse*.

2.5.3.1 Abrangência do Problema de Reformulação de Consultas Usando Visões Materializadas

Reescrever consultas usando visões representa apenas um desafio inicial ao ambiente de *data warehousing*. Em particular, a existência de diversos níveis de agregação pré-armazenados incrementa o número de representações possíveis para responder a uma mesma consulta. Como resultado, à medida que diferentes níveis de agregação mais inferiores podem ser utilizados para responder às consultas de níveis de agregação mais superiores, uma consulta pode apresentar diversas reformulações logicamente equivalentes. Fica a cargo do otimizador de consultas do ambiente gerar todas estas expressões logicamente equivalentes e otimizá-las de forma que a expressão que garanta o menor custo seja escolhida para ser executada [AHLS00].

Em especial, os algoritmos de Gupta *et al*. [GHQ95], Srivastava *et al*. [SDJL96] e Duschka e Genesereth [DG97] podem ser integrados a otimizadores de consultas convencionais para reduzir o custo da consulta. Por outro lado, o trabalho de Chang e Lee [CL98] propõe a implementação de um módulo independente para buscar por uma consulta reformulada mais eficiente sem o suporte de um otimizador de consultas, de forma que a consulta escolhida possa ser submetida a qualquer otimizador de consultas. O sistema DynaMat [KR99], por ser um sistema mais completo, engloba a decisão de quais fragmentos podem ser eficientemente usados para responder à consulta. Esta decisão é baseada em um modelo de custo que compara o custo de responder a uma consulta usando o *view pool* com o custo de executar a mesma consulta contra o *data warehouse*.

Um artigo interessante que realiza um levantamento bibliográfico extenso no contexto de otimização de consultas em sistemas relacionais é o de Chaudhuri [Cha98]. A seção 2.4.2.1 destaca diversos trabalhos existentes neste sentido, ao passo que em [ZDNS98] e em [RSSB00] são descritos algoritmos voltados à otimização de várias consultas multidimensionais relacionadas através da utilização de planos de execução que contêm tarefas compartilhadas.

2.6 Considerações Finais

Este capítulo discutiu os principais aspectos relacionados ao tema *data warehousing*, desde a sua caracterização e as suas funcionalidades, até a sua contextualização no mercado atual e sua utilização através de todos os tipos de negócio. Esta visão geral sobre o tema foi abordada em termos da arquitetura do ambiente de *data warehousing* e da descrição e da discussão de diversos algoritmos e ferramentas com as mais variadas funcionalidades voltados ao oferecimento dos requisitos impostos pelo tema. Também foi salientada a diferença de paradigma existente entre o ambiente operacional, voltado ao processamento de transações OLTP, e o ambiente informacional, voltado ao processamento de consultas analíticas OLAP. Além disto, foram caracterizados diversos problemas decorrentes da utilização do paradigma OLAP, tais como a necessidade de extensão dos componentes de um SGBD convencional, a forma especial de organização dos dados em diversos níveis de agregação e a existência de modelos de dados específicos para a modelagem multidimensional. Durante toda a discussão do tema, foram destacados diversos trabalhos relevantes e atuais que abordam os problemas em questão.

Nos últimos anos, o tema *data warehousing* tem recebido especial atenção tanto por parte da área acadêmica quanto por parte da indústria. Na área acadêmica, Widom [Wid95] representa um trabalho inicial introduzindo o tema e abordando tópicos de pesquisa na área, através do esboço de uma arquitetura inicial de um ambiente de *data warehousing* e da discussão de vários problemas técnicos a partir desta arquitetura. Os trabalhos de Chaudhuri e Dayal [CD97], Campos e Rocha Filho [CR97] e Wu e Buchmann [WB97], por sua vez, realizam uma descrição do estado da arte no tema, sugerindo extensões e enfatizando novos requerimentos. Ambos trabalhos de Chaudhuri e Dayal e Campos e Rocha Filho abordam o tema através da descrição de ferramentas (de integração e manutenção – *back end*, de análise e consulta – *front end*, para gerenciamento do repositório de metadados e do *data warehouse*) e de modelos de dados multidimensionais para OLAP. Tais trabalhos diferem-se pelo fato de que em [CD97] aspectos relacionados a extensões para o processamento eficiente de consultas são caracterizados, tais como utilização de estruturas de indexação, otimização de consultas complexas, uso de visões materializadas e extensão da linguagem SQL, ao passo que Campos e Rocha Filho evidenciam etapas do desenvolvimento de um *data warehouse*. Por outro lado, Wu e Buchmann estruturam os vários tópicos relacionados a *data warehousing* em diferentes problemas, a saber: de modelagem, de manutenção (incluindo ferramentas *back end*, gerenciamento dos metadados e do

data warehouse), arquiteturais, operacionais (incluindo extensão da linguagem SQL e índices) e de otimização (incluindo visões materializadas e otimização de consultas). Samtani *et al*. [SMKK98], por sua vez, discutem sucintamente os problemas relacionados à modelagem multidimensional dos dados e à manutenção de visões materializadas. Mais recentemente, Vassiliadis [Vas00] aborda o tema *data warehousing* realizando um paralelo entre os problemas que têm sido prioritariamente enfocados pela área acadêmica (identificação de visões a serem materializadas, integração, processamento de consultas complexas e manutenção incremental de visões materializadas) e aqueles que têm sido comumente enfrentados pela indústria (problemas de projeto, técnicos, de treinamento e sociotécnicos). Como resultado, esse último trabalho destaca a influência dos avanços da pesquisa acadêmica no segmento comercial de produtos de *data warehousing*. Um trabalho abrangendo os diversos aspectos abordados neste capítulo pode ser encontrado em [CS00a].

Com relação à indústria, existe atualmente uma grande variedade de sistemas de *data warehousing* e de ferramentas comerciais com os mais variados propósitos, como salientado ao longo das discussões realizadas neste capítulo. Em especial, diversos sistemas de *data warehousing* foram citados em momentos apropriados, tais como Hyperion Essbase [Hyp], Informix Red Brick [Inf] e Pilot Decision Support [Pil98]. Outros sistemas incluem IBM Visual Warehouse [IBM], Oracle Warehouse [Ora] e MicroStrategy 6 [Mic]. Com o objetivo de oferecer funcionalidades mais flexíveis e eficientes, muitos destes sistemas têm incorporado diversos esforços obtidos pela área acadêmica. É o caso do sistema Informix Red Brick, que passou a utilizar a tecnologia de índices do tipo mapa de *bits*.

Uma característica peculiar da área de *data warehousing* é que ela se encontra na interseção de diversos outros tópicos de pesquisa. Conseqüentemente, além de introduzir desafios específicos, muitos dos problemas enfrentados por esta área têm sido abordados pela comunidade de banco de dados em diferentes contextos, exigindo apenas extensões que se adaptem aos novos requisitos. Por exemplo, avanços em metodologias para a limpeza, a tradução e a integração dos dados na área de bancos de dados heterogêneos são úteis para o processo de carregamento dos dados dos provedores de informação no *data warehouse*. Outro exemplo refere-se ao problema de manutenção de visões, o qual estendido para manipulação eficiente de operadores de agregação e de agrupamento, possibilidade de acesso a provedores de informação remotos e suporte a visões dotadas de automanutenção, tem sido de grande valia para o processo de atualização dos dados do *data warehouse*.

Apesar de muita pesquisa na área de *data warehousing* estar sendo realizada, ainda existem muitos problemas em aberto. Além disto, os resultados práticos obtidos até então são limitados. É evidente que já existem diversos produtos comerciais disponíveis atualmente no mercado, como exemplificado anteriormente. Entretanto, as soluções utilizadas por esses sistemas são geralmente inflexíveis e assumem diversas simplificações. Assim, muitos resultados, tanto teóricos quanto práticos, ainda precisam ser obtidos.

Uma questão fundamental que merece destaque está relacionada à distribuição dos dados manipulados pelo ambiente de *data warehousing*. Do ponto de vista dos dados do *data warehouse* propriamente dito, poucos trabalhos são voltados ao oferecimento de uma metodologia que permita a distribuição de seus dados nos diversos *data marts* que compõem o ambiente, de acordo com os requisitos impostos pelas aplicações, tais como a multidimensionalidade dos dados e as necessidades dos usuários de SSD. Com relação ao repositório de metadados, a distribuição monitorada de seus metadados em diversos repositórios relacionados também torna-se imprescindível, uma vez que, assim como o *data warehouse*, o repositório de metadados armazena um volume enorme de informação e, em adição, tais informações auxiliam o ambiente durante todo o seu funcionamento. Em especial, a distribuição

dos dados tanto do *data warehouse* quanto dos metadados introduz a necessidade de processamento paralelo.

Segundo Garcia-Molina *et al*. [GMLWZ98], o processamento paralelo e distribuído também representa novas oportunidades para o componente de integração e manutenção, o qual pode ser considerado como uma entidade separada de ambos os provedores de informação e do *data warehouse*. Assim, o componente de integração e manutenção pode ser composto por vários componentes distribuídos individuais, cada um possuindo tarefas específicas e representando um processo separado. Como resultado, além de serem distribuídos, tais processos podem ser executados em paralelo. Esta abordagem é utilizada no sistema WHIPS, o qual implementa os módulos do sistema como objetos CORBA que podem ser executados em qualquer máquina.

Não somente a distribuição dos metadados em diversos repositórios é importante, mas também o compartilhamento destes metadados pelas diversas ferramentas utilizadas pelo sistema de *data warehousing* durante todas as suas fases de projeto, de criação, de manutenção e de acesso. Tal compartilhamento impõe tanto a necessidade de um modelo comum quanto uma maior integração do repositório de metadados com os demais processos do sistema, além da necessidade de módulos gerenciadores de metadados mais robustos. Em especial, diversas funcionalidades a serem oferecidas por estes módulos foram destacadas na seção 2.2.5. É importante destacar que, embora já existam atualmente algumas tentativas no sentido de se obter um padrão para a representação e a troca de metadados em ambientes de *data warehousing*, tais tentativas têm enfrentado muitas resistências.

A existência de diversas ferramentas *back end* e *front end* que auxiliam processos específicos de um sistema de *data warehousing* também introduz desafios. Por um lado, a utilização de diferentes ferramentas voltadas ao suporte destes processos introduz a necessidade de interoperabilidade entre estas ferramentas. Esta interoperabilidade é de suma importância para a flexibilidade das empresas para introduzir novos serviços de aplicação, reduzir custos e suprir necessidades. Por outro lado, existem várias ferramentas distintas que apresentam a mesma funcionalidade. Assim, torna-se interessante oferecer um mecanismo automatizado que direcione a escolha da melhor ferramenta, levando-se em consideração fatores tais como os requisitos da aplicação, o custo e o desempenho da ferramenta, e a sua facilidade de utilização, além da sua interoperabilidade com outras ferramentas.

Um tópico que sempre merecerá grande evidência consiste no aprimoramento das funcionalidades oferecidas pela máquina relacional – extensão da linguagem SQL, estruturas de indexação, processamento de consultas complexas e processamento paralelo – visando diminuir cada vez mais as funcionalidades da máquina ROLAP. Por exemplo, a necessidade de índices eficientes para o aumento do desempenho no processamento de consultas é contínua. Apesar dos índices do tipo mapa de *bits* serem eficazes, Sarawagi [Sar97] mostra que alguns índices inicialmente projetados para bancos de dados espaciais, tal como árvore-R, apresentam melhores resultados que índices do tipo mapa de *bits* em certas situações. Assim, sempre existirão novas oportunidades nesta área. Outras extensões, referentes ao processamento de consultas complexas, são o suporte a consultas de agregação aproximadas, assim como em bancos de dados estatísticos, e a otimização simultânea de várias expressões logicamente equivalentes correspondentes à reformulação de uma consulta do usuário de SSD visando diminuir os custos envolvidos neste processo. Em especial, somente mais recentemente a obtenção eficiente de respostas aproximadas com o objetivo de garantir um melhor tempo de resposta às consultas de suporte à decisão sobre *data warehouses* volumosos tem sido pesquisada pela área acadêmica [CDN01, Gib01, LM01]. Já para sistemas MOLAP, o trabalho de Dinter *et al*. [DSHB98] detalha diversos pontos falhos ou ineficientes relacionados a estes sistemas, tais como distribuição, paralelismo, conceitos transacionais e suporte a versões. A utilização da tecnologia objeto-

relacional para o armazenamento dos dados multidimensionais, por sua vez, requer um estudo mais aprofundado e exaustivo.

Com relação ao tópico de visões materializadas, muito trabalho ainda precisa ser realizado no sentido de se determinar dinamicamente qual o melhor conjunto de visões a serem materializadas de forma a oferecer um processamento de consultas eficiente sem confrontar com os requisitos de espaço de armazenamento e de tempo de manutenção. Embora o trabalho de Kotidis e Roussopoulos [KR99] represente um primeiro avanço neste sentido, a estrutura dos fragmentos armazenados pelo sistema introduz algumas limitações, tal como o fato de que a consulta deve ser calculada a partir de um único fragmento do *view pool* ou completamente a partir das informações do *data warehouse*. Um possível caminho a ser seguido consiste na utilização de blocos como unidades de materialização [SDN98]. Outra extensão que deve ser considerada refere-se ao suporte a tipos mais complexos de consultas dos usuários de SSD. Com respeito à atualização dos dados do *data warehouse*, existe a necessidade de um estudo sistemático que permita determinar qual a periodicidade de manutenção adequada, em termos do nível de consistência desejado (convergência, consistência fraca, consistência forte e completude), da janela de atualização, da quantidade de agregações e das necessidades das aplicações.

Outro desafio associado ao tema *data warehousing* consiste na necessidade de exploração e análise dos dados armazenados no *data warehouse* via Web, sem afetar a segurança, a privacidade e a autonomia dos provedores de informação individuais e oferecendo aos usuários de SSD acesso uniforme e consistente às informações desejadas. Um ambiente de *data warehousing* para Web deve oferecer: (i) uma interface que facilite que usuários de SSD realizem consultas de maneira amigável; (ii) suporte a um grande número de usuários; (iii) mecanismos de comunicação eficientes e seguros que permitam a troca de informações entre os provedores localizados na Web e o *data warehouse*, e entre o *data warehouse* e os clientes; (iv) tolerância a falhas; (v) disponibilidade dos dados 24 horas por dia, 7 dias por semana; e (vi) reconfiguração dinâmica. Outros requisitos de um ambiente de *data warehousing* na Web também foram identificados na seção 1.4. Enquanto que esse desafio utiliza a Web como uma tecnologia subjacente, Harinarayan [Har97] discute a utilização da tecnologia OLAP como uma tecnologia subjacente para muitos *sites* atualmente disponíveis na Web, através de uma mudança no paradigma de domínio de analistas para o domínio dos consumidores. Esse trabalho argumenta que *sites* de busca, tais como Alta Vista [Alt] e Lycos [Lyc] são basicamente "*data warehouses*", mas apresentam uma infra-estrutura para suporte à decisão que é extremamente rudimentar e limitada. Através da estruturação destes "*data warehouses*" aplicando-se os avanços obtidos até então nesta área, consultas complexas tais como as comumente realizadas em sistemas de suporte à decisão podem ser eficientemente respondidas.

Alguns temas abordados ao longo deste capítulo que somente recentemente têm recebido mais atenção da área acadêmica dentro do contexto de *data warehousing* são os problemas de linhagem dos dados (seção 2.2.3.2), de recuperação de falhas (seção 2.2.2.1.6) e de expiração dos dados do *data warehouse* (seção 2.2.2.3). O problema de linhagem dos dados pode ser adaptado ao processo de limpeza dos dados durante a fase de carregamento, facilitando a identificação e a correção de dados suspeitos ou incorretos. Já a recuperação de falhas em ambientes de *data warehousing* deve ser realizada em duas situações distintas: tanto durante as fases de carregamento dos dados e de atualização dos dados do *data warehouse*, quanto durante o processamento de consultas dos usuários de SSD. Em ambos os casos, as atividades realizadas durante estas duas situações são de longa duração, manipulam um volume razoável de dados e utilizam muitos recursos do sistema. Conseqüentemente, soluções eficazes baseadas em baixa sobrecarga do sistema são essenciais para integrar a recuperação de falhas às demais atividades

sendo realizadas. Atualmente, a determinação de quais dados devem ser expirados é realizada pelo administrador do ambiente de *data warehousing*, de acordo com a regulamentação da organização e das necessidades das aplicações. Assim, a especificação de regras que auxiliem esta tomada de decisão representa um tópico de interesse para este problema. De maneira geral, trabalhos apresentando novas metodologias para resolver os problemas de linhagem dos dados, de recuperação de falhas e de expiração dos dados do *data warehouse*, com algoritmos mais eficientes, são indispensáveis para um avanço nestes temas.

Em contrapartida, apesar de já existirem vários trabalhos voltados ao oferecimento de um modelo de dados conceitual multidimensional (seção 2.4.3), ainda há espaço para novas pesquisas nesta área. Isto é principalmente decorrente do fato que ainda não existe um modelo de dados conceitual padrão amplamente aceito e utilizado para a representação de aplicações de *data warehousing*. Em especial, é importante que o modelo de dados conceitual multidimensional seja semanticamente mais rico do que o modelo lógico utilizado atualmente para o armazenamento do cubo de dados. Modelos de dados semanticamente mais ricos oferecem uma variedade maior de abstrações, característica imprescindível na modelagem de aplicações de *data warehousing*. Além disto, o modelo de dados conceitual multidimensional deve oferecer conceitos próximos à forma na qual os usuários enxergam os dados, permitindo que tais usuários possuam um melhor entendimento da aplicação sendo modelada.

Finalmente, como discutido no início deste capítulo, o volume de dados manipulado por aplicações de *data warehousing* tem crescido exponencialmente, e em pouco tempo, propiciará o surgimento de bancos de dados contendo *petabytes* de dados multidimensionais. Apesar do crescimento na capacidade de armazenamento dos discos nos últimos anos, estes dispositivos não conseguirão armazenar sozinhos grandes volumes de dados em um futuro próximo, seja em discos magnéticos ou magnético-óticos. Este fato introduz o problema de gerenciar um novo nível de hierarquia de armazenamento, chamado armazenamento terciário. Este terceiro nível utiliza dispositivos (tais como *compact-disk juke boxes* ou *tape silos*) que são ordens de magnitude mais lentos do que os dispositivos de armazenamento secundário (discos), aliados a uma capacidade de armazenamento muito superior. Assim, um tópico muito interessante consiste em atacar o problema de armazenamento e de recuperação de grandes volumes de dados multidimensionais em dispositivos de armazenamento terciário. Serão necessários, para tanto, novos mecanismos de escalonamento e de *bufferização* que permitam mascarar a diferença de velocidade existente entre os dispositivos de armazenamento secundário e terciário. Também será imprescindível a adaptação de algumas técnicas de implementação tais como armazenamento dos dados, indexação e expiração dos dados. Em especial, a identificação de quais visões correspondentes aos níveis de agregação materializar deve levar em consideração este novo nível de armazenamento. Por exemplo, é mais rápido não materializar uma visão e calculá-la em tempo de execução, ou armazená-la em meio terciário?.

Referências Bibliográficas

[ACM+99] Abiteboul, S., Cluet, S., Milo, T., Mogilevsky, P., Siméon, J., Zohar, S. Tools for Data Translation and Integration. *IEEE Data Engineering Bulletin*, 22(1):3-8, March 1999.

[ADP] ADP Brokerage Information Service Group. Internet. Available at URL http://www.adp.com.

[Ago] Agouron Pharmaceuticals. Internet. Available at URL http://www.agouron.com.

[AGS97] Agrawal, R., Gupta, A., Sarawagi, S. Modeling Multidimensional Databases. In *Proceedings of the 13th International Conference on Data Engineering*, pages 232-243, Birmingham, UK, April 1997.

[Agu95] Aguiar, C. D. Integração de Sistemas de Banco de Dados Heterogêneos em Aplicações de Planejamento Urbano. Master's thesis, Universidade Estadual de Campinas, Campinas, SP, BR, 1995. 169 pp.

[AHLS00] Albrecht, J., Hümmer, W., Lehner, W., Schlesinger, L. Query Optimization by Using Derivability in a Data Warehouse Environment. In *Proceedings of the 3rd International Conference on Data Warehousing and OLAP*, pages 49-56, Washington, DC, USA, November 2000.

[Alt] Alta Vista. Internet. Available at URL http://www.alta-vista.com.

[Alu96] Alur, N. The Enterprise Data Warehouse and Data Mart Debate. *InfoDB Magazine*, 10(2):13-20, April 1996.

[Ame] American Airlines. Internet. Available at URL http://www.aa.com.

[AMS+96] Agrawal, R., Mehta, M., Shafer, J., Srikant, R., Arning, A., Bollinger, T. The Quest Data Mining System. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 244-249, Oregon, August 1996.

[AOSS00] Abelló, A., Oliva, M., Samos, J., Saltor, F. Information System Architecture for Secure Data Warehousing. In *Proceedings of the 3rd International Workshop on Engineering Federated Information Systems*, pages 33-40, Ireland, June 2000.

[Ape88] Apers, P.M.G. Data Allocation in Distributed Database Systems. *ACM Transactions on Database Systems*, 13(3):263-304, September 1988.

[APT96] APT Data Group. What is Metadata. *Data Warehousing Tools Bulletin*, January 1996. Available at URL http://pwp.starnetinc.com/larryg/articles.html.

[ARM] ARM Financial Group, Inc. Internet. Available at URL http://www.armfinancial.com.

[ASS00] Abelló, A., Samos, J., Saltor, F. Benefits of an Object-Oriented Multidimensional Data Model. In *Proceedings of the International Symposium on Objects and Databases*, pages 141-152, Sophia Antipolis and Cannes, France, June 2000, volume 1.944 of *Lecture Notes in Computer Science*, Springer, 2000.

[ASS01a] Abelló, A., Samos, J., Saltor, F. Understanding Analysis Dimensions in a Multidimensional Object-Oriented Model. In *Proceedings of the 3rd International Workshop on Design and Management of Data Warehouses*, pages 4.1-4.9, Interlaken, Switzerland, June 2001.

[ASS01b] Abelló, A., Samos, J., Saltor, F. Understanding Facts in a Multidimensional Object-Oriented Model. In *Proceedings of the 4th International Workshop on Data Warehousing and OLAP*, pages 32-39, Atlanta, USA, November 2001.

[ATT] AT&T Corp. Internet. Available at URL http://www.att.com.

[Ban] Bank of America. Internet. Available at URL http://www.bofa.com.

[BHP92] Bright, M. W., Hurson, A. R., Pakzad, S. H. A Taxonomy and Current Issues in Multidatabase Systems. *Computer*, 25(3):50-59, March 1992.

[BL93] Berners-Lee, T. A General Overview of the WWW. An overview talk given in the Geneva University Seminar, Geneva, Switzerland, June 1993. Available at URL http://www.w3.org/talks.

[BLCL+94] Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H.F., Secret, A. The World-Wide Web. *Communications of the ACM*, 37(8):76-82, August 1994.

[BLR97] Beeri, C., Levy, A.Y., Rousset, M-C. Rewriting Queries Using Views in Description Logics. In *Proceedings of the 16th Symposium on Principles of Database Systems*, pages 99-108, Tucson, Arizona, May 1997.

[BPT97] Baralis, E., Paraboschi, S., Teniente, E. Materialized View Selection in a Multidimensional Database. In *Proceedings of the 23rd Very Large Data Bases Conference*, pages 25-29, Athens, Greece, August 1997.

[BS96] Byard, J., Schneider, D. The Ins & Outs (and Everything in Between) of Data Warehousing. Transparencies of the tutorial presented at International Conference on Management of Data, Montreal, Canada, June 1996. 44 pp. Available at URL http://www.informix.com/informix/solutions/dw/redbrick/wpapers.

[BS97] Berson, A., Smith, S. *Data Warehousing, Data Mining, and OLAP*. McGraw-Hill, USA, 1997. 630 pp. ISBN 0-07-006272-2.

[Bur97] Buretta, M. *Data Replication: Tools and Techniques for Managing Distributed Information*. John Wiley & Sons, Inc., USA, 1997. 360pp. ISBN 0-471-15754-6.

[Cam99] Campos, M.L. Metadados em Ambiente de DW. Transparências apresentadas na disciplina de mestrado em Informática da UFRJ, Brasil, 1999. 42 transparências. Available at URL http://genesis.nce.ufrj.br/dataware.

[Cam00] Campos, M.L. MOF e Metamodelagem. Transparências apresentadas na disciplina de mestrado em Informática da UFRJ, Brasil, 2000. 20 transparências. Available at URL http://genesis.nce.ufrj.br/dataware.

[Cas+95] Castro, M.A.S., *et al*. *Tutorial HTML*. Instituto de Ciências Matemáticas e de Computação (ICMC), Universidade de São Paulo (USP), São Carlos, São Paulo, Brasil, 1995. Available at URL http://www.icmc.sc.usp.br/manuals/HTML.

[CB97] Cheswick, W.R., Bellovin, S.M. *Firewalls and Internet Security*. Addison-Wesley Publishing Company, USA, 1997. 306pp. ISBN 0-20-163357-4.

[CC98] Ciferri, R.R., Ciferri, C.D.A. Firewalls. *Revista Tecnológica*, 1(7):49-58, October 1998.

[CCS93] Codd, E.F., Codd, S.B., Salley, C.T. Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate. White paper, 1993. Available at URL http://www.arborsoft.com/papers.

[CCS99] Ciferri, C.D.A., Ciferri, R.R., Souza, F.F. Uma Arquitetura de Replicação/Fragmentação para Acesso Distribuído a *Data Warehouse* via Web. In *Memorias da XXV Conferencia Latino-americana de Informática*, volume 1, pages 139-150, Asunción, Paraguay, September 1999.

[CD97] Chaudhuri, S., Dayal, U. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65-74, March 1997.

[CDN01] Chaudhuri, S., Das, G., Narasayya, V. A Robust, Optimization-Based Approach for Approximate Answering of Aggregate Queries. In *Proceedings of the 2001 International Conference on Management of Data*, Santa Barbara, California, USA, May 2001. *SIGMOD Record*, 30(2):295-306, June 2001.

[CDSS98] Cluet, S., Delobel, C., Siméon, J., Smaga, K. Your Mediators Need Data Conversion! In *Proceedings of the International Conference on Management of Data*, pages 177-188, Seattle, Washington, USA, June 1998.

[Cha98] Chaudhuri, S. An Overview of Query Optimization in Relational Systems. In *Proceedings of the 17th Symposium on Principles of Database Systems*, pages 34-43, Seattle, Washington, June 1998.

[CI98] Chan, C.-Y., Ioannidis, Y.E. Bitmap Index Design and Evaluation. In *Proceedings of the International Conference on Management of Data*, pages 355-366, Seattle, Washington, USA, June 1998.

[CL98] Chang, J-y., Lee, S.-g. Query Reformulation Using Materialized Views in Data Warehouse Environment. In *Proceedings of the 1st International Workshop on Data Warehousing and OLAP*, pages 54-59, Washington, DC, USA, November 1998.

[CLS00] Chan, M., Leong, H.V., Si, A. Incremental Update to Aggregated Information for Data Warehouses over Internet. In *Proceedings of the 3rd International Conference on Data Warehousing and OLAP*, pages 57-64, Washington, DC, USA, November 2000.

[CMVN94] Chakravarthy, S., Muthuraj, J., Varadarajan, R., Navathe, S.B. An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis. *Distributed and Parallel Databases*, 2(2):183-207, April 1994.

[CNP82] Ceri, S., Negri, M. Pelagatti, G. Horizontal Data Partitioning in Database Design. In *Proceedings of the 1982 International Conference on Management of Data*, pages 128-136, Orlando, Florida, USA, June 1982.

[CNW83] Ceri, S., Navathe, S.B., Wiederhold, G. Distribution Design of Logical Database Schemas. *IEEE Transactions on Software Engineering*, 9(4):487-504, July 1983.

[Col96] Colliat, G. OLAP, Relational, and Multidimensional Database Systems. *SIGMOD Record*, 25(3):64-69, September 1996.

[CP82a] Ceri, S., Pelagatti, G. A Solution Method for the Non-Additive Resource Allocation Problem in Distributed System Design. *Information Processing Letters*, 15(4):174-178, October 1982.

[CP82b] Ceri, S., Pelagatti, G. Allocation of Operations in Distributed Database Access. *IEEE Transactions on Computers*, 31(2):119-129, February 1982.

[CR97] Campos, M.L., Rocha Filho, A.V. Data Warehouse. In *Anais da XVI Jornada de Atualização em Informática – XVII Congresso da Sociedade Brasileira de Computação*, pages 221-261, Brasília, DF, Brasil, August 1997.

[CRGMW96] Chawathe, S.S., Rajaraman, A., Garcia-Molina, H., Widom, J. Change Detection in Hierarchically Structured Information. In Jagadish, H.V., Mumick, I.S., editors, *Proceedings of the 1996 International Conference on Management of Data*, volume 25 of *SIGMOD Record*, pages 493-504. ACM Press, June 1996.

[CS00a] Ciferri, C.D.A., Souza, F.F. *Data Warehousing*: Estado da Arte e Direções Futuras. In *Memorias da XXVI Conferência Latino-americana de Informática*, pages CD-ROM, Mexico City, Mexico, September 2000.

[CS00b] Ciferri, C.D.A., Souza, F.F. Distribuindo os Dados do *Data Warehouse*. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, pages 346-360, João Pessoa, Paraíba, Brasil, October 2000.

[CS01a] Ciferri, C.D.A., Souza, F.F. Distribuição dos Dados em Ambientes de Data Warehousing. In *Memorias da XXVII Conferencia Latinoamericana de Informática*, pages CD-ROM, Mérida, Venezuela, September 2001. 12 pp.

[CS01b] Ciferri, C.D.A., Souza, F.F. Materialized Views in Data Warehousing Environments. In *Proceedings of the XXI International Conference of the Chilean Computer Science Society*, pages 3-12, Punta Arenas, Chile, November 2001. IEEE Computer Society Press.

[CS02] Ciferri, C.D.A., Souza, F.F. Focusing on Data Distribution in the WebD^{2}W System. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, pages 265-274, Aix-en-Provence, France, September 2002, volume 2454 of *Lecture Notes in Computer Science*, Springer, 2002.

[CT98] Cabbibo, L., Torlone, R. A Logical Approach to Multidimensional Databases. In Schek, H.-J., Saltor, F., Ramos, I., Alonso, G., editors, *Advances in Database Technology –Proceedings of the 6th International Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, pages 183-197. Springer, 1998.

[CT00] Cabibbo, L., Torlone, R. The Design and Development of a Logical System for OLAP. In *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery*, pages 1-10, London, UK, September 2000, volume 1.874 of *Lecture Notes in Computer Science*, Springer, 2000.

[CW00] Cui, Y., Widom, J. Lineage Tracing in a Data Warehousing System. In *Proceedings of the 16th International Conference on Data Engineering*, San Diego, California, February 2000. Demonstration Description.

[CWW97] Cui, Y., Widom, J., Wiener, J.L. Tracing the Lineage of View Data in a Warehousing Environment. Technical Report, Stanford University Database Group, November 1997. 50pp. Available at URL http://www-db.stanford.edu/pub/papers/lineage-full.ps.

[DDJ+98] Do, L., Drew, P., Jin, W., Jumani, V., Rossum, D.V. Issues in Developing Very Large Data Warehouses. In *Proceedings of the 24th Very Large Data Bases Conference*, pages 633-636, New York, USA, August 1998.

[Deu+92] Deux, O., *et al*. The Story of O_2. In: Bancilhon, F., Delobel, C., Kanellakis, P., editors, *Building an Object-Oriented Database System: The Story of O_2*, pages 21-57. Morgan Kaufmann Publisher, Inc., 1992. ISBN 558-60169-4.

[DG97] Duschka, O.M., Genesereth, M., R. Answering Recursive Queries Using Views. In *Proceedings of the 16th Symposium on Principles of Database Systems*, pages 109-116, Tucson, Arizona, May 1997.

[Die97] Diestel, R. *Graph Theory*. Springer-Verlag New York, Inc., USA, 1997. 286 pp. ISBN 0-387-98211-6.

[DMT98] Datta, A., Moon, B., Thomas, H. A Case for Parallelism in Data Warehousing and OLAP. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, pages 226-231, Vienna, Austria, August 1998.

[DSHB98] Dinter, B., Sapia, C., Höfling, G., Blaschka, M. The OLAP Market: State of the Art and Research Issues. In *Proceedings of the 1st International Workshop on Data Warehousing and OLAP*, pages 22-27, Washington, DC, USA, November 1998.

[EN00] Elmasri, R., Navathe, S.B. *Fundamentals of Database Systems.* Addison-Wesley Publishing Company, Menlo Park, CA, USA, 3rd edition, 2000. 955pp. ISBN 0-201-54263-3.

[EP90] Elmagarmid, A. K., Pu C. Guest Editors' Introduction to the Special Issue on Heterogeneous Databases. *ACM Computing Surveys*, 22(3):175-178, September 1990.

[EWH85] Elmasri, R., Weeldreyer, J., Hevner, A. The Category Concept: An Extension to the Entity-Relationship Model. In: Schneider, H. J., editor, *Data and Knowledge Engineering*, pages 75-116. North Holland, 1985.

[Fay98] Fayyad, U. Mining Databases: Towards Algorithms for Knowledge Discovery. *IEEE Data Engineering Bulletin*, 21(1):39-48, March 1998.

[FF96] Frisch, E., Frisch, A. *Essential System Administration*. O'Reilly & Associates, 1996. 788p. ISBN 1-565-92127-5

[For] Fordham University. Internet. Available at URL http://www.fordham.edu.

[Fre] Fremont Compensation Insurance Company. Internet. Available at URL http://www.fremontcomp.com.

[Fre98] Freitas, A.A. Data Mining. Transparencies of the tutorial presented at XII Simpósio Brasileiro de Engenharia de Software, Maringá, Paraná, Brasil, October 1998. 52 pp.

[FSGM+98] Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R., Ullman, J.D. Computing Iceberg Queries Efficiently. In *Proceedings of the 24th Very Large Data Bases Conference*, pages 299-310, N.Y., USA, August 1998.

[FZ92] Folk, M.J., Zoellick, B. *File Structures*. Addison-Wesley Publishing Company, Inc, USA, 2nd edition, 1992. 590pp. ISBN 0-201-55713-4.

[GBLP96] Gray, J., Bosworth, A., Layman, A., Pirahesh, H. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. In *Proceedings of the 12th International Conference on Data Engineering*, pages 152-159, New Orleans, Louisiana, USA, February 1996.

[Gen] General Electric. Internet. Available at URL http://www.ge.com.

[GHQ95] Gupta, H., Harinarayan, V., Quass, D. Aggregate-Query Processing in Data Warehousing Environments. In *Proceedings of the 21st Very Large Data Bases Conference*, pages 358-369, Zurich, Swizerland, September 1995.

[GHRU97] Gupta, H., Harinarayan, V., Rajaraman, A., Ullman, J.D. Index Selection for OLAP. In *Proceedings of the 13th International Conference on Data Engineering*, pp. 208-219, Birmingham, UK, April 1997.

[Gib01] Gibbons, P.B. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 541-550, Roma, Italy, September 2001.

[GL95] Griffin, T., Libkin, L. Incremental Maintenance of Views with Duplicates. In Carey, M.J., Schneider, D.A., editors, *Proceedings of the 1995 International Conference on Management of Data*, volume 24 of *SIGMOD Record*, pages 328-339. ACM Press, June 1995.

[GM95] Gupta, A., Mumick, I.S. Maintenance of Materialized Views: Problems, Techniques, and Applications. *IEEE Data Engineering Bulletin*, 18(2):3-18, June 1995.

[GMLWZ98] Garcia-Molina, H., Labio, W.J., Wiener, J.L., Zhuge, Y. Distributed and Parallel Computing Issues in Data Warehousing. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Puerto Vallarta, Mexico, July 1998.

[GMLY98] Garcia-Molina, H., Labio, W.J., Yang, J. Expiring Data in a Warehouse. In *Proceedings of the 24th Very Large Data Bases Conference*, pages 500-511, New York, USA, August 1998.

[GMR98] Golfarelli, M., Maio, D., Rizzi, S. Conceptual Design of Data Warehouses from E/R Schemes. In *Proceedings of the 31st Hawaii International Conference on System Sciences*, pages 334-343, Kona, Hawaii, January 1998.

[GMR00] Golfarelli, M., Maio, D., Rizzi, S. Applying Vertical Fragmentation Techniques in Logical Design of Multidimensional Databases. To appear in *Proceedings of the 2nd International Conference on Data Warehousing and Knowledge Discovery*, London-Greenwich, UK, September 2000.

[Gre] Greenfield, L. The Data Warehousing Information Center. Internet. Available at URL http://www.dwinfocenter.org.

[Gri01] Griffin, J. The Pitfalls of 'Virtually' Building a Data Warehouse. White paper, 2001. Available at URL http://www.datawarehouse.com/iknowledge/articles/article.cfm?ContentID=1522.

[Har97] Harinarayan, V. Issues in Interactive Aggregation. *IEEE Data Engineering Bulletin*, 20(1):12-18, March 1997.

[HGMW+95] Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., Zhuge, Y. The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin*, 18(2):41-48, June 1995.

[HM96] Hadjiefthymiades, S., Martakos, D. A Generic Framework for the Deployment of Structured Databases on the WWW. In *Proceedings of the 5th* International World Wide Web *Conference*, Paris, France, May 1996. Available at URL http://www5conf.inria.fr/fich_html/papers/P22/Overview.html.

[HMV99] Hurtado, C.A., Mendelzon, A.O., Vaisman, A.A. Maintaining Data Cubes under Dimension Updates. In *Proceedings of the 15th International Conference on Data Engineering*, pages 346-355, Sydney, Australia, March 1999.

[Hos] Hospital das Clínicas da Faculdade de Medicina da Universidade de São Paulo. Internet. Available at URL http://www.hcnet.usp.br/pagina1.htm.

[HPDW01] Han, J., Pei, J., Dong, G., Wang, K. Efficient Computation of Iceberg Cubes with Complex Measures. In *Proceedings of the 2001 International Conference on Management of Data*, Santa Barbara, California, USA, May 2001. *SIGMOD Record*, 30(2):1-12, June 2001.

[HRU96] Harinarayan, V., Rajaraman, A., Ullman, J.D. Implementing Data Cubes Efficiently. In Jagadish, H.V., Mumick, I.S., editors, *Proceedings of the 1996 International Conference on Management of Data*, volume 25 of *SIGMOD Record*, pages 205-216. ACM Press, June 1996.

[Hsi92] Hsiao, D. K. Federated Databases and Systems: Part I – A Tutorial on Their Data Sharing. *The VLDB Journal*, 1(1):127-179, July 1992.

[Huy97] Huyn, N. Multiple-View Self-Maintenance in Data Warehousing Environments. In *Proceedings of the 23rd Very Large Data Bases Conference*, pages 16-25, Athens, Greece, August 1997.

[Hyp] Hyperion. Essbase. Internet. Available at URL http://www.hyperion.com.

[IBM] IBM. Visual Warehouse. Internet. Available at URL http://www-4.ibm.com/software/data/vw.

[IH94] Inmon, W.H., Hackathorn, R.D. *Using the Data Warehouse*. John Wiley & Sons, Inc., USA, 1994. 285 pp. ISBN 0471-05966-8.

[Inf] Informix Red Brick. Decision-Makers, Business Data and RISQL. White paper. Available at URL http://www.informix.com/informix/solutions/dw/redbrick/wpapers.risql.html.

[Inm96] Inmon, W.H. *Building the Data Warehouse*. John Wiley & Sons, Inc, USA, 2nd edition, 1996. 401 pp. ISBN 0-471-15337-0.

[KCT99] Klein, L.Z., Campos, M.L.M., Tanaka, A.K. A Tecnologia Objeto-Relacional em Ambientes de Data Warehouse: Uso de Séries Temporais como Tipo de Dado Não Convencional. In *Anais do XIV Simpósio Brasileiro de Banco de Dados*, pages 365-378, Florianópolis, Santa Catarina, Brasil, October 1999.

[Ken96] The Ken Orr Institute. Data Warehousing Technology. White paper, 1996. Available at URL http://www.kenorrinst.com/dwpaper.html.

[Ken00] The Ken Orr Institute. Data Warehousing Technology. White paper, revised edition, 2000. Available at URL http://www.kenorrinst.com/dwpaper.html.

[Kim96] Kimball, R. *The Data Warehouse Toolkit*. John Wiley & Sons, Inc, USA, 1996. 388 pp. ISBN 0-471-15337-0.

[Kim98a] Kimball, R. Meta Meta Data Data. *DBMS online Magazine*, March 1998. Available at URL http://www.dbmsmag.com/9803d05.html.

[Kim98b] Kimball, R. Brave New Requirements for Data Warehousing. *Intelligent Enterprise*, October 1998. Available at URL http://www.intelligententerprise.com/dwframe.html.

[KKM93] Keim, D. A., Kriegel, H. P., Miethsam, A. Object-Oriented Querying of Existing Relational Databases. In Marík, V., Lazanský, J., Wagner, R., editors, *Proceedings of the 4th International Database and Expert Systems Applications Conference*, volume 720 of *Lecture Notes in Computer Science*, pages 325-336. Springer, 1993.

[KM00] Kimball, R., Merz, R. The Data Webhouse Toolkit: Building the Web-Enabled Data Warehouse. John Wiley & Sons, Inc., USA, 2000. 401 pp. ISBN 0-471-37680-9.

[KR98] Kotidis, Y., Roussopoulos, N. An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. In Proceedings *of the 1998 International Conference on Management of Data*, pages 249-258, Seattle, Washington, June 1998.

[KR99] Kotidis, Y., Roussopoulos, N. DynaMat: A Dynamic View Management System for Data Warehouses. In *Proceedings of the International Conference on Management of Data*, pages 371-382, Philadephia, EUA, June 1999.

[KW96] Kalakota, R., Whinston, A. *Electronic Commerce: A Manager's Guide*. Addison-Wesley Publishing Company, USA, 1996. 430pp. ISBN 0-2018-8067-9.

[LGM96] Labio, W.J., Garcia-Molina, H. Efficient Snapshot Differential Algorithms for Data Warehousing. In *Proceedings of the 22nd Very Large Data Bases Conference*, pages 63-74, Mumbai (Bombai), India, September 1996.

[Lim97] Lima, I. N. Ambiente Web Banco de Dados: Funcionalidades e Arquiteturas de Integração. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, BR, 1997. 158 pp.

[LM01] Lazaridis, I., Mehrotra, S. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *Proceedings of the 2001 International Conference on Management of Data*, Santa Barbara, California, USA, May 2001. *SIGMOD Record*, 30(2):401-412, June 2001.

[LWGMG00] Labio, W.J., Wiener, J.L., Garcia-Molina, H., Gorelik, V. Efficient Resumption of Interrupted Warehouse Loads. In *Proceedings of the 2000 International Conference on Management of Data*, pages 46-57, Dallas, Texas, May 2000.

[Lyc] Lycos. Internet. Available at URL http://www.lycos.com.

[LYGM99] Labio, W.J., Yerneni, R., Garcia-Molina, H. Shrinking the Warehouse Update Window. In *Proceedings of the 1999 International Conference on Management of Data*, pages 383-394, Philadelphia, Pennsylvania, USA, June 1999.

[LZW+97] Labio, W.J., Zhuge, Y., Wiener, J.L., Gupta, H., Garcia-Molina, H., Widom, J. The WHIPS Prototype for Data Warehouse Creation and Maintenance. In Peckham, J., editor, *Proceedings of the 1997 International Conference on Management of Data*, volume 26 of *SIGMOD Record*, pages 557-559. ACM Press, June 1997.

[Mal+96] Malaika, S., *et al*. DataBase Management Systems and the Internet. In *Proceedings of the 22nd Very Large Data Bases Conference*, pages 600-601, Mumbai (Bombai), India, September 1996.

[Mat99] Mattison, R. Web Warehousing and Knowledge Management. McGrall-Hill Companies, Inc., USA, 1999. 576 pp. ISBN 0-07-041103-4.

[MC00] Monteiro Neto, R. R., Campos, M.L.M. Modelo de Custos para Seleção Dinâmica de Agregados a Materializar em Data Warehouses. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, pages 331-345, João Pessoa, Paraíba, Brasil, October 2000.

[MCVN93] Muthuraj, J., Chakravarthy, S., Varadarajan, R., Navathe, S.B. A Formal Approach to the Vertical Partitioning Problem in Distributed Database Design. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, Issues, Architectures, and Algorithms*, pages 26-34, San Diego, CA, USA, January 1993.

[Mel97] Melo, R.N. Data Warehouse Technology. Transparencies of the tutorial presented at XII Simpósio Brasileiro de Banco de Dados, Brasil, October 1997. 44 pp.

[Met99a] META Group. 1999 Data Warehouse Marketing Trends/Opportunities. Internet, 07/01/99. Available at URL http://www.metagroup.com.

[Met99b] Meta Data Coalition and Object Management Group Form Cooperative Relationship to Build Consensus on Metadata Standards. Internet, April 20, 1999. Available at URL http://www.MDCInfo.com/press/pr19990420.html.

[MF98] Mesquita, E.J.S., Finger, M. Projeto de Dados em Bancos de Dados Distribuídos. In *Anais do XIV Simpósio Brasileiro de Banco de Dados*, pages 87-101, Maringá, Paraná, Brasil, October 1998

[Mic] MicroStrategy. MicroStrategy 6. Internet. Available at URL http://www.microstrategy.com/products/index.htm.

[Mim99] Mimo, P.R. Mistakes to Avoid in Building Data Warehouses. *Cutter IT Journal*, 12(6):36-50, June 1999.

[Mir00] Miranda, R. M. Utilização do Modelo Dimensional para Diagnóstico de Saúde no Município de Belo Horizonte. Monografia de Especialização, Prodabel/IRT-PUC/MG, Belo Horizonte, MG, BR, 2000.

[MO98] Monteiro, R.R., Oliveira, P.C.S. Padrões de Metadados MDIS e MS Repository. Trabalho apresentado na disciplina de mestrado em Informática da UFRJ, Brasil, 1998. Available at URL http://genesis.nce.ufrj.br/dataware.

[Moe01] Moeller, R.A. *Distributed Data Warehousing using Web Technology*. AMACON – American Management Association, USA, 2001. 383 pp. ISBN 0-8144-0588-6.

[MST97] Melo, R.N., Silva, S.D., Tanaka, A.K. *Banco de Dados em Aplicações Cliente-Servidor*. Livraria e Editora Infobook S.A., RJ, Brazil, 1997. 257 pp. ISBN 85-7331-039-1.

[MWM99] Munneke, D., Wahlstrom, K., Mohania, M. Fragmentation of Multidimensional Databases. In *Proceedings of the 10th Australasian Database Conference*, pages 153-164, Auckland, New Zealand, January 1999.

[NCWD84] Navathe, S.B., Ceri, S., Wiederhold, G., Dou, J. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680-710, December 1984.

[NR89] Navathe, S.B., Ra, M. Vertical Partitioning for Database Design: A Graphical Algorithm. In *Proceedings of the 1989 International Conference on Management of Data*, Portland, Oregon, USA, June 1989. *SIGMOD Record*, 18(2):440-450, June 1989.

[OQ97] O'Neil, P., Quass, D. Improved Query Performance with Variant Indexes. In Peckham, J., editor, *Proceedings of the 1997 International Conference on Management of Data*, volume 26 of *SIGMOD Record*, pages 38-49. ACM Press, June 1997.

[Ora] Oracle Corp. Oracle Warehouse. Internet. Available at URL http://www.oracle.com/datawarehouse.

[Orl96] Orli, R.J. Data Extraction, Transformation and Migration Tools. White paper, September 1996. Available at URL http://www.kismeta.com/ex2.html.

[OV99] Özsu, M.T., Valduriez, P. *Principles of Distributed Database Systems.* Prentice Hall, Inc., USA, 2nd edition, 1999. 666 pp. ISBN 0-13-659707-6.

[Pag98] Pagliusi, P.S. Introdução de Mecanismos de Segurança em Sistemas de Correio Eletrônico. Master's thesis, Universidade Estadual de Campinas, Campinas, SP, BR, 1998. 195 pp.

[PB94] Premerlani, W. J., Blaha, M. R. An Approach for Reverse Engineering of Relational Databases. *Communications of the ACM*, 37(5): p.42-49, May 1994.

[PC] Pendse, N., Creeth, R. The OLAP Report. Internet. Available at URL http://www.olapreport.com.

[Pil98] Pilot Software, Inc. An Introduction to OLAP – Multidimensional Terminology and Technology. White paper, 1998. 24pp. Available at URL http://www.pilotsw.com/olap/olap.htm.

[PJ99] Pedersen, T.B., Jensen, C.S. Multidimensional Data Modeling for Complex Data. In *Proceedings of the 15th International Conference on Data Engineering*, pages 336-345, Sydney, Australia, March 1999.

[PP00] Priebe, T., Pernul, G. Towards OLAP Security Design: Survey and Research Issues. In *Proceedings of the 3rd International Conference on Data Warehousing and OLAP*, pages 33-40, Washington, DC, USA, November, 2000.

[PR99] Pourabbas, E., Rafanelli, M. Characterization of Hierarchies and Some Operators in OLAP Environment. In *Proceedings of the 2nd International Workshop on Data Warehousing and OLAP*, pages 54-59, Kansas City, USA, November 1999.

[QGMW96] Quass, D., Gupta, A., Mumick, I.S., Widom, J. Making Views Self-Maintainable for Data Warehousing. In *Proceedings of Parallel and Distributes Information Systems*, pages 158-169, Miami Beach, Florida, December 1996.

[Qua96] Quass, D. Maintenance Expressions for Views with Aggregation. In *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, pages 110-118, Montreal, Canada, June 1996.

[QW97] Quass, D., Widom, J. On-Line Warehouse View Maintenance. In *Proceedings of the International Conference on Management of Data*, pages 393-404, Tucson, Arizona, USA, May 1997.

[RBPE91] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. *Object-Oriented Modeling and Design*. Prentice Hall, Inc., January 1991. 528 pp.

[Ree97] Reese, G. *Database Programming with JDBC and JAVA*. O'Reilly & Associates, Inc., USA, 1st edition, 1997. 224pp. ISBN 1-56592-270-0.

[RK98] Roussopoulos, N., Kotidis, Y. The Cubetree Storage Organization. In *Proceedings of the 24th Very Large Data Bases Conference*, page 700, New York, USA, August 1998.

[Rou98] Roussopoulos, N. Materialized Views and Data Warehouses. *SIGMOD Record*, 27(1):21-26, March 1998.

[RS97] Ross, K.A., Srivastava, D. Fast Computation of Sparse Datacubes. In *Proceedings of the 23rd Very Large Data Bases Conference*, pages 116-125, Athens, Greece, August 1997.

[RSSB00] Roy, P., Seshadri, S., Sudarshan, S., Bhobe, S. Efficient and Extensible Algorithms for Multi Query Optimization. In *Proceedings of the 2000 International Conference on Management of Data*, pages 249-260, Dallas, Texas, USA, May 2000.

[RTZ99] Ravat, F., Teste, O., Zurfluh, G. Towards Data Warehouse Design. In *Proceedings of the 1999 International Conference on Information and Knowledge Management*, pages 359-366, Kansas City, USA, November 1999.

[Rud96] Rudin, K. What's New in Data Warehousing. *DBMS Data Warehouse Supplement*, 9(9), August 1996. Available at URL http://www.dbmsmag.com/9608d00.html.

[San] San Diego Gas & Electric. Internet. Available at URL http://www.sdge.com.

[Sar97] Sarawagi, S. Indexing OLAP Data. *IEEE Data Engineering Bulletin*, 20(1):36-43, March 1997.

[SC98] Siméon, J., Cluet, S. Using YAT to Build a Web Server. In *Proceedings of the 1998 International Workshop on the Web and Databases*, Valencia, Spain, March 1998.

[SDJL96] Srivastava D., Dar S., Jagadish H.V., Levy A.Y. Answering Queries with Aggregation Using Views. In *Proceedings of the 22nd Very Large Data Bases Conference*, pages 318-329, Mumbai, India, September 1996.

[SDN98] Shukla, A., Deshpande, P.M., Naughton, J.F. Materializes View Selection for Multidimensional Datasets. In *Proceedings of the 24th Very Large Data Bases Conference*, pages 488-499, New York, USA, August 1998.

[She] Shell E&P Company. Internet. Available at URL http://www.shell.com.

[Sho97] Shoshani, A. OLAP and Statistical Databases: Similarities and Differences. In *Proceedings of the 16th Symposium on Principles of Database Systems*, pages 185-196, Tucson, Arizona, May 1997.

[SL90] Sheth, A. P., Larson, J. A. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183-235, September 1990.

[SL94] Sacramento, E. R., Laender, A. H. R. Uma Abordagem Orientada a Objetos para o Projeto Lógico de Bancos de Dados Relacionais. In *Anais do IX Simpósio Brasileiro de Banco de Dados*, pages 110-126, São Carlos, São Paulo, Brasil, October 1994.

[SM00] Souza, M.G., Marino, M.T. Open Information Model – OIM. Transparências apresentadas na disciplina de mestrado em Informática da UFRJ, Brasil, 2000. 48 transparências. Available at URL http://genesis.nce.ufrj.br/dataware.

[SMKK98] Samtani, S., Mohania, M., Kumar, V., Kambayashi, Y. Recent Advances and Research Problems in Data Warehousing. In Kambayashi, Y., Lee, D.L., Lim, E.-P., Mohania, M.K., editors, *Advances in Database Technology –Proceedings of the ER '98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management*, volume 1552 of *Lecture Notes in Computer Science*, pages 81-92. Springer, 1998.

[SMR00] Stöhr, T., Märtens, H., Rahm, E. Multi-Dimensional Database Allocation for Parallel Data Warehouses. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 273-284, Cairo, Egypt, September 2000.

[SS99] Souza, M.F., Sampaio, M.C. Efficient Materialization and Use of Views in Data Warehouses. *SIGMOD Record*, 28(1):78-83, March 1999.

[SSSB98] Samos, J., Saltor, F., Sistac, J., Bardés, A. Database Architecture for Data Warehousing: An Evolutionary Approach. In Quirchmayr, G., Schweighofer, E., Bench-Capon, T.J.M., editors, *Proceedings of the 9th International Database and Expert Systems Applications Conference*, volume 1.460 of *Lecture Notes in Computer Science*, pages 746-756. Springer, 1998.

[SSU+00] Silva, D.S., Siqueira, S.W.M., Uchôa, E.M.A., Braz, M.H.L.B., Melo, R.N. An Architecture for Data Warehouse Systems Using a Heterogeneous Database Management System. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, pages 65-78, João Pessoa, Paraíba, Brasil, October 2000.

[SSU96] Silberschatz, A., Stonebraker, M., Ullman, J., editors. Database Research: Achievements and Opportunities Into the 21st Century. *Journal of the Brazilian Computer Society*, 3(2):5-10, April 1996.

[TB00] Theodoratos, D., Bouzeghoub, M., A General Framework for the View Selection Problem for Data Warehouse Design and Evolution. In *Proceedings of the 3rd International Workshop on Data Warehousing and OLAP*, pages 1-8, Washington, DC, USA, November, 2000.

[TBC99] Tryfona, N., Busborg, F., Christiansen, J.G.B. starER: A Conceptual Model for Data Warehouse Design. In *Proceedings of the 2nd International Workshop on Data Warehousing and OLAP*, pages 3-8, Kansas City, USA, November 1999.

[Tha99] Thazar Solutions Corporation. The Reality of Virtual Data Warehousing. White paper, 1999. Available at URL http://www.thazar.com/PDFs/Thazar_WP_Virtual_Data_Warehousing.pdf.

[TS97] Theodoratos, D., Sellis, T. Data Warehouse Configuration. In *Proceedings of the 23rd Very Large Data Bases Conference*, pages 126-135, Athens, Greece, August 1997.

[ULM98] Uchôa, E.M.A., Lifschitz, S., Melo, R.N. A Heterogeneous Object Oriented Database System. In Quirchmayr, G., Schweighofer, E., Bench-Capon, T.J.M., editors, *Proceedings of the 9th International Database and Expert Systems Applications Conference*, volume 1.460 of *Lecture Notes in Computer Science*, pages 223-230. Springer, 1998.

[UM99] Uchôa, E.M.A., Melo, R.N. HEROS^{fw}: A Framework for Heterogeneous Database Systems Integration. In Bench-Capon, T.J.M., Soda, G., Tjoa, A.M., editors, *Proceedings of the 10th International Database and Expert Systems Applications Conference*, volume 1.677 of *Lecture Notes in Computer Science*, pages 656-667. Springer, 1999.

[Uni] United States Postal Service. Internet. Available at URL http://www.usps.gov.

[Var00] Vargas, N. Só para os Maiores. *Veja*, 1.641(12):134-136, March 2000.

[Vas00] Vassiliadis, P. Gulliver in the Land of Data Warehousing: Practical Experiences and Observations of a Researcher. In *Proceedings of the 2nd International Workshop on Design and Management of Data Warehouses*, pages 12.1-12.16, Stockholm, Sweden, June 2000.

[VD01] Vaduva, A., Dittrich, K.R. Metadata Management for Data Warehousing: Between Vision and Reality. In *Proceedings of the International Database Engineering & Applications Symposium*, pages 129-135, Grenoble, France, July 2001.

[VVS00] Vetterli, T., Vaduva, A., Staudt, M. Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metamodel. *SIGMOD Record*, 29(3):68-75, September 2000.

[Wal] Wal-Mart Stores, Inc. Internet. Available at URL http://www.wal-mart.com.

[WB97] Wu, M.-C., Buchmann, A.P. Research Issues in Data Warehousing. In *Proceedings of The German Database Conference*, pages 61-82, Ulm, Germany, March 1997.

[Wel95] Weldon, J.L. Managing Multidimensional Data: Harnessing the Power. *Database Programming and Design*, 8(8), 1995.

[WGL+96] Wiener, J.L., Gupta, J., Labio, W.J., Zhuge, Y., Garcia-Molina, J., Widom, J. A System Prototype for Warehouse View Maintenance. In *Proceedings of the Workshop on Materialized Views: Techniques and Applications*, pages 26-33, Montreal, Canada, June 1996.

[Whi99] White, C. Managing Distributed Data Warehouse Meta Data. *Data Management Review Magazine*, February 1999. Available at URL http://www.dmreview.com/master_sponsor.cfm?NavID=55&EdID=159.

[Wid95] Widom, J. Research Problems in Data Warehousing. In *Proceedings of the 4th International Conference on Information and Knowledge Management*, pages 25-30, Baltimore, Maryland, December 1995.

[Wie82] Wiederhold, G. *Database Design*. McGraw Hill, USA, 2nd edition, 1982.

[Wie97] Wiener, J.L. Data Warehousing: What is it? & related Stanford DB research. An overview talk given in the Stanford DB Seminar series, Stanford University, USA, Fall 1997. 55 pp. Available at URL http://www-db.stanford.edu/warehousing/publications.html.

[Wil97] Williams, J. Tools for Traveling Data. *DBMS online Magazine*, June 1997. Available at URL http://www.dbmsmag.com/9706d16.html.

[Xer] Xerox – The Document Company. Internet. Available at URL http://www.xerox.com.

[YW98] Yang, J., Widom, J. Maintaining Temporal Views Over Non-Historical Information Sources for Data Warehousing. In Schek, H.-J., Saltor, F., Ramos, I., Alonso, G., editors, *Advances in Database Technology –Proceedings of the 6th International EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 389-403. Springer, 1998.

[YW00] Yang, J., Widom, J. Temporal Views Self-Maintainable. In Zaniolo, C., Lockemann, P.C., Scholl, M.H., Grust, T., editors, *Proceedings of the 7th International Conference on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, pages 395-412, Springer, 2000.

[ZDN97] Zhao, Y., Deshpande, P.M., Naughton, J.F. An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. In Peckham, J., editor, *Proceedings of the 1997 International Conference on Management of Data*, volume 26 of *SIGMOD Record*, pages 159-170. ACM Press, June 1997.

[ZDNS98] Zhao, Y., Deshpande, P., Naughton, J.F., Shukla A. Simultaneous Optimization and Evaluation of Multiple Dimensional Queries. In *Proceedings of the 1998 International Conference on Management of Data*, pages 271-282, Seattle, Washington, USA, June 1998.

[ZGMHW95] Zhuge, Y., Garcia-Molina, H., Hammer, J., Widom, J. View Maintenance in a Warehousing Environment. In Carey, M.J., Schneider, D.A., editors, *Proceedings of the 1995 International Conference on Management of Data*, volume 24 of *SIGMOD Record*, pages 316-327. ACM Press, June 1995.

[ZGMW98] Zhuge, Y., Garcia-Molina, H., Wiener, J.L. Consistency Algorithms for Multi-Source Warehouse View Maintenance. *Journal of Distributed and Parallel Databases*, 6(1):7-40, January 1998.

[Zha93] Zhang, Y. On Horizontal Fragmentation of Distributed Database Design. In *Proceedings of the 4th Australasian Database Conference*, pages 121-130, Brisbane, Queensland, Australia, February 1993.

[ZWGM97] Zhuge, Y., Wiener, J.L., Garcia-Molina, H. Multiple View Consistency for Data Warehousing. In *Proceedings of the 13th International Conference on Data Engineering*, pages 289-300, Birmingham, UK, April 1997.