DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-2001-08

ASPECTOS CONCEITUAIS EM DATA WAREHOUSE

ISABEL CRISTINA ITALIANO
JOÃO EDUARDO FERREIRA
OSVALDO KOTARO TAKAI

Setembro de 2001

Aspectos conceituais em Data Warehouse

Isabel Cristina Italiano, ici@ime.usp.br
João Eduardo Ferreira, jef@ime.usp.br
Osvaldo Kotaro Takai, otakai@odin.unaerp.br

Instituto de Matemática e Estatística
Universidade de São Paulo

Setembro de 2001

Resumo

A utilização dos Data Warehouses[1] vem aumentando muito rapidamente nos últimos anos. Sua importância, como base para os sistemas de apoio aos processos decisórios, é notadamente relevante quer no âmbito comercial, quer no acadêmico. Este relatório aborda os aspectos conceituais que servem como ponto de partida para o entendimento e a modelagem de um Data Warehouse. São apresentadas, também, discussões sobre as várias opções dos modelos mais utilizados e as vantagens e desvantagens das abordagens estática e dinâmica em sua implementação.

PALAVRAS CHAVE: Data Warehouse; OLAP; Star Schema; Snowflake Schema; Materialized Views.

1. Introdução

Warehousing[2] é uma técnica utilizada para recuperação e integração de dados a partir de fontes distribuídas, autônomas e, possivelmente, heterogêneas [ZGMHW94].

Estes dados são armazenados em um grande depósito chamado de *Data Warehouse*. Um data warehouse sumaria os dados que são organizados em dimensões, disponibilizando-os para consultas e análises através de aplicações OLAP (On-Line Analytical Processing) e sistemas de suporte à decisão [GM96].

[1] O termo *Data Warehouse*, que pode ser traduzido para *armazém de dados*, é sempre encontrado na literatura escrita em português, na sua forma original, em inglês.
[2] O termo *Warehousing* não possui tradução adequada para o português.

Os data warehouses vêm sendo muito utilizados pelas empresas, já que proporcionam um alicerce sólido de integração de dados corporativos e históricos para a realização de análises gerenciais. Sua construção e implementação são feitas de uma maneira passo a passo, organizando e armazenado os dados sob uma perspectiva de longo prazo. Assim, partindo-se dos dados históricos básicos, pode-se realizar análises de tendências [IH97].

Por sua característica básica, que é a integração de dados provenientes de várias fontes diferentes, a etapa mais complexa na implementação de um data warehouse é o processo de carga. Neste processo, os dados distribuídos pelos vários ambientes operacionais (bases de dados de produção, que contêm os dados utilizados pelos vários sistemas transacionais de uma empresa) devem ser selecionados, trabalhados com o objetivo de padronização e limpeza, transferidos para o novo ambiente e finalmente carregados, sempre atendendo ao padrão da modelagem utilizada para o data warehouse. Este processo é feito periodicamente, sendo que sua freqüência depende de vários fatores relacionados ao modelo de negócios utilizado pela empresa e, normalmente, não é menor que 24 horas. Desta forma, podemos dizer que os dados armazenados no data warehouse são, para todos os propósitos práticos, uma longa série de fotografias, tiradas ao longo do tempo. Uma vez que os dados são armazenados no data warehouse, eles não mais sofrem atualizações, sendo, portanto, um ambiente apenas de carga e acesso.

Após sua criação e primeira carga, o data warehouse passa a sofrer cargas incrementais que devem refletir o ambiente operacional ao longo do tempo tornando-o uma imensa base de dados para os sistemas de apoio à decisão.

Torna-se claro, portanto, que o data warehouse tem características de um ambiente estático, que só reflete as alterações ocorridas no ambiente operacional após períodos pré-definidos. Este fato em si não é tão grave, já que em várias áreas de negócio, as análises são feitas baseadas em resumos mensais, por exemplo. O que é um fator crítico é a alta complexidade do processo de carga que se transforma em um ponto muito suscetível à introdução de erros, que podem levar ao colapso de todo o processo de tomada de decisão.

Como o data warehouse sofre periodicamente novas cargas, aumentando constantemente seu tamanho, surge mais um grande problema em sua utilização, que é a dificuldade em responder às consultas dos usuários de forma rápida e eficiente. As otimizações nos componentes que envolvem este processo, visando agilizar as respostas às consultas, vêm sendo projetadas em vários níveis:

1. mudanças nas estruturas físicas que compõem a base de dados;
2. ferramentas de análise mais eficientes e
3. melhorias no modelo de dados implementado no data warehouse;

Os itens 1 e 2 acima têm sido tratados pelos fornecedores de bancos de dados e de ferramentas de análise, respectivamente e, portanto, devemos nos concentrar no item 3 apresentado.

Os problemas citados anteriormente, ou seja, a alta complexidade do processo de carga aliada às características estáticas do data warehouse e a necessidade de melhorar o modelo de dados implementado, são a motivação para este trabalho.

Com o objetivo de conferir ao data warehouse uma característica mais dinâmica e otimizar o acesso aos dados, vários estudos têm sido feitos indicando o uso de *visões materializadas*[3] como alternativa viável.

Uma visão é uma relação derivada, definida em termos de relações base, que é computada todas as vezes em que uma referência a ela é feita. Uma visão é dita materializada quando ela é realmente armazenada na base de dados em vez de ser computada a partir das relações base em resposta a consultas [QGMW97]. Uma visão materializada pode ser vista como um cache – uma cópia dos dados que pode ser acessada rapidamente. Os data warehouses podem, portanto, armazenar estas visões materializadas com o objetivo de possibilitar acesso rápido à informação que está integrada a partir de diferentes fontes de dados distribuídas [DEB95].

Existem duas abordagens diferentes no que se refere à utilização de visões materializadas nos data warehouses. A primeira delas define o próprio data warehouse como um conjunto de visões materializadas baseadas nos dados dos ambientes operacionais [QGMW97, Gup97]. A segunda abordagem propõe um conjunto compartilhado de visões materializadas definidas a partir de uma análise das consultas mais freqüentes executadas no data warehouse. Estas visões, por estarem materializadas, agilizariam o acesso aos dados necessários para a realização da consulta [YKL96].

O uso de visões materializadas nas abordagens citadas acima traz vários aspectos que serão cuidadosamente analisados e resolvidos, entre eles:

1. a seleção das visões mais adequadas levando-se em conta uma análise dos custos de manutenção e os benefícios de cada visão e os algoritmos disponíveis;
2. os mecanismos que permitem a propagação correta quando da atualização das fontes de dados base para vários tipos de visões, preferencialmente de forma online;
3. a manutenção das visões de forma dinâmica e preferencialmente autônoma utilizando o conceito de *self maintainable views*[4].

Este relatório confronta as características estáticas dos data warehouses atuais com a abordagem mais dinâmica implementada através das visões materializadas, sendo que a base conceitual é tratada no capítulo 2, incluindo o modelo formal da base de dados multidimensional. O capítulo 3 apresenta as características da abordagem estática do Data Warehouse com seus elementos básicos e discute as implementações típicas (Star Schema e Snowflake Schema) e suas variações. No capítulo 4 pode-se encontrar os elementos para a implementação de uma abordagem dinâmica através da utilização de visões materializadas: como selecionar e manter estas visões, tornando o Data Warehouse sincronizado com as bases de dados operacionais.

[3] A expressão *visões materializadas* foi traduzida do inglês *materialized views*.
[4] A expressão *self maintainable views*, que se refere a visões que possuem a capacidade de auto manutenção, será utilizada em sua forma original, em inglês.

2 Conceitos

2.1 O que é o Data Warehouse

De acordo com [AV98], o data warehouse existe para responder as questões que as pessoas têm sobre os negócios. Esta função contrasta fortemente com o propósito dos sistemas transacionais que as empresas utilizam e requer que o desenho ou o modelo de dados do data warehouse siga princípios completamente diferentes. As técnicas de modelagem dimensional, se aplicadas corretamente, garantem que o desenho do data warehouse reflita a forma de pensar dos gerentes de negócio e possa ser utilizado para responder suas questões.

Em todas as empresas, o processo de criação de negócios é composto por uma série de eventos que caracterizam suas principais atividades. A natureza e freqüência destes eventos variam conforme o tipo de negócio em que uma empresa está envolvida: um produto é manufaturado, uma conta é creditada enquanto outra é debitada, um assento é reservado, um pedido é incluído etc. O controle e processamento corretos destes eventos são críticos para uma empresa, sendo que estas atividades contribuem para seu sucesso ou fracasso. Para isso, a maioria das organizações possui um conjunto de sistemas conhecidos como *sistemas transacionais* ou *sistemas OLTP* (OnLine Transaction Processing) que capturam os eventos de forma individual e todos os detalhes associados a eles. Cada um destes sistemas está encarregado de um tipo diferente de atividades e trata das transações de negócios segundo um conjunto de regras que garanta sua consistência e o armazenamento de todos os detalhes associados. Para um sistema OLTP, os princípios de desenho como normalização e consistência de transações são extremamente críticos. Porém, por tratar as transações de forma individual, os sistemas OLTP falham no que se refere a questões sobre o processo do negócio, como "quais foram os produtos mais vendidos durante o mês passado?" ou "quais produtos estão perdendo *market share?*" ou ainda "quais são nossos clientes mais fiéis?".

O data warehouse é desenhado para suprir estas falhas. Ele é construído para responder questões que não estão limitadas às transações individuais, porém tratam do processo como um todo. Para isso, o desenho do data warehouse deve refletir a forma com que os especialistas enxergam o negócio e este é o ponto chave que distingue um data warehouse de um sistema OLTP. A técnica utilizada para se obter um modelo para o data warehouse que identifique e represente a informação importante para o modelo de negócios é a *modelagem dimensional*. Quando bem definido, o modelo dimensional pode ser uma ajuda de valor incalculável para as áreas de negócio, apoiando e otimizando todo o processo de tomada de decisões.

Conforme [Kel94], um data warehouse corporativo pode ser definido em termos de seis características básicas que o diferenciam dos outros sistemas na empresa. Os dados em um data warehouse são:

1. *Separados* dos sistemas transacionais da empresa e populados a partir destes;
2. *Disponíveis*, na sua totalidade, para a atividade de serem interrogados pelos usuários de negócios;

3. *Integrados* para ser uma base única e padrão para o modelo da empresa;
4. *Associados à informação temporal* e a períodos de tempo definidos, como fechamentos mensais ou baseados no ano fiscal;
5. *Orientados por assunto*, ou seja, organizado para descrever o desempenho do negócio;
6. *Acessível* aos usuários que tenham um conhecimento limitado de sistemas computacionais ou estruturas de dados.

Além destas características, podemos citar sua *não-volatilidade*, ou seja, uma vez que o dado foi carregado no data warehouse, não deve mais sofrer alterações.

2.2 O modelo dimensional e suas implementações [AV98]

A modelagem de um data warehouse normalmente utiliza uma abordagem diferente da modelagem entidade-relacionamento, que é a maneira convencional para desenhar uma base de dados relacional contemplando toda a teoria de normalização dos dados.

O modelo dimensional (também chamado de multidimensional), utilizado para definir um data warehouse, representa os indicadores importantes para uma área de negócios, que são chamados de fatos ou métricas e os parâmetros, chamados dimensões, através dos quais estas métricas são vistas. A figura 1 mostra um exemplo de modelo dimensional para um processo de pedidos. As métricas definidas estão no quadro central e as dimensões estão representadas nos quadros ao redor das métricas. As métricas são sumariadas ou detalhadas de acordo com o interesse da análise a ser feita sobre os dados. Este modelo é fácil de ser entendido por uma pessoa da área de negócios, já que "as coisas que eu avalio" estão na parte central do diagrama e "as formas de se olhar para elas" estão nos quadros em volta.

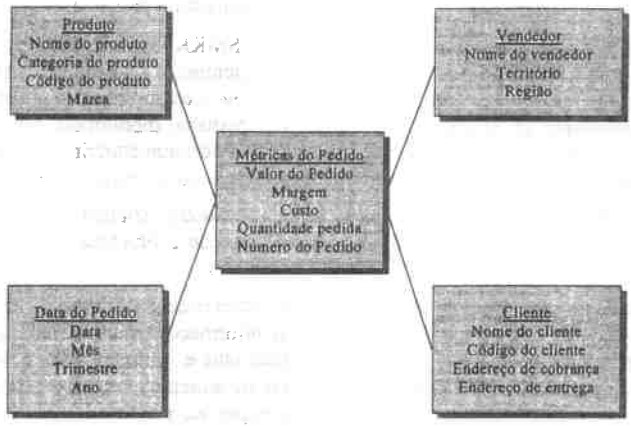

Figura 1 - Modelo dimensional para um processo de pedidos

Fica fácil perceber que estes quadros facilmente se transformarão em tabelas, que podem ser utilizadas para armazenar toda a informação. Um modelo como este não muda muito ao ser implementado em um banco de dados relacional. Cada quadro com os atributos de uma dimensão se torna uma tabela, chamada de *tabela dimensão,* na base de dados e o quadro central se torna uma grande tabela, chamada *tabela fato*, que contém, por vezes, milhões ou bilhões de linhas.

Porém, os modelos dimensionais nem sempre são implementados em bases de dados relacionais. Existem no mercado os *bancos de dados multidimensionais*, ou *MDDBs*, que armazenam as informações em um formato diferente, freqüentemente chamados de *cubos*. Os cubos são construídos de tal forma que, cada combinação de atributos das dimensões com uma métrica ou é precalculado ou é calculado muito rapidamente. Entretanto, a natureza de uma base de dados multidimensional também significa que não é possível manipular volumes de dados extremamente grandes já que, uma transação de análise dos dados, com uma ferramenta OLAP, que envolva um grande volume de dados vai consumir grande quantidade de memória ou simplesmente não se efetua. Além disso, o número de atributos dimensionais armazenados em um cubo pode impactar o tamanho e o desempenho do cubo.

Uma das alternativas para solucionar estes problemas pode ser a implementação do modelo dimensional em um banco de dados relacional e, após isto, utilizá-lo como fonte para os cubos. Esta abordagem é muito utilizada em empresas que querem executar análises em pequenos subconjuntos de um grande conjunto de dados armazenados em um data warehouse. Quando esta abordagem é implementada, o data warehouse como um todo fica armazenado no banco de dados relacional, enquanto que os cubos, contêm partes ou segmentos do data warehouse que são chamados de *data marts*.

Uma outra alternativa é utilizar uma ferramenta de consulta acessando o data warehouse no banco relacional transformando o resultado da consulta em um cubo que permita a análise rápida dos dados. Estes cubos podem ser gerados no computador do usuário ou em um servidor de aplicações. Esta abordagem é interessante para os usuários, já que não requer a administração centralizada destes cubos. É importante notar, porém, que o usuário fica sujeito aos limites de capacidade de processamento de seu PC ou do servidor de aplicações.

Em [SMKK98] tem-se uma arquitetura para o Data Warehouse, representada na figura 2. Nesta arquitetura, os dados são provenientes dos sistemas operacionais. Estas fontes são conectadas a *wrappers* ou monitores que efetuam o processo de seleção, transformação e limpeza dos dados. Além disso, monitoram as alterações nas fontes de dados propagando-as a um componente integrador, que combina os dados selecionados a partir das diferentes fontes operacionais. Estes dados, já consistentes, são propagados para o warehouse.

O metadados contém informações relevantes sobre a criação, gerenciamento e uso do data warehouse e funciona como uma ponte entre os usuários do data warehouse e os dados nele contidos.

O warehouse pode ser acessado através de um servidor OLAP, que tem como objetivo, apresentar as informações multidimensionais para as ferramentas de acesso, análise, geradores de relatórios, planilhas e ferramentas de mineração de dados (*Data Mining tools)*. Basicamente, o servidor OLAP interpreta as consultas dos usuários convertendo-as em instruções adequadas, muitas vezes complexas, para o acesso ao data warehouse. Atualmente existem dois tipos de solução para implementação para acesso ao repositório de data warehouse. Uma é a utilização de modelos relacionais associados a tecnologias de buscas multidimensionais em cubos pré-construídos

(MOLAP). Outra é a utilização de gerenciadores relacionais incrementados com tecnologias de índices bitmap e recuperação de dados com listas invertidas(ROLAP).

Na figura 2, podemos identificar as várias camadas que compõem tal arquitetura:

1. Ferramentas de acesso para os usuários;
2. Servidor OLAP(MOLAP-ROLAP)
3. Sistema de gerenciamento do data warehouse e ferramentas para selecionar, transformar, limpar, integrar e copiar os dados;
4. Dados dos sistemas operacionais.

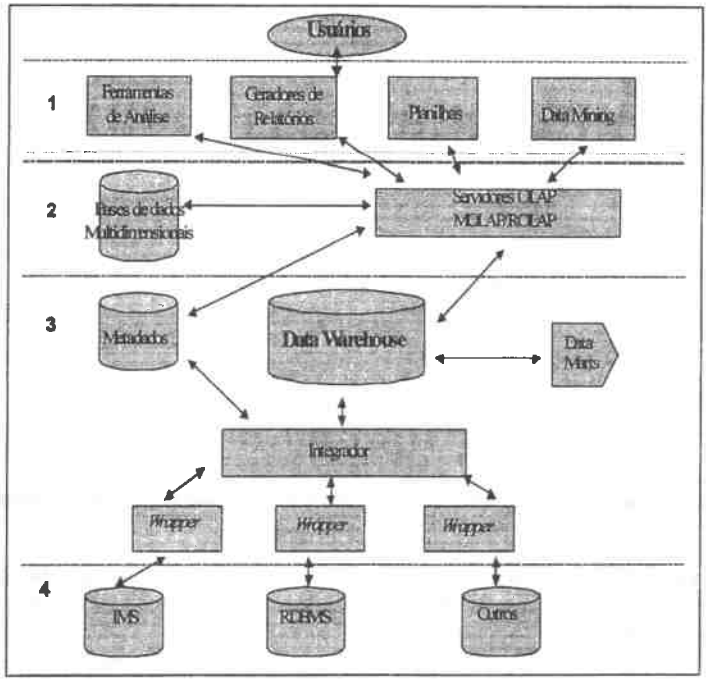

Figura 2 – Arquitetura de data warehouse proposta por [SMKK98]

2.3 O modelo formal da base de dados multidimensional

[BPT97] define formalmente o modelo multidimensional implementado em uma base de dados relacional, que será apresentado nesta seção.

A estrutura básica deste modelo pode ser representada por um diagrama entidade-relacionamento como na figura 3.

Definição 2.3.1 *Uma* **base de dados multidimensional** *é uma coleção de relações D_1,..., D_n, F, onde:*

- *Cada D_i é uma* **tabela dimensão**, *isto é, uma relação caracterizada por um identificador que identifica unicamente cada tupla (d_i é a chave primária de D_i).*
- *F é uma* **tabela fato**, *isto é, uma relação que conecta todas as tabelas D_1,..., D_n; o identificador de F é composto pelas chaves estrangeiras d_1,..., d_n de todas as tabelas dimensão conectadas. O esquema de F contém um conjunto de atributos adicionais V (que representam os valores sobre os quais serão aplicadas as funções de agregação).*

As tabelas dimensão podem conter hierarquias.

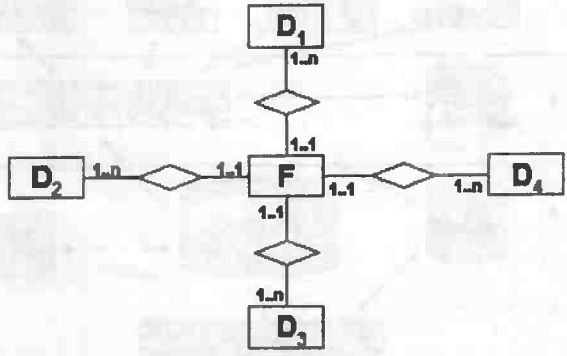

Figura 3 – Representação Entidade-Relacionamento de uma base de dados multidimensional

Definição 2.3.2 *Seja D uma tabela dimensão com o identificador d. Uma* **hierarquia de atributos** *em D é um conjunto de dependências funcionais $FD_D = \{fd_0, fd_1, ..., fd_n\}$, onde cada fd_i é caracterizado por dois conjuntos de atributos $A^l_i \subset$ Attr(D) e $A^r_i \subset$ Attr(D) (chamados respectivamente de lado esquerdo e lado direito da dependência); a dependência é representada por $fd_i : A^l_i \rightarrow A^r_i$.*

Cada dependência funcional fd_i é uma restrição ao conteúdo da tabela dimensão D, sendo que: para cada par de tuplas t_1 e $t_2 \in D$, $t_1[A^l_i] = t_2[A^l_i] \Rightarrow t_1[A^l_i] = t_2[A^r_i]$. Uma dependência fd_0 com $A^l_0 = \{d\}$ e $A^r_0 = \{Attr(D) - d\}$ estará sempre presente em FD_D. As dependências funcionais devem ser acíclicas, isto é, o grafo obtido pelo desenho de um arco partindo de a_x e chegando em a_y, deve ser acíclico, se $\exists fd_i \in FD_D \mid a_x \in A^l_i \wedge a_y \in A^r_i$.

Exemplo: Vamos considerar um exemplo prático, retirado de [Kim96], a base de dados multidimensional para uma grande cadeia de lojas de varejo. Com um grande número de lojas, cada uma delas sendo um supermercado que vende uma ampla variedade de

diferentes produtos. A base de dados multidimensional armazena informação sobre cada venda, por loja, por dia e considera também as promoções em cada produto vendido. Podemos identificar as seguintes dimensões: *Product*[5], que caracteriza cada produto vendido, *Store*, que caracteriza cada ponto de venda, *Time* e *Promotion*, que descreve as características das promoções dos produtos. A tabela fato fornece as informações sobre as vendas, sobre as quais serão realizadas as análises financeiras. Inclui identificadores para todas as dimensões e vários atributos descrevendo as vendas (como valor da venda, quantidade vendida, etc.)

Se considerarmos a dimensão *Store*, do exemplo acima, com chave s e restrita a um conjunto de atributos $\{z, c, s_t, n\}$, que representam respectivamente *zip code*, *county*, *state* e *number of sale clerks*. A dimensão tem a seguinte hierarquia de atributos:

$$\{fd_0 : s \rightarrow \{z, c, s_t, n\}, fd_1 : z \rightarrow c, fd_2 : c \rightarrow s_t\}$$

Definição 2.3.3 *Uma* **hierarquia de atributos da base de dados multidimensional** FD_{DB} *é a união das hierarquias de atributos* DF_{D_j} *de todas as dimensões* D_j *existentes na base de dados multidimensional.*

[5] Por razões didáticas, os nomes de tabelas e atributos serão mantidos no idioma original, inglês.

3 A abordagem estática do data warehouse atual

3.1 Características

Conforme já comentado, o modelo dimensional é poderoso, pois reflete a maneira de pensar dos especialistas de negócios e responde às suas necessidades de informações. A tecnologia relacional de bancos de dados possibilita ao data warehouse ser utilizado para responder as questões de forma rápida e precisa. Para isso, são necessários três componentes essenciais, a saber:

1. Os dados provenientes das várias fontes distribuídas pela empresa e armazenados em um único local;
2. Ferramentas que possibilitem a análise das informações armazenadas de forma rápida, flexível com alta qualidade de apresentação e
3. O conhecimento do especialista de negócios.

Existem inúmeras ferramentas, como as citadas no item 2 acima, disponíveis no mercado e são chamadas de OLAP (Online Analytical Processing). Estas ferramentas permitem ao usuário visualizar os vários níveis de detalhamento da informação, sob as visões das diferentes dimensões definidas no modelo e têm sido alvo de vários trabalhos acadêmicos. O conhecimento do especialista de negócios é outro componente essencial, já que apenas ele pode tirar as conclusões das informações apresentadas, com o objetivo de tomada de decisões da corporação.

Em linhas gerais, o processo de implementação de um data warehouse está dividido nas seguintes etapas:

1. Levantamento do processo de negócio a modelar;
2. Definição dos modelos conceitual, lógico e físico;
3. Definição do processo de carga;

Dentre todas as etapas citadas acima, a que envolve o processo de carga é de longe a mais complexa. Além de trazer os dados de vários sistemas transacionais diferentes, o processo engloba atividades de verificação, padronização, limpeza e transformação dos dados antes da carga. A definição da periodicidade da carga depende da natureza do negócio que compõe o data warehouse e do tipo de informação armazenada. Algumas áreas de negócios requerem carga diária, enquanto que outras necessitam apenas de cargas mensais. Seja qual for a periodicidade escolhida, fica claro que o data warehouse não está sincronizado tempo real com os sistemas transacionais. Para algumas aplicações, esta característica estática do data warehouse não compromete o resultado das análises porém, para outras, um data warehouse dinâmico, ou seja, sincronizado com os sistemas transacionais, é essencial. Para implementar um data warehouse dinâmico necessitaríamos definir uma arquitetura que permitisse propagar as atualizações nas bases transacionais no instante em que elas ocorrem.

Atualmente, podemos dizer que os data warehouses implantados são de natureza estática, tendo processos de carga de alta complexidade e que requerem um grande tempo de processamento.

As próximas seções deste trabalho detalham vários aspectos envolvidos na modelagem do data warehouse.

3.2 Conceitos da modelagem [Tan97]

A modelagem dimensional combina tabelas fato que armazenam dados históricos temporais (normalmente numéricos), indexadas por chaves dimensionais que estão descritas nas tabelas dimensão correspondentes. As tabelas dimensão contêm informações como, por exemplo: períodos de tempo, produtos, mercados, organizações, contas, vendedores e clientes e inclui as descrições e os atributos destas dimensões. Além disso, as tabelas dimensão contemplam toda a estrutura da dimensão, como os agrupamentos dos produtos em marcas e em categorias, as cidades em estados, em regiões e em países e assim por diante.

As tabelas fato contêm as métricas ou os fatos a serem analisados dentro do modelo de negócios. Cada um destes fatos está diretamente relacionado às dimensões, que descrevem suas condições de ocorrência. Todo o relacionamento entre a tabela fato e as tabelas dimensão é feito através de chaves.

Uma consulta executada neste modelo dimensional, geralmente se inicia por uma pesquisa nas tabelas dimensão, aplicando-se os filtros de valores e obtendo-se como resultado um conjunto de chaves. Após isso, o acesso é feito à tabela fato, garantindo assim a precisão no acesso aos dados através de uma estrutura completa de chaves, eliminando-se um *table scan* e, com isso, obtendo-se o melhor desempenho possível de uma tecnologia relacional. O conceito de armazenamento das dimensões separadamente garante que a base de dados trate os vetores esparsos de maneira eficaz, isto é, sem armazenar vazios, assegurando o mais eficiente acesso possível.

O modelo dimensional pode ser implementado utilizando vários tipos de esquemas diferentes. Provavelmente, o *star schema*, apresentado por R. Kimball [Kim96], seja o primeiro esquema utilizado para representar o data warehouse implementado em um banco de dados relacional.

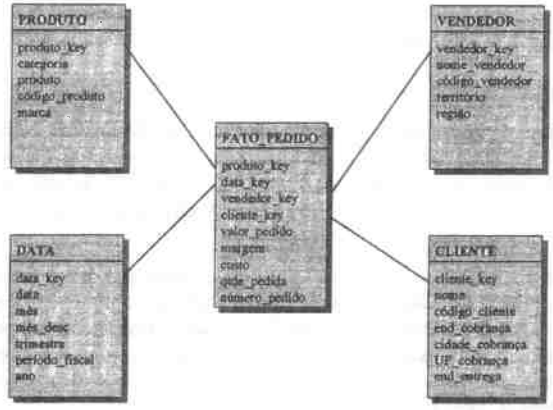

Figura 4 - Esquema para um processo de pedidos

Apesar de ser o mais conhecido, o star schema não é o único. De fato, existe uma série de variações que serão analisadas posteriormente neste trabalho. Porém, antes de se iniciar a modelagem em um esquema particular, temos que definir o processo de negócio a ser modelado.

A figura 4 mostra um exemplo de esquema para um processo de pedidos. A tabela central é a tabela fato e as tabelas em torno da fato são as dimensões. Neste exemplo estão representadas as dimensões Produto, Tempo (data), Cliente e Vendedor (incluindo informações de ordem geográfica). Os fatos representados são: valor do pedido, margem, custo, quantidade pedida e número do pedido.

Após definido o escopo do data warehouse, os próximos passos são: definir a granularidade das informações representadas na tabela fato, identificar as dimensões e enumerar as métricas ou fatos. A seguir daremos uma descrição sucinta sobre cada uma destas etapas e, na seqüência, mostraremos os tipos de esquemas básicos e suas variações.

3.2.1 A granularidade das informações [AV98]

Como já dissemos, uma das primeiras decisões feita para modelar o data warehouse está relacionada com o nível de detalhe das métricas a serem armazenadas. Este nível de detalhamento é conhecido como granularidade da tabela fato. É muito importante que todas as linhas na tabela fato sejam armazenadas com informações exatamente no mesmo nível de detalhes. Como exemplo, um processo de pedidos teria sua granularidade definida no nível de detalhe da linha individual do pedido. Se armazenássemos diferentes níveis de detalhamento na tabela fato, a utilização da base de informações ficaria bastante prejudicada. Vamos supor que tenhamos armazenado a maioria das informações no nível da linha de detalhe do pedido, porém algumas informações foram armazenadas no nível do cabeçalho do pedido. No nível de detalhe da linha do pedido existe um relacionamento com cliente, produto, representante comercial e tempo. Porém, no nível do cabeçalho do pedido, o relacionamento com os produtos específicos que foram pedidos desaparece. As informações relevantes que estejam em diferentes níveis de granularidade deveriam ser armazenadas em tabelas fato diferentes.

Em geral, podemos dizer que, a aderência de uma tabela fato a uma certa granularidade requer que as chaves que relacionam as linhas da tabela fato às linhas das dimensões nunca sejam nulas. Um relacionamento opcional a uma dimensão geralmente indica um problema de granularidade.

3.2.2 As dimensões

Uma vez que o nível de granularidade esteja definido, a escolha das dimensões a serem utilizadas deve ser o próximo passo. O ideal seria definir uma dimensão com um grande número de atributos, representando um rico conjunto de detalhes sobres o processo de negócio. É comum que as dimensões apresentem 100 ou até mesmo 200 atributos em uma única tabela dimensão [AV98].

As tabelas dimensão contêm informações sobre as dimensões dos dados, ou seja, tempo, produto, mercado, contas e assim por diante e devem ser desenhadas a partir da perspectiva do usuário. Por esta razão, os atributos e suas descrições devem ser definidos de forma significativa para o usuário e adequados para a posterior exibição em relatórios. As principais funções da tabela dimensão são reunir os atributos que serão utilizados para qualificar as consultas e cujos valores serão utilizados para agrupar e sumariar as métricas [AV98].

Cada tabela dimensão deve ter múltiplos atributos que contenham valores ou textos que possam ajudar a descrever a chave. Estas colunas de atributos são utilizadas para filtrar o conteúdo da dimensão. Além disso, a utilização de atributos do tipo inteiro, quando apropriados, favorecem as operações de filtragem como *maior que*, *menor que* e *entre* [Tan97].

Os atributos de uma dimensão podem compor uma hierarquia ou ser apenas descritivos. Por exemplo, em uma dimensão *produto*, a hierarquia pode ser composta pelos atributos *item*, *marca*, *tipo* e *divisão*. A hierarquia definida na dimensão é requisito básico para as funções de agregação das métricas contidas na tabela fato. Através dos agrupamentos dos elementos na hierarquia, os usuários podem analisar os fatos em um nível maior ou menor de detalhes, conforme sua necessidade específica. Este conceito de hierarquia, que permite a implementação de agregações das métricas, é muito importante também para a dimensão *Tempo* e, vale ressaltar que, conforme [Rad96], é muito raro um modelo dimensional que não inclua a dimensão *Tempo* como uma dimensão fundamental.

Deve-se resistir ao impulso de aplicar as regras de normalização nas tabelas dimensão [AV98]. O processo de normalização, separando os atributos em várias tabelas diferentes faz com que as consultas fiquem bem mais complexas, o banco de dados leve mais tempo para recuperar os dados e, por conseqüência, os usuários esperem mais tempo pelas respostas. Este custo é muito alto como resultado da economia de apenas uns poucos bytes em uma tabela dimensão que, em comparação com uma tabela fato, é minúscula.

Manter as tabelas dimensão desnormalizadas faz com que as hierarquias naturais contidas nos dados fiquem bem definidas. Em nosso exemplo de pedidos, podemos notar alguns exemplos como a dimensão Tempo (Data) inclui os atributos Ano, Trimestre, Mês e Data (que inclui o dia), juntamente com outros atributos relacionados a uma data específica. Estes componentes não foram colocados em uma série de tabelas progressivamente mais detalhadas. Em vez disso estão em uma única tabela que deixa clara a hierarquia. Enquanto que o ano de 1998 pode aparecer na tabela 365 vezes, isto ficará invisível para o usuário. Quando as métricas forem sumariadas por ano, apenas uma ocorrência de 1998 aparecerá no relatório.

Um atributo muito importante da tabela dimensão é sua chave. A chave primária de uma tabela dimensão dever ser sempre um atributo único e definido pelo sistema com um valor genérico. Por questões de desempenho, não se utilizam chaves compostas por várias partes nem tampouco chaves concatenadas. Também não são utilizados as chaves ou os identificadores provenientes de outros sistemas como, por exemplo, código do cliente ou código do produto. Existem várias razões para se utilizar chaves genéricas em vez de chaves com valores significativos. Em caso de alterações de atributos de um cliente como, por exemplo, seu endereço, teremos que dar um tratamento especial e inserir este mesmo cliente, com seu novo endereço, com outra chave. Se estivermos utilizando o código do cliente como chave primária isto não será possível. Os tratamentos de alterações serão analisados neste trabalho, em seções posteriores.

3.2.3 Os fatos

Após identificarmos a granularidade e as dimensões, é o momento de focalizarmos a tabela fato. Para isso, iniciamos definindo quais as métricas ou fatos que queremos avaliar no data warehouse. Estes fatos são os números que serão analisados através das diferentes dimensões [AV98]. De acordo com [Rad96], a seleção dos fatos para compor o modelo do data warehouse é

relativamente simples: uma vez que a área de negócios esteja definida, a lista de fatos a serem utilizados responde a questão: "*O que estamos avaliando?*".

Podemos ver em nosso exemplo, representado nas figuras 1 e 4, em suas respectivas tabelas centrais, quais são os valores relevantes para a análise do processo de pedidos.

Existem três tipos de métricas ou fatos e, após discuti-los na próxima seção, estaremos analisando outros elementos de uma tabela fato na seção seguinte.

3.2.3.1 Os três tipos de métricas

As métricas mais comuns são as *completamente aditivas*. Dizemos que uma métrica é completamente aditiva, quando faz sentido sumarizá-la adicionando seus valores ao longo de qualquer dimensão. Em nosso exemplo de pedidos, o valor do pedido, a margem, o custo e a quantidade pedida são todas métricas completamente aditivas. Apesar de serem armazenadas na tabela fato para um determinado produto, um cliente, um vendedor e uma data específica, podemos facilmente sumarizá-las da maneira que nos interesse. Para verificar os pedidos de um determinado mês, basta adicionar os valores dos pedidos de todas as datas daquele mês. O mesmo ocorre se quisermos verificar a margem obtida para uma determinada categoria de produtos. Basta adicionar os valores de margem para todos os produtos de uma determinada categoria.

As métricas completamente aditivas são bastante úteis e poderosas, já que não existem limitações em sua utilização. Podem ser facilmente sumariadas para qualquer combinação de valores das dimensões.

Em contraste total com este tipo de métrica, temos as métricas *não aditivas*. As métricas não aditivas não podem ser adicionadas ao longo dos valores das dimensões. Vamos considerar a métrica margem, expressa como uma porcentagem de vendas, a qual chamaremos de *margem percentual*. Esta métrica representa a margem como percentual e não mais como valor expresso na moeda corrente. Seria muito simples adicionarmos esta métrica à nossa tabela fato, porém esta métrica teria pouca utilização, já que não poderíamos sumarizá-la de acordo com a dimensão de nosso interesse. Por exemplo, em uma determinada data, um vendedor vende a um cliente 4 tipos diferentes de produtos, cada um deles com uma margem percentual de 25%. Não faz sentido adicionarmos os quatro valores de margem percentual para calcularmos a margem total para este pedido.

Aparentemente podemos concluir que este tipo de métrica não pode ser utilizado em nossa tabela fato. Na verdade, esta métrica é derivada de outras duas métricas que são aditivas: margem e valor do pedido. A solução é, portanto, armazenar na tabela fato apenas seus componentes, que são completamente aditivos, sendo que o cálculo para expressar a margem percentual deverá ser feito pela aplicação, que neste caso é a divisão da margem pelo valor.

O terceiro tipo de métrica é a *semi-aditiva*. A métrica semi-aditiva pode ser sumariada ao longo de determinadas dimensões, porem não todas. Vamos considerar como exemplo o gerenciamento de saldo das contas de um banco. O saldo é armazenado no final de cada dia, para cada cliente, por conta ao longo do tempo. Em alguns casos este saldo é aditivo. Se um cliente tem uma conta corrente e uma conta poupança, podemos adicionar os saldos de cada conta no final de um dia e obter resultado significativo. É possível também adicionar os saldos de uma determinada agência para obter um panorama da situação geral de cada localidade. Entretanto, não faz o menor sentido

adicionar o saldo de um cliente ao longo do tempo. Por esta razão, a métrica saldo é considerada semi-aditiva.

Segundo [GMR98] este tipo de atributo não aditivo ou semi-aditivo pode ser agregado ou sumariado utilizando-se outros operadores como média, máximo ou mínimo. É o caso da métrica temperatura, que é considerada não aditiva, já que adicionar temperaturas dificilmente faz sentido.

3.2.3.2 Outros elementos da tabela fato

Além das métricas, cada tabela fato tem uma chave primária. Esta chave primária é composta por várias colunas, sendo que cada uma delas corresponde logicamente a uma chave na tabela dimensão. Cada elemento componente da chave deve, portanto, estar representado e descrito em uma tabela dimensão correspondente, que logicamente se une a uma ou mais tabelas fato através de colunas idênticas. Podemos dizer, então, que a chave primária de uma tabela fato é uma chave composta por um subconjunto de chaves estrangeiras para as tabelas dimensão. A dimensão *Tempo* é sempre representada como parte da chave primária.

A figura 5 ilustra uma tabela fato que contém o valor das vendas e a quantidade de unidades vendidas, logicamente unida a uma tabela dimensão de *Mercado* correspondente. Na figura não estão representadas as outras dimensões *Produto* e *Tempo*.

Tabela Fato

TEMPO KEY	PRODUTO KEY	MERCADO KEY	Valor	Quantidade
980715	101	4030	348,00	140
980716	101	4030	287,00	114
980716	101	4010	443,00	170
980717	101	2010	580,00	232
980718	101	2010	686,00	274

Dimensão Mercado

MERCADO KEY	Mercado/Descrição	Região	Estado	Cidade	Loja	Nível
1010	Região Sul	1				4
1020	Região Sudeste	2				4
2010	SP	2	10			3
2020	RJ	2	20			3
3010	São Paulo	2	10	101		2
3020	Campinas	2	10	102		2
4010	Loja ABC	2	10	101	1001	1
4020	Loja DEF	2	10	102	1002	1
4030	Loja GHI	2	10	102	1003	1

Figura 5 – Junção lógica entre tabelas fato e dimensão

Ocasionalmente, não é necessário armazenar nenhuma métrica em uma tabela fato. Conforme [AV98], às vezes, o simples armazenamento de um relacionamento entre as dimensões em um certo ponto do tempo é tudo que uma área de negócios necessita como métrica. Este tipo de tabela fato é a tabela sem fato, ou seja, do original *factless fact table*. O exemplo mais típico deste

tipo de tabela fato é o controle de freqüência de alunos em determinadas disciplinas. Não existe uma métrica associada. Apenas a existência do relacionamento entre as dimensões já é um indicativo suficiente para o processo de análise. Este tipo de tabela encontra-se ilustrado na figura 6.

Figura 6 – Um exemplo de tabela fato do tipo factless

Pode ocorrer que a mesma tabela fato tenha múltiplas chaves estrangeiras, como parte de sua chave primária, que se relacionam com a mesma dimensão. Por exemplo, podemos ter o mesmo fato com múltiplas datas associadas a ele, como no caso da tabela fato de remessa de produtos, onde duas chaves estrangeiras, *data da remessa* e *data do pedido*, estão associadas a uma mesma tabela dimensão *tempo*. A figura 7 ilustra esta situação.

Neste caso, não é necessária a criação de duas tabelas dimensão *data*. Em vez disso, utilizamos visões ou sinônimos para referenciar duas cópias virtuais da mesma tabela dimensão em uma única instrução SQL, no momento da consulta.

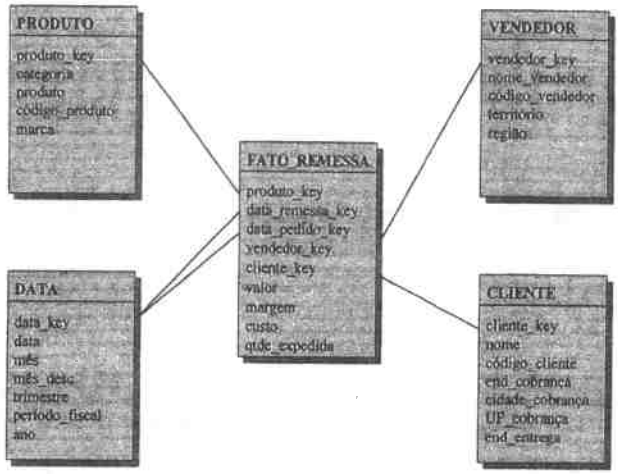

Figura 7 –Relacionamento múltiplo entre uma tabela fato e uma dimensão

3.3 Partindo do MER para o Data Warehouse

A criação de um sistema de informações, que seja bem documentado e que ao mesmo tempo responda aos requerimentos especificados pelos usuários, deve ter como base um cuidadoso desenho conceitual. Apesar disso, a maior parte da literatura sobre data warehouse trata da modelagem lógica e física, sem ao menos mencionar os aspectos relacionados à sua modelagem conceitual.

A falta de interesse em assuntos relacionados ao desenho conceitual deve-se a, principalmente, dois fatores [GMR98]:

1. O uso do data warehouse começou no mundo industrial como resultado das solicitações de usuários que, tipicamente, não davam importância aos aspectos conceituais;
2. O desenho lógico e físico afeta mais a otimização do desempenho dos sistemas, que é o objetivo principal nas aplicações de data warehouse, que o desenho conceitual.

Portanto, é comum encontrarmos aspectos do desenho conceitual inseridos no desenho lógico do data warehouse.

O modelo Entidade Relacionamento (E/R) é amplamente utilizado pelas empresas como um formalismo conceitual que fornece documentação padronizada para os sistemas relacionais de informação e, por esta razão, muitos esforços têm sido feitos com o objetivo de utilizá-los como entrada para o desenho de bases de dados não relacionais.

[SBHD99] apresenta uma extensão ao modelo Entidade Relacionamento, argumentando que este tipo de modelo básico não é apropriado para a modelagem conceitual já que a semântica das caraterísticas principais da modelagem multidimensional não pode ser adequadamente

representada. Em [SBHD99] é apresentada uma especialização do modelo E/R, chamado Modelo Entidade Relacionamento Multidimensional (ME/R), sendo que, para representar a estrutura multidimensional, são definidos conjuntos especializados de relacionamentos e entidades, estendendo assim o modelo ER básico.

[GMR98] também propõe um modelo conceitual, chamado *dimensional fact schema* (DF). Eles utilizam uma notação gráfica e uma metodologia que resulta em um modelo DF, partindo-se de um modelo ER dos dados que será a fonte para a definição do data warehouse. As técnicas utilizadas, apesar de apoiarem conceitos semanticamente ricos, não estão baseadas em um modelo de dados formal (o modelo ER é utilizado apenas como entrada para o processo). Sua argumentação, para a não utilização do modelo ER, baseia-se, inicialmente, na seguinte citação de [Kim96]:

"Os modelos de dados Entidade Relacionamento... não podem ser entendidos pelos usuários e não podem ser navegados de forma proveitosa pelo software de banco de dados. Os modelos Entidade Relacionamento não podem ser utilizados como base para os data warehouses das empresas".

Nesta seção serão apresentados o modelo conceitual gráfico para data warehouse, o *dimensional fact schema*, e uma metodologia semi-automática para construir o modelo DF a partir de um modelo E/R já existente e que descreva um sistema de informações operacional, ambos baseados no trabalho desenvolvido por [GMR98]. Em alguns casos, a documentaçao do E/R não está disponível, porém, a metodologia apresentada pode ser aplicada a partir da base de dados relacional existente, desde que as multiplicidades (-para-um ou -para-muitos) das associações lógicas estabelecidas pelas restrições das chaves estrangeiras sejam conhecidas.

3.3.1 O modelo conceitual para data warehouses

Dentro do escopo desta abordagem, o modelo conceitual para um data warehouse é composto por um conjunto de esquemas de fatos estruturados em árvores. Estes esquemas de fatos têm como componentes os fatos, as dimensões e as hierarquias. Como já discutido anteriormente, um fato é um objeto de interesse da empresa, que será o foco das análises; uma dimensão determina a granularidade adotada para representar os fatos e a hierarquia determina como as instâncias dos fatos podem ser agregadas e selecionadas para apoiar o processo de tomada de decisão.

A estrutura em árvore utilizada para representar estes esquemas tem como raiz um fato. Este fato é representado com uma caixa que indica seu nome e tipicamente, um ou mais atributos numéricos que "medem" o fato a partir de diferentes pontos de vista. A figura 8 representa um esquema de fatos para o fato SALE em uma cadeia de lojas. Os atributos fato *quantity sold* e *returns* contêm os valores que serão utilizados para a análise.

Conforme representado na figura 8, cada vértice diretamente conectado ao fato é uma dimensão. As sub-árvores, que têm como raiz uma dimensão, são as hierarquias. Os atributos, representados por círculos nos vértices, podem assumir um conjunto discreto de valores, sendo que, as linhas que conectam um par de atributos representam um relacionamento do tipo -para-um. Na raiz de cada hierarquia está representado o nível mais detalhado de granularidade da agregação, sendo que, os vértices ao longo da hierarquia definem progressivamente níveis menos detalhados.

Podemos ver, portanto, que o esquema da figura 8 tem três dimensões: *week*, *product* e *store*.

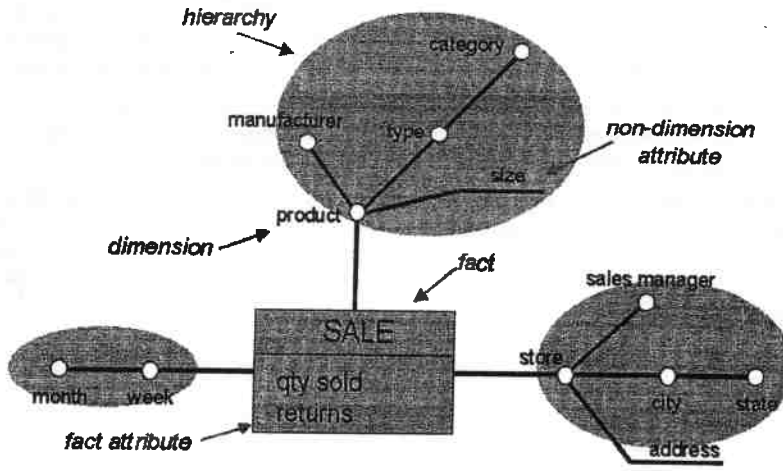

Figura 8 - Um exemplo de esquema de fato com três dimensões para uma cadeia de lojas.

Alguns vértices terminais do esquema de fatos podem ser representados por linhas em vez de círculos, como no caso de *size* e *address* na figura 8. Estes vértices correspondem a atributos não-dimensionais, ou seja, são atributos que contêm informações adicionais sobre um atributo da hierarquia e está conectado a este por um relacionamento do tipo -para-um. Diferentemente dos atributos da hierarquia, os atributos não-dimensionais não podem ser utilizados para agregação.

A opcionalidade nos relacionamentos pode ser representada neste esquema, bastando acrescentar um sinal "*f*" na linha que conecta um par de atributos.

Um fato expressa um relacionamento muitos-para-muitos entre as dimensões. Cada combinação de valores das dimensões define uma instância do fato caracterizada por exatamente um valor para cada atributo. Estas instâncias são as informações básicas representadas em um data warehouse. Na figura 8, uma instância do fato descreve a quantidade de um produto vendida durante uma semana em uma loja e seu correspondente valor de lucro total.

Um esquema de fatos pode não ter nenhum atributo fato. Neste caso, cada instância do fato armazena a ocorrência de um evento. Os esquemas de fatos sem atributos fato correspondem, no modelo lógico, a um tabela de fatos do tipo factless, tipicamente utilizada para acompanhamento de eventos.

3.3.2 A representação da aditividade no modelo

As pesquisas em um DW são tipicamente baseadas na extração de dados sumariados, apresentados na forma de relatórios estruturados, analisados durante os processos de tomada de decisão. Apesar da possibilidade de apresentação dos dados em seu nível mais detalhado, as análises são freqüentemente realizadas em diferentes níveis de abstração e, para isso, as instâncias dos fatos devem ser agregadas. Estas agregações requerem a definição de um operador, para que possamos compor os valores dos atributos caracterizando cada nível de . Em

geral, utilizamos o operador soma para agregar estes valores ao longo da hierarquia, já que a maioria dos atributos fato é aditiva. Um exemplo de atributo aditivo, no exemplo da figura 8, é a quantidade vendida (*qty sold*), ou seja, a quantidade vendida para um determinado gerente de vendas é a soma das quantidades vendidas em todas as lojas gerenciadas por este gerente.

Os conceitos de semi-aditividade e não-aditividade, já apresentados em seções anteriores também serão representados no modelo, lembrando que estes tipos de atributos podem ser agregados utilizando outros operadores como média, valor máximo e mínimo.

Em um modelo DF, os atributos são aditivos em todas as dimensões por *default*. A semi-aditividade e não-aditividade são representadas explicitamente indicando em quais dimensões o atributo não pode ser somado. Se algum outro operador (que não a soma) for utilizado, deve ser indicado explicitamente no modelo. A figura abaixo exemplifica este tipo de atributo e sua representação no modelo:

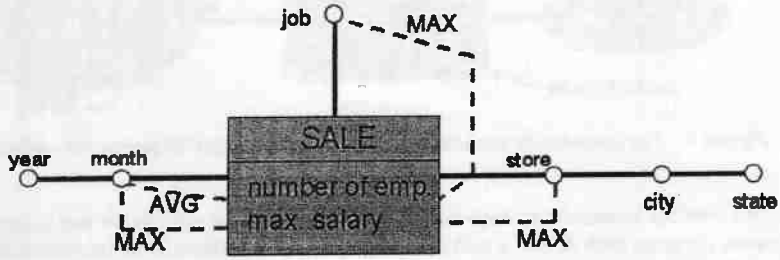

Figura 9 - Representação da semi-aditividade no esquema de fatos

3.3.3 Representando modelos de pesquisa em um esquema de fatos

Os operadores básicos das ferramentas OLAP, para a formulação de pesquisas nos data warehouses são: roll-up, drill down, drill across e slice-and-dice e são utilizados, respectivamente, para agregar atributos fato com o objetivo de visualizar os dados em um nível maior de abstração, desagregar atributos fato para aumentar o nível de detalhamento, relacionar e comparar fatos distintos, selecionar e projetar fatos para reduzir sua dimensionalidade.

Em um esquema de fatos, uma pesquisa pode ser representada por um *query pattern*, composto por um conjunto de marcadores (representados por círculos preenchidos) colocados nos atributos das dimensões. Estes marcadores podem ser colocados dentro de cada hierarquia, para indicar quais os níveis de agregação das instâncias de fatos. Uma dimensão que não possui marcadores indica que nenhum de seus atributos está envolvido na pesquisa. Os atributos não-dimensionais não precisam ser mostrados em um query pattern. A figura 10, abaixo, mostra o *query pattern* representando a seguinte pesquisa: "a quantidade total vendida e a média de lucro por unidade vendida para cada semana e para cada tipo de produto". A média de lucro por unidade vendida é a razão entre o total de lucro e a quantidade vendida (*returns / qty sold*).

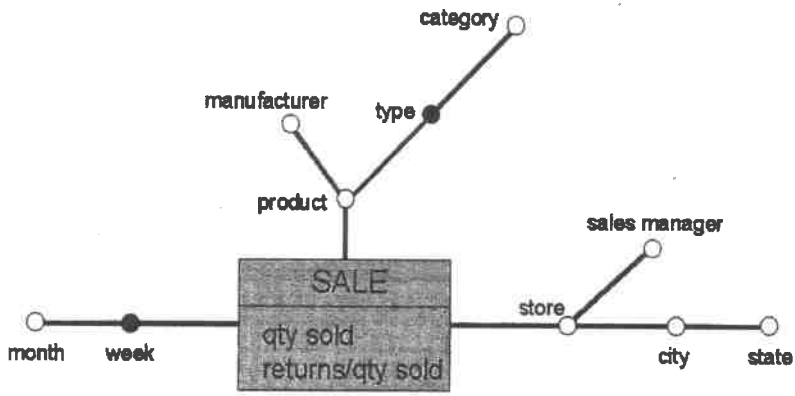

Figura 10 - Representação de um modelo de pesquisa (query pattern)

Caracterizando os processos de consulta, [MF01] descreve as operações típicas do ambiente OLAP:

- *pivoteamento*: orienta a visão multidimensional dos dados - dadas as dimensões (a e b, por exemplo) temos que, cada ponto (x,y) corresponde ao valor agregado do par de dimensões escolhidas.
- *rollup*: aumenta o nível de agregação - diminui o detalhe sobre uma ou mais hierarquias de dimensões;
- *drill-down*: decresce o nível de agregação - aumenta o detalhe sobre uma ou mais hierarquias de dimensões;
- *slice_and_dice*: seleção e projeção - corresponde a reduzir/trocar a dimensão, tomando como critério de seleção um determinado valor de uma dimensão e projetando tal resultado como critério de seleção de valores de outras dimensões. Como exemplo ver consulta 4.

Como exemplo, vamos supor a existência do modelo representado pela figura 11. Suponha agora que desejamos conhecer a quantidade de cada um dos produtos vendidos em nosso sistema por mês. Para isso devemos pivotear as seguintes dimensões: Tabela de Fatos (Fatos), *Produto* e *Data*, onde teríamos o seguinte comando SQL:

Consulta 1 – Pivoteamento:

```
SELECT SUM(F.Quantidade) as quantidade_vendida, P.nome, D.mes
FROM Fatos AS F, Produto AS P, Data AS D
WHERE F.id_produto = P.id_produto
            AND F.id_data = D.id_data
GROUP BY P.nome, D.mes;
```

Desta maneira conseguiremos obter a resposta para o nosso problema.

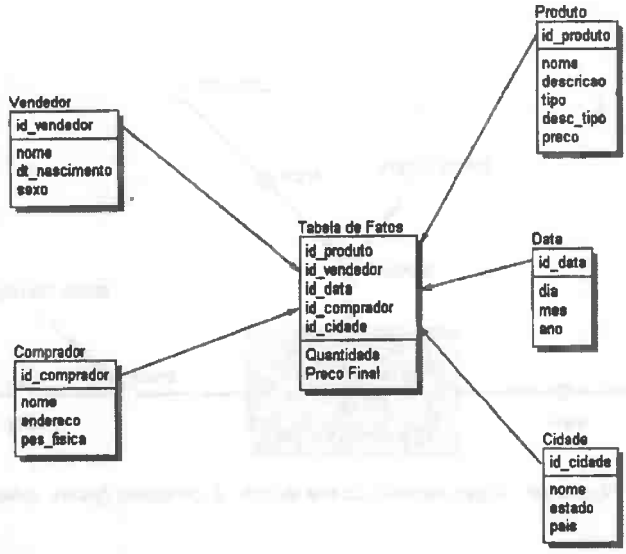

Figura 11 – Modelo para exemplos de consultas OLAP

Outras operações relacionadas com o pivoteamento são *rollup* e *drill-down*. Se fizermos o *rollup* sobre a consulta anterior estaremos aumentando o nível de agregação, ou seja, passaremos a analisar a venda de produtos por ano (próximo nível de agregação em relação ao mês). *Drill-down* é o processo inverso de *rollup*, ou seja, decresceríamos o nível de agregação, passando a analisar as vendas de produtos por dia e não mais por mês. Note que, para as operações de *rollup* e *drill-down* é necessário que haja uma hierarquia. Em nosso exemplo tais operações são permitidas para as dimensões de *Produto*, *Data* e *Cidade*. Os respectivos comandos SQL para *rollup* e *drill-down* seriam:

Consulta 2 – *Rollup:*

```
SELECT SUM(F.Quantidade) as quantidade_vendida, P.nome, D.ano
FROM Fatos AS F, Produto AS P, Data AS D
WHERE F.id_produto = P.id_produto
        AND F.id_data = D.id_data
GROUP BY P.nome, D.ano;
```

Consulta 3 – *Drill-down:*

```
SELECT SUM(F.Quantidade) as quantidade_vendida, P.nome, D.dia
FROM Fatos AS F, Produto AS P, Data AS D
WHERE F.id_produto = P.id_produto
        AND F.id_data = D.id_data
GROUP BY P.nome, D.dia;
```

Suponha que, após tal análise (consulta 1), desejássemos saber a quantidade vendida de determinado produto (tênis Bical, por exemplo), para cada cidade durante todo o período. Neste

caso, estaríamos realizando uma operação de *slice_and_dice*. Necessitaríamos, então, da tabela de Fatos e das dimensões *Região* e *Produto*. O comando SQL correspondente seria:

Consulta 4 – *Slice_and_dice:*

```
SELECT SUM(F.Quantidade) as quantidade_vendida, C.nome
FROM Fatos AS F, Cidade AS C, Produto AS P
WHERE F.id_produto = P.id_produto
        AND F.id_cidade = Cidade.id_cidade
        AND P.nome = "Tênis Bical"
GROUP BY C.nome;
```

Note que, a consulta 4 não se utiliza das dimensões *Data* e *Vendedor*, ao contrário da consulta 1. Porém, se utiliza da dimensão *Cidade* que não foi utilizada pela consulta 1. Vemos ainda que a consulta 4 seleciona um determinado valor para a dimensão de *Produto* ("Tênis Bical").

3.3.4 Do E/R aos esquemas de fatos

Nas últimas décadas, a maioria dos sistemas de informação foi documentado utilizando os esquemas E/R (Entidade/Relacionamento), portanto, parece natural que o modelo conceitual de um data warehouse tenha como base um esquema E/R existente. A metodologia para geração do DF, partindo de um esquema E/R, contempla os seguintes passos:

1. Definição dos fatos;
2. Para cada fato:
   a) Criação da árvore de atributos;
   b) Adequação da árvore de atributos ("podar" e "enxertar" a árvore);
   c) Definição das dimensões;
   d) Definição dos atributos fato;
   e) Definição das hierarquias.

Cada etapa apresentada acima será detalhada nas próximas seções, utilizando com exemplo um esquema E/R simplificado conforme mostrado na figura 12. Cada instância do relacionamento SALE representa um item que se referencia a um único produto em um *ticket* de venda. O atributo *unitPrice* é colocado em SALE em vez de em PRODUCT, já que o preço dos produtos pode variar com o tempo.

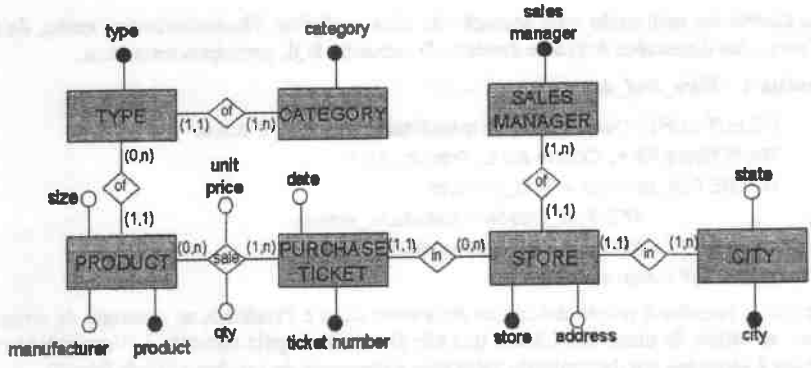

Figura 12 - Esquema E/R simplificado base para a geração do modelo DF SALE.

3.3.4.1 Definição dos Fatos

Os fatos são os conceitos de maior interesse no processo de tomada de decisões. Tipicamente, eles correspondem aos eventos que ocorrem dinamicamente no ambiente empresarial. Um fato pode ser representado no diagrama E/R ou por uma entidade **F** ou por um relacionamento n-ário entre as entidades **E1,..., En**. Se o fato for representado por um relacionamento, devemos transformar este relacionamento em uma entidade **F** substituindo cada ramo **Ei** por um relacionamento binário entre **F** e **Ei**. Cada relacionamento binário tem multiplicidade (1,1) no lado de **F** e (*mi, Mi*) no lado de **Ei**. Os atributos do relacionamento se tornam atributos de **F** e a chave de **F** é a combinação das chaves de **Ei**, $i = 1,...,n$.

As entidades ou relacionamentos que representam arquivos freqüentemente atualizados (como SALE) são bons candidatos para se tornarem fatos, sendo que os arquivos mais estáticos, como cadastros, não o são.

Cada fato identificado no diagrama E/R se torna a raiz de um esquema de fatos. No caso de SALE, o maior interesse da análise de negócios é a venda de um produto, que no diagrama E/R da figura 12, é representado pelo relacionamento SALE. Na figura 13, abaixo, representamos a transformação deste relacionamento em uma entidade.

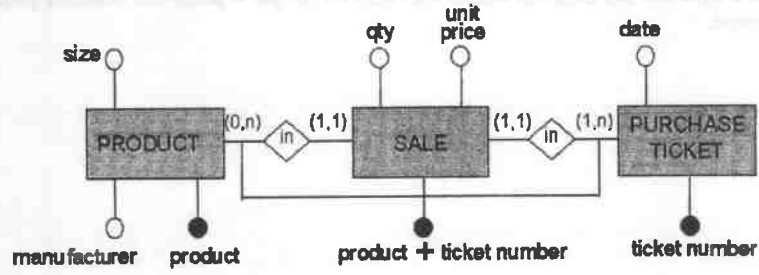

Figura 13 - Transformação do relacionamento SALE em uma entidade.

3.3.4.2 Criação da árvore de atributos

Uma vez definido o segmento do diagrama ER e a entidade F em que vamos trabalhar, podemos definir a árvore de atributos como sendo a árvore onde:

- Cada vértice corresponde a um atributo do esquema;
- A raiz corresponde à chave de F;
- Para cada vértice v, o atributo correspondente determina funcionalmente todos os atributos correspondentes dos descendentes de v.

Para facilitar a construção desta árvore, podemos aplicar o seguinte algoritmo recursivo sobre a entidade F, que representa o fato a ser analisado:

```
translate(F,identifier(F))
```

onde:

```
translate(E,v):
//E is the current entity, v is the current vertex
{ for each atributte a ∈ E | a ≠ identifier(F)
  do addchild(v,a);    //adds child a to vertex v
  for each entity G connected to E by a x-to-one
     relationship R
  do
  { for each attribute b ∈ R
    do addChild(v,b);
    addChild(v,identifier(G));
    translate(G,identifier(G));
  }
}
```

Algumas observações:

- Se F é identificado pela combinação de dois ou mais atributos, `identifier(F)` denota esta concatenação;
- Um relacionamento um-para-um pode ser encarado como um tipo particular de um relacionamento muitos-para-um e, por isso, pode ser inserido na árvore de atributos. Porém, em uma pesquisa de data warehouse, a operação de drill down em um relacionamento um-para-um significa acrescentar uma linha sem nenhum nível de detalhe adicional e este tipo de atributo pode ser representado como um atributo não-dimensional;
- Relacionamentos x-para-muitos não podem ser inseridos na árvore de atributos. De fato, a representação destes relacionamentos no nível lógico seria impossível de ser representado sem que a primeira forma normal fosse desrespeitada;

- Como já sabemos, um relacionamento n-ário é equivalente a *n* relacionamentos binários, sendo que a maioria dos relacionamentos n-ários tem multiplicidade máxima maior que 1 em todos os seus ramos. Este tipo de relacionamento determina, então, *n* relacionamentos binários um-para-muitos que não podem ser inseridos na árvore de atributos. Porém, um ramo com multiplicidade máxima igual a um determina um relacionamento binário um-para-um que pode ser inserido na árvore.

A árvore de atributos correpondente ao esquema E/R da figura 13 é mostrada abaixo:

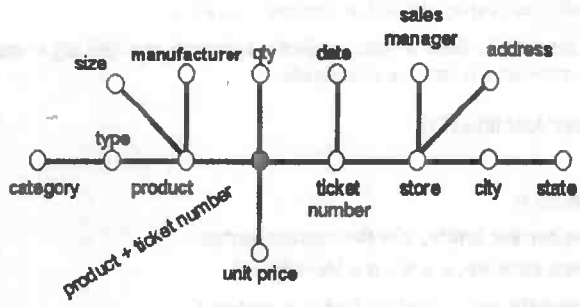

Figura 14 - Árvore de atributos para o exemplo SALE (a raiz está em cinza).

3.3.4.3 Adequação da árvore de atributos ("podar" e "enxertar" a árvore)

Como nem todos os atributos representados na árvore são de interesse para o data warehouse, é necessária a adequação da árvore com o objetivo de eliminar os níveis de detalhe desnecessários e, para isso, a árvore deve ser "podada" e "enxertada".

A operação *pruning* ("podar") resulta na eliminação alguma subárvore. Os atributos eliminados não serão incluídos no esquema de fatos e, por esta razão, não poderão ser utilizados para agregações de dados. Como exemplo, no caso de SALES a subárvore que inclui *city* e *state* poderia ser eliminada.

A operação *grafting* ("enxertar") é usada quando um determinado vértice da árvore expressa uma informação desnecessária, porém, seus descendentes devem ser preservados. Como exemplo, alguém pode querer classificar os produtos diretamente por categoria (*category*) sem considerar a informação do seu tipo (*type*).

Seja *v* o vértice a ser eliminado e *v'*, seu pai, a operação *grafting* pode ser implementada através do algoritmo abaixo:

```
graft(v):
{ for each v" | v" is child of v
    do addChild(v',v");
  drop v;
}
```

Como podemos verificar, a operação *grafting* move toda a subárvore enraizada em v para v'. Com isto, o atributo v não será incluído no esquema de fatos e seu correspondente nível de agregação será perdido, mantendo-se, porém, todos os níveis de seus descendentes. No caso de SALES, o detalhe dos tíquetes de compras não interessam, portanto, devemos aplicar a operação *grafting* no vértice *ticket number*, tendo como resultado a árvore representada na figura 15.

De forma geral podemos dizer que, se aplicarmos a operação *grafting* em um filho da raiz, teremos um nível mais alto de granularidade das instâncias do fato e que, se o nó que sofre a operação *grafting* tiver 2 ou mais filhos, isto nos levará a um aumento no número de dimensões no esquema de fatos. Além disso, se um vértice opcional sofre a mesma operação, todos os seus filhos herdam sua opcionalidade

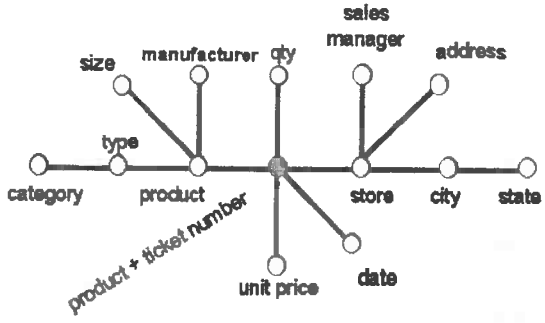

Figura 15 - Árvore de atributos para o exemplo SALES após a operação grafting ("enxertar") sobre o vértice ticket number.

3.3.4.4 Definição das dimensões

As dimensões vão determinar como as instâncias dos fatos podem ser agregadas de forma significativa para o processo de tomada de decisão. As dimensões serão escolhidas entre os vértices que são filhos da raiz (incluindo os vértices que se tornaram filhos da raiz após as operações de *grafting*). A escolha das dimensões é de grande importância já que isto determina a granularidade das instâncias dos fatos.

A dimensão tempo é um elemento chave em todo data warehouse, porém os diagramas E/R podem ser classificados como *snapshot* ou temporais, dependendo da forma com que tratam o tempo. Os esquemas *snapshot* descrevem o estado atual do domínio da aplicação enquanto que os esquemas temporais descrevem a evolução do domínio da aplicação durante um intervalo de tempo, ou seja, as versões antigas dos dados são explicitamente representadas e armazenadas. Nos esquemas temporais, o tempo está representado e, portanto é um candidato óbvio para uma dimensão. Se o atributo relacionado ao tempo aparecer na árvore de atributos como filho de algum vértice diferente da raiz, a árvore deve sofrer operações *grafting* para que o tempo se torne uma dimensão, ou seja, se torne um filho da raiz. No caso dos esquemas do tipo *snapshot*, o tempo não está explicitamente representado, já que é assumido que o esquema representa os

dados no momento atual. Ainda assim, para este tipo de esquema, o tempo deve ser adicionado como uma dimensão do esquema de fatos.

No exemplo SALES, que estamos analisando, os atributos escolhidos como dimensões são: *product*, *store* e intervalos de *date* que correpondem às semanas.

3.3.4.5 Definição dos atributos do fato

Normalmente, os atributos do fato são contadores do número de instâncias de F ou a soma/média/máximo/mínimo de expressões que evolvem atributos numéricos da árvore de atributos (são excluídos os atributos que já foram escolhidos como dimensões). Pode acontecer também que um fato não possua atributos e, neste caso, a informação é armazenada apenas para indicar a ocorrência do fato. Caso estes atributos existam, é o momento de se criar um glossário com as expressões que os descrevem. Por exemplo:

quantity sold = SUM(SALE.qty)

total returns = SUM(SALE.qty * SALE.unitPrice)

number of customers = COUNT(SALE)

Os operadores de agregação poderão ser aplicados a todas instâncias de SALES que sejam da mesma semana, loja e produto.

3.3.4.6 Definição das hierarquias

Este é o último passo na criação do esquema de fatos. Ao longo de cada hierarquia, os atributos deverão ser organizados em uma árvore, onde o relacionamento entre cada nó e seus descendentes é do tipo x-para-um. Neste estágio, ainda é possível realizar operações de *pruning* e *grafting* para eliminar detalhes irrelevantes, como por exemplo, um vértice conectado a seu pai através de um relacionamento do tipo um-para-um. Também é possível adicionar novos níveis de agregação dentro da hierarquia. Isto ocorre tipicamente na dimensão tempo. No exemplo SALES, foram introduzidos os níveis *month* e *week*, permitindo a agregação dos valores numéricos em meses e semanas. Durante esta etapa, os atributos que não serão utilizados para agregação, mas apenas para propósitos informativos devem ser identificados como atributos não dimensionais.

Ao término desta etapa, chegamos a uma definição gráfica do modelo conceitual do data warehouse, conforme apresentado na figura 8.

3.4 Os esquemas básicos e suas variações

Após a definição do modelo conceitual, podemos partir para o modelo lógico e sua conseqüente implementação física. A implementação física do modelo dimensional, em uma base de dados relacional, pode ser feita de várias formas diferentes. No entanto, para todos os esquemas, os fatos e as dimensões são implementados em tabelas físicas, onde cada linha é única e identificada por uma chave genérica, já discutida anteriormente. Nas próximas seções, estaremos analisando os esquemas básicos, conhecidos como *star schema* e *snowflake schema*, e suas variações.

3.4.1 O esquema *Star* clássico

Provavelmente, o *Star schema*, apresentado por Ralph Kimball [Kim96], seja o primeiro esquema utilizado para desenhar o data warehouse implementado em um banco de dados relacional. Neste esquema, a tabela fato contempla as seguintes características:

- Uma chave primária composta, sendo um elemento da chave para cada dimensão;
- Cada elemento chave para a dimensão deve ser representado e descrito na tabela dimensão correspondente, para efetuar o *join*;
- Opcionalmente, uma ou mais métricas ou fatos a serem avaliados;
- Pode armazenar as métricas sumariadas em diferentes níveis de agregação.

Uma tabela dimensão é definida para cada dimensão do negócio a ser analisada, contendo:

- Uma chave genérica, gerada pelo sistema;
- Uma coluna de descrição genérica para a dimensão ou colunas de descrição para cada atributo da hierarquia;
- Atributos que permitam efetuar os filtros;
- Um indicador *nível*, para estabelecer o nível da hierarquia a que se refere a linha da tabela;
- Valores nulos ocasionais, dependendo do nível hierárquico da linha.

O objetivo de se armazenar toda a hierarquia é possibilitar, durante as consultas, a utilização de filtros que permitam a agregação ou sumário dos fatos nos vários níveis hierárquicos.

a.

GEOGRAFIA_KEY	Geografia Descrição	Região	Estado	Cidade	Loja	Nível
1010	**Região Sul**	**1**				**4**
1020	**Região Sudeste**	**2**				**4**
2010	SP	2	10			3
2020	RJ	2	20			3
3010	**São Paulo**	**2**	**10**	**101**		**2**
3020	Campinas	2	10	102		2
4010	Loja ABC	2	10	101	1001	1
4020	Loja DEF	2	10	102	1002	1
4030	Loja GHI	2	10	102	1003	1

b.

GEOGRAFIA_KEY	Região	Região_desc	Estado	Estado_desc	Cidade	Cidade_desc	Loja		Nível
1010	**1**	**Região Sul**						--	**4**
1020	**2**	**Região Sudeste**						--	**4**
2010	2	Região Sudeste	10	São Paulo				--	3
2020	2	Região Sudeste	20	Rio de Janeiro				--	3
3010	**2**	**Região Sudeste**	**10**	**São Paulo**	**101**	**São Paulo**		--	**2**
3020	**2**	**Região Sudeste**	**10**	**São Paulo**	**102**	Campinas		--	**2**
4010	2	Região Sudeste	10	São Paulo	101	São Paulo	1001	--	1
4020	2	Região Sudeste	10	São Paulo	102	Campinas	1002	--	1
4030	2	Região Sudeste	10	São Paulo	102	Campinas	1003	--	1

Figura 16 – a. Tabela dimensão em detalhes com apenas uma descrição (genérica)

b. Tabela dimensão com uma descrição para cada nível da hierarquia – Star expandido.

As figuras 16a e 16b mostram os detalhes de uma tabela dimensão, chamada *Geografia* que descreve e define uma hierarquia entre a loja, a cidade onde se encontra, o estado (UF) e a região. Na figura 16a encontramos uma das possibilidades, com apenas uma descrição genérica para toda a dimensão, na figura 16b encontramos colunas de descrição específicas para cada elemento da hierarquia. É importante notar que, não existe uma preocupação com a normalização das tabelas no data warehouse, conforme já exposto no item 3.2.2 acima. Este é um dos aspectos que diferenciam bastante a modelagem do data warehouse da modelagem convencional. Outro aspecto a destacar é que o indicador *nível* não precisa ser numérico e pode conter o nome do nível da hierarquia, como por exemplo: "cidade", "loja", "mês" etc.

A dimensão *Tempo* é uma dimensão especial, sendo diferente de todas as outras dimensões. Esta dimensão requer alguns atributos comuns a todas as dimensões e três atributos especiais que serão de extrema importância no momento das consultas.

As colunas que devem compor a dimensão *Tempo* são:

- Chave única genérica, gerada pelo sistema, preferencialmente numérica e inteira, para ligação lógica com a tabela fato;
- Descrição do tempo contido naquela linha. Conforme os exemplos citados na figura 16, esta descrição pode ser única para a tabela ou específica para cada nível da hierarquia;
- Atributos da hierarquia, como nas outras dimensões;
- Nível ou Resolução, que indica o nível da hierarquia representado naquela linha, que pode ser numérico ou um pequeno texto;
- Seqüência na resolução, que indica a seqüência de um período de tempo dentro do nível ou resolução ao qual ele pertence. Este valor é um número inteiro consecutivo que designa a ordem cronológica para todas os dias, meses, semestres, anos etc, ou seja, para todos os períodos de tempo existentes na tabela dimensão. A seqüência é definida dentro de cada nível ou período de resolução, sem repetições e, [IA99] recomenda que sejam utilizados intervalos de valores para estabelecer a seqüência. Por exemplo, para a seqüência de *anos* a numeração seria de 1 a 10, para *semestres*, entre 100 e 200, todos os *meses* estariam entre 500 e 999 e todos os *dias* teriam seqüências numeradas a partir de 1.000. Na figura 17 abaixo, vemos que a seqüência na resolução para Janeiro de 1999 é 500, para Fevereiro de 1999 é 501, Março de 1999 tem seqüência 502 e assim por diante. É importante ressaltar novamente que o valor para esta coluna deve ser único dentro de um período de resolução;
- Indicador Corrente, se utilizada em conjunto com a coluna de seqüência na resolução, descrita no item anterior, indica qual o período de tempo dentro da resolução especificada, que deveria ser definido como *corrente*. Com isto, é possível definir qual será o tempo corrente para as consultas: a informação carregada mais recentemente, a data de hoje ou algum período de tempo futuro. A utilização desta coluna permite que determinadas condições de negócio estabeleçam o momento em que os dados estarão disponíveis para os usuários finais. Esta coluna é, normalmente, definida como um caracter e preenchida com 'S' ou 'N', onde 'S' indica que o dado já foi carregado no data warehouse. O período de tempo corrente, dentro da resolução (pode ser o dia corrente, o mês corrente etc.), será aquele que contiver na

coluna Indicador Corrente o valor de 'S' e o maior valor na coluna Seqüência na Resolução, descrita no item anterior.

Tempo key	Descrição	Resolução ou Nível	Sequência na Resolução	Indicador Corrente	Ano	Semestre	Mês	Dia
...	...	...	...	...	...	...	...	...
101	Janeiro 1999	Mês	501	S	1999	011999	011999	
102	Fevereiro 1999	Mês	502	S	1999	011999	021999	
103	Março 1999	Mês	503	S	1999	011999	031999	
104	Abril 1999	Mês	504	S	1999	011999	041999	
105	Maio 1999	Mês	505	S	1999	011999	051999	
106	Junho 1999	Mês	506	S	1999	011999	061999	
113	Janeiro 2000	Mês	513	S	2000	012000	012000	
114	Fevereiro 2000	Mês	514	S	2000	012000	022000	
115	Março 2000	Mês	515	S	2000	012000	032000	
116	Abril 2000	Mês	516	S	2000	012000	042000	
21	1 Sem. 1999	Semestre	101	S	1999	011999		
22	2 Sem. 1999	Semestre	102	S	1999	021999		
23	1 Sem. 2000	Semestre	103	S	2000	012000		
24	2 Sem. 2000	Semestre	104	S	2000	022000		
3	1999	Ano	3	S	1999			
4	2000	Ano	4	S	2000			
5	2001	Ano	5	N	2001			
6	2002	Ano	6	N	2002			
...	...	...	...	...	...	...	...	...

Figura 17 – Tabela representando a dimensão Tempo, com as colunas específicas para este tipo de dimensão.

Nem sempre é possível capturar todas as métricas e dimensões importantes para um modelo de negócios, em um único esquema *Star*. Ao contrário, a existência de múltiplas tabelas fato é a norma, definindo um modelo de múltiplas estrelas. Cada estrela é composta por uma tabela fato e suas dimensões associadas [AV98].

O principal fator que pode levar à separação das métricas em diferentes tabelas fato é o modelo de negócios analisado. Um processo de negócios pode envolver diferentes conjuntos de dimensões e as métricas relacionadas a cada processo podem estar sendo coletadas em diferentes intervalos de tempo.

O esquema *Star* é desenhado para simplicidade e velocidade nas consultas. Podemos citar como vantagens, na sua utilização, os seguintes aspectos:

- Facilidade de entendimento do modelo, principalmente por parte do usuário final;
- Excelente desempenho, devido ao uso de chaves genéricas, geralmente numéricas e inteiras;
- Facilidade na definição das hierarquias;
- Número de operações de *join* reduzidas, já que o modelo implementa todos os níveis da hierarquia das dimensões em apenas uma tabela por dimensão além de estar bastante desnormalizado;
- O metadados é simples.

Na figura 18 está representado um esquema *Star* clássico, composto por uma tabela fato e suas dimensões *Geografia*, *Produto* e *Tempo* associadas.

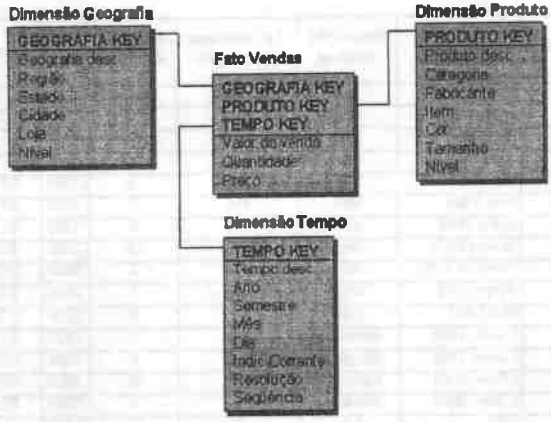

Figura 18 – Representação de um data warehouse em esquema Star clássico

Apesar de amplamente utilizado, o esquema *Star* apresenta alguns problemas. O primeiro deles está relacionado ao indicador *nível*, presente em todas as tabelas dimensão. Este indicador deve ser utilizado em todas as consultas, já que a mesma tabela dimensão armazena informações sobre todos os níveis da hierarquia. Durante a construção da consulta, o usuário deve conhecer a estrutura da base de dados, para poder especificar o nível da informação a ser pesquisada. Este indicador pode ser causa potencial para erros já que, se for esquecido ou mal utilizado, pode levar a resultados incorretos e baixa flexibilidade. Além disso, se a estrutura da dimensão for alterada, incluindo ou eliminando algum nível da hierarquia, o data warehouse necessitará de alterações físicas, elevando os custos de manutenção.

O segundo problema é que, por manter todos os níveis hierárquicos da dimensão fisicamente em uma única tabela, esta se torna muito grande, em vários casos. Outro fator que contribui para o aumento do tamanho da tabela é a desnormalização, que gera uma grande quantidade de valores de atributos repetidos. A desnormalização fica bastante clara, ao analisarmos, por exemplo, a dimensão *Geografia*, que aparece na figura 16b. No diagrama, representado na figura 19, podemos ver as dependências funcionais que ocorrem em uma tabela deste tipo.

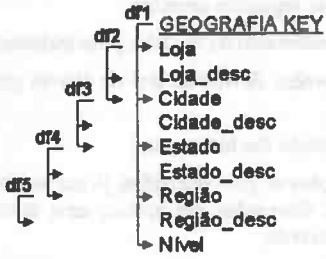

Figura 19 – Dependências funcionais da tabela dimensão GEOGRAFIA

O esquema *Star* clássico pode apresentar algumas variações, dependendo do problema de negócio apresentado. Nas seções seguintes descreveremos estas variantes do *Star*.

3.4.2 As variações do esquema *Star*

A primeira variação a ser discutida é o esquema *Partial Star*[Tan97]. Nesta variação existem múltiplas tabelas para cada dimensão e para a fato. Estas múltiplas tabelas estão lógica e fisicamente separadas em função dos seus níveis de agregação. Este modelo cria múltiplas estrelas, cada uma delas representando uma combinação de níveis de agregação em cada dimensão. Não existem ligações lógicas entre as várias tabelas fato ou dimensão, apenas entre a tabela dimensão e a tabela fato de cada grupo. Cada dimensão é representada por múltiplas tabelas que são fisicamente separadas pelo nível de agregação, sendo que as tabelas possuem como chave primária, uma chave genérica gerada pelo sistema, da mesma forma que o esquema *Star* clássico. Pode acontecer que uma métrica ocorra apenas para um determinado nível de uma dimensão. Por exemplo, a métrica *Preço* existe unicamente no nível *Item* da dimensão *Produto*. A figura abaixo retrata esta variação do *Star*, representando apenas a dimensão *Geografia*, separada nos níveis de *Região* e de *Cidade*, e as tabelas fatos associadas a estes níveis.

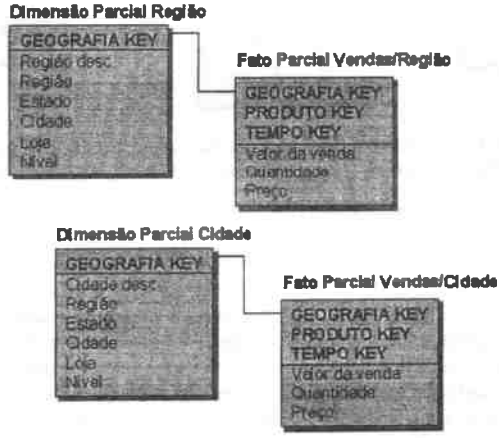

Figura 20 – Esquema Partial Star mostrando apenas a dimensão Geografia e as tabelas fato associadas.

Esta partição das informações, pelos níveis hierárquicos de agregação, leva à criação de duas estrelas: a primeira representando o nível de *Região* e a segunda representando o nível de *Cidade*. Ao adicionarmos a este esquema a dimensão *Tempo* e a dimensão *Produto* agregadas nos níveis de *Fabricante* e *Marca*, conforme o modelo representado na figura 18 (Star Clássico), expandiremos para *Região/Fabricante/Ano*, *Região/Marca/Ano*, *Cidade/Fabricante/Ano* e *Cidade/Marca/Ano*, sendo que cada combinação será representada por uma estrela parcial separada:

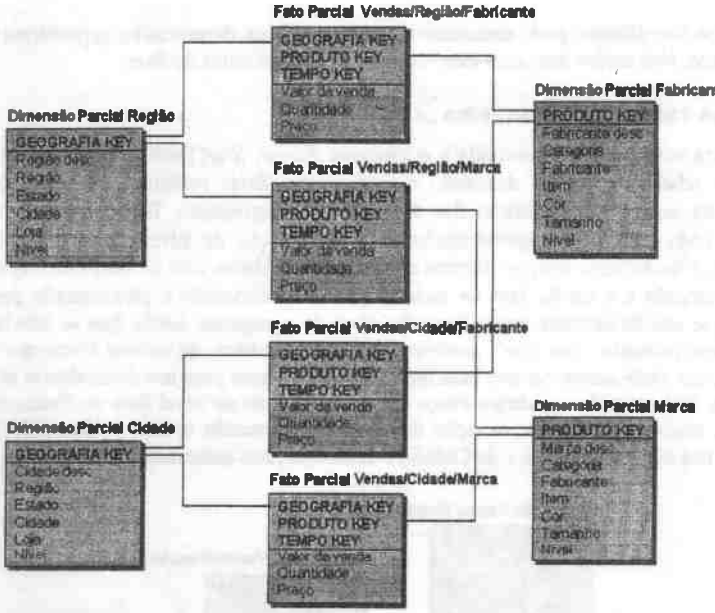

Figura 21 – Esquema Partial Star mostrando as dimensões Geografia e Produto separadas em diferentes níveis de agregação e as tabelas fato associadas.

São várias as vantagens encontradas no esquema *Partial Star*, apesar de sua visível complexidade. Uma delas é a possibilidade de maior controle do tempo de carga, *backup* e manutenção das tabelas, pois seu tamanho fica reduzido em função da partição. Estas partições possibilitam também a existência de fatos em certos níveis específicos e é possível eliminar as colunas que não tenham significado em determinados níveis, reduzindo assim a esparsidade. Como já dissemos, a complexidade do modelo aumenta em relação ao *Star* clássico e cada estrela requer uma definição específica no metadados, dificultando o processo de manutenção. Em relação às consultas, serão necessários múltiplos comandos SQL para analisar dados em mais de um nível de agregação em um mesmo relatório, sendo que esta condição pode degradar o desempenho da consulta. Além disso, a estrutura física de cada tabela ou grupo de tabelas requer alterações sempre que novos níveis de agregação ou novas combinações forem adicionados ou removidos.

Para simplificar um pouco o esquema *Partial Star*, utiliza-se uma combinação de seus princípios com o *Star* clássico. Pode-se particionar apenas a tabela fato e manter cada dimensão como uma única tabela ou o inverso, ou seja, manter a tabela fato única e particionar as dimensões. O particionamento é feito sempre baseado na agregação de determinados níveis da hierarquia. Estas variações são chamadas de *Fact Partitioning* e *Dimension* Partition, respectivamente e detalharemos cada uma delas a seguir. A variação *Fact Partitioning*, também chamada *Fact Constellation* é o esquema que particiona apenas a tabela fato. Este esquema utiliza uma única tabela para cada dimensão que fica ligada às múltiplas tabelas fato, particionadas pelos diferentes

níveis de agregação. Este esquema possibilita a existência de fatos em certos níveis, já que as tabelas são particionadas, assim como o *Partial Star*. Se particionarmos a informação em níveis de agregação desde o nível mais detalhado, o desempenho, no caso de consultas executadas nos níveis de detalhe mais altos, será melhor. Deve-se levar em conta que, na implementação deste esquema, se a hierarquia é extensa, numerosas tabelas fato serão criadas, aumentando a complexidade do desenho. Além disso, consultas que requeiram a análise das informações em mais de um nível de agregação emitirão múltiplos comandos SQL, o que pode acarretar em um desempenho inferior. A figura 22 abaixo representa um exemplo desta variação, onde a tabela fato está particionada em função dos níveis hierárquicos *Região* e *Cidade*, da dimensão *Geografia* (as outras dimensões não estão representadas):

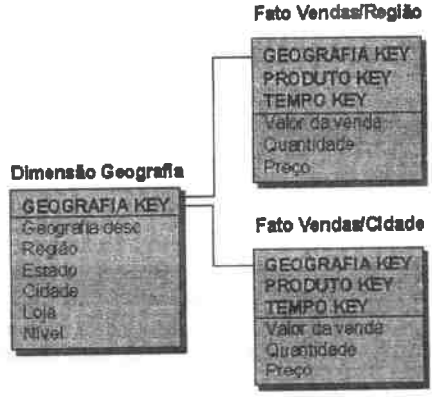

Figura 22 – Exemplo do esquema Fact Partitioning ou Fact Constellation para a dimensão Geografia e as tabelas fato correspondentes às partições por Região e Cidade.

A variação *Dimension Partitioning* mantém uma única tabela fato, que contém as métricas dos vários níveis de agregação, incluindo o nível mais detalhado. Esta tabela fato está logicamente ligada às múltiplas tabelas de dimensão, particionadas por diferentes níveis de agregação. A figura 23, abaixo, representa um esquema deste tipo, onde a dimensão *Geografia* encontra-se particionada por *Região* e por *Cidade* (as outras dimensões não estão representadas). As vantagens da *Dimension Partitioning* incluem a possibilidade de assinalar atributos que são específicos cada nível de agregação e um bom desempenho, já que apenas um comando SQL é emitido para a tabela fato, sem a necessidade de efetuar *joins*. Ao mesmo tempo, a vantagem de manter todos os fatos, detalhados e sumariados na mesma tabela, dificulta a recuperação das informações dos níveis mais altos.

A escolha do esquema a ser implementado, deve levar em conta vários fatores do negócio, bem como os tipos de consultas e o tamanho de cada uma das tabelas definidas no data warehouse. Seja qual for a variação escolhida, o *Star* se caratériza pela simplicidade, rapidez no acesso e desnormalização. O esquema *Star* pode ser refinado e transformado em um esquema chamado *Snowflake* que, para implementar as hierarquias dos atributos, se utiliza de tabelas de subdimensões. Estas subdimensões facilitam a normalização do modelo. Manter as tabelas desnormalizadas, implementando-se um esquema Star é um aspecto bastante discutível, pois a

divisão das tabelas, como em um esquema *Snowflake*, em nome da normalização, pode levar as consultas a um desempenho mais baixo.

Figura 23 – Implementação do esquema Dimension Partitioning, representando o particionamento da tabela dimensão Geografia.

O *Snowflake* pode ser considerado um *Star* normalizado, e nas próximas seções analisaremos este esquema e suas variações.

3.4.3 O esquema *Snowflake*

O esquema *Snowflake* emprega uma combinação de normalização da base de dados, para manter a integridade e reduzir os dados armazenados de forma redundante, com uma desnormalização para obter melhor desempenho [Tan97]. Neste esquema as dimensões são normalizadas em subdimensões, sendo que cada nível da hierarquia fica em uma subdimensão. Por esta razão, não há necessidade de se utilizar o indicador de nível que existe nos esquemas do tipo *Star*. A tabela principal da dimensão tem uma chave para cada nível hierárquico representado na subdimensão e não mais uma única chave, como no *Star*.

O *Snowflake* apresenta duas variações básicas que diferem na disposição das tabelas que representam as subdimensões, os *Snowflake Lookup* e o *Snowflake Chain*, que serão descritos na próxima seção. Sua representação gráfica fica similar a um floco de neve, devido ao particionamento das tabelas dimensão.

3.4.4 As variações do esquema *Snowflake*

O esquema *Snowflake Lookup* emprega tabelas adicionais para nomes e descrições dos atributos, todas ligadas a uma tabela principal da dimensão. Desta forma é possível reduzir o tamanho da tabela dimensão, eliminando a redundância do armazenamento das mesmas descrições em várias linhas diferentes, sendo que as tabelas adicionais atuam como tabelas de *lookup* para a chave ou valores codificados da tabela principal da dimensão que, por sua vez, está logicamente ligada a uma única tabela de fatos. A figura abaixo mostra o mesmo exemplo citado na figura 18, porém modelado em *Snowflake Lookup*. Nesta figura está representada apenas a dimensão *Gegrafia* e suas subdimensões *Região*, *Estado*, *Cidade* e *Loja*.

Na figura 24, pode-se notar que a ligação entre a tabela fato e a tabela da dimensão principal é feita da mesma forma que no esquema *Star*, através de uma chave gerada genérica. A tabela principal da dimensão se conecta logicamente às subdimensões, que são as tabelas de *lookup*, através de suas chaves primárias.

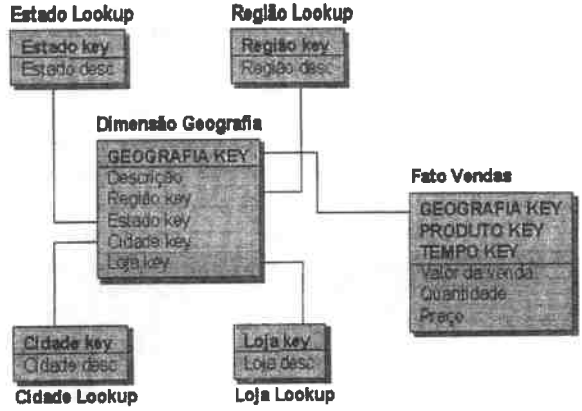

Figura 24 – Desenho lógico da dimensão Geografia em um data warehouse implementando o esquema Snowflake Lookup

As descrições não precisam ser repetidas como no esquema *Star*, simplificando o armazenamento, reduzindo o tamanho relativo das tabelas dimensão e melhorando o controle de integridade dos dados. O uso das chaves geradas genéricas, geralmente números inteiros, levam a um melhor desempenho, menor manutenção do metadados e mais flexibilidade ao data warehouse. Vale ressaltar que o desempenho pode ficar afetado se múltiplas consultas ou múltiplos *joins* têm que ser emitidos para decodificar as chaves, ao buscar as descrições nas tabelas adicionais. Além disso, a manutenção da base de dados requer manutenção adicional dado o número maior de tabelas físicas distintas.

O esquema *Snowflake* apresenta um nível muito maior de normalização, se comparado aos esquemas do tipo *Star*. A figura 25 mostra os diagramas que representam as dependências funcionais do esquema que aparece na figura 24 em *Snowflake Lookup*.

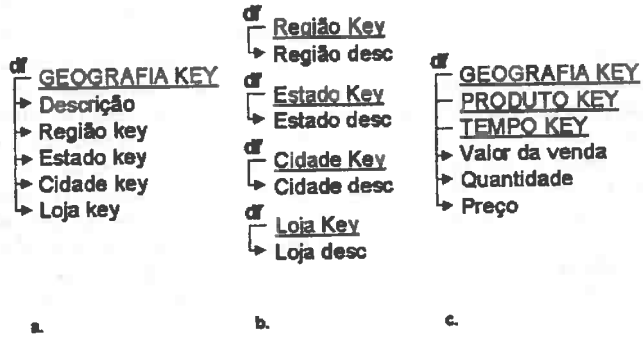

Figura 25 – Diagramas representando as dependências funcionais em:
a. Tabela principal da dimensão Geografia;
b. Tabelas de Lookup (subdimensões) e
c. Tabela Fato Vendas

A segunda variação do esquema *Snowflake*, conhecida por *Snowflake Chain*, utiliza também subdimensões, particionadas pelos níveis hierárquicos da dimensão, sendo que a tabela principal da dimensão representa o nível mais baixo (mais detalhado) da hierarquia. As subdimensões estão encadeadas, partindo-se da tabela fato que está logicamente conectada à tabela da subdimensão principal ou raiz. Na figura 26 encontramos um exemplo desta implementação, representando a dimensão *Geografia* com suas subdimensões e a tabela fato de vendas.

Figura 26 – Exemplo de implementação do esquema Snowflake Chain, representando a ligação da tabela Fato com a dimensão Geografia (dividida em subdimensões encadeadas).

Cada tabela da subdimensão contém sua chave primária e suas descrições associadas. Além disso, contém também a chave para o próximo nível da hierarquia da dimensão e assim por diante, até chegarmos à última subdimensão, que contém o nível mais alto (menos detalhado) da hierarquia. É tipicamente utilizada quando os fatos contêm informações apenas sobre o nível de detalhe mais baixo da hierarquia. Como podemos verificar na figura 26, a tabela fato já não utiliza uma chave para a dimensão *Geografia* como um todo, a chave utilizada é diretamente para a subdimensão principal. Fica claro que esta implementação não é recomendada quando os relatórios necessitam freqüentemente vários níveis de agregação da informação, já que são necessários vários passos na cadeia para se chegar ao resultado. Este esquema oferece um alto grau de integridade de dados, pois os nomes e as descrições são mantidos em um único local, reduzindo o tamanho das tabelas de dimensão. A maior desvantagem está no baixo desempenho, uma vez que todos os níveis da cadeia devem ser acessados, mesmo quando se requer apenas os níveis mais altos de agregação. Em situações práticas, existe uma alternativa para minimizar este efeito, adicionando-se, a cada subdimensão, chaves para todos os níveis superiores da hierarquia. Com isso, pode-se "saltar" determinados níveis, utilizando-se a chave que leva diretamente para o nível requerido. A figura 27 abaixo apresenta esta alternativa, que visa basicamente um melhor desempenho do *Snowflake Chain*.

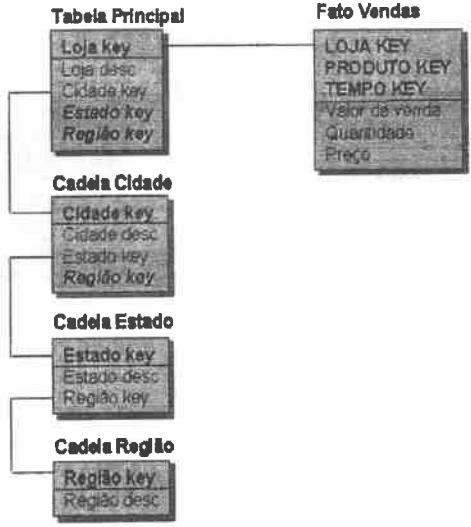

Figura 27 – Exemplo de Snowflake Chain com chaves adicionais para melhorar o desempenho das consultas.

Podem ocorrer, em determinados modelos, vários atributos diferentes que não estão associados a nenhuma dimensão específica. Em vez de criarmos várias dimensões com apenas um atributo, é possível agrupá-los em apenas uma dimensão. Este esquema é chamado de *Snowflake Attribute* [Tan97], sendo que a dimensão artificial pode também implementar a combinação de dimensões pouco utilizadas com o objetivo de reduzir o número de dimensões do data warehouse, simplificar o modelo e melhorar o desempenho. A tabela principal desta dimensão artificial contém as chaves estrangeiras dos diferentes atributos ou das tabelas dimensão a serem agrupadas. A chave primária da dimensão artificial é uma chave genérica gerada pelo sistema e uma única linha é adicionada para cada combinação válida de todos os atributos ou dimensões envolvidas. Cada chave estrangeira da tabela principal leva a uma chave primária na tabela de descrição daquele atributo ou dimensão. Na figura 28, encontramos um exemplo, onde *Tamanho*, *Cor*, *Formato* e *Perfume* são atributos não dimensionais dos produtos vendidos, que não podem ser associados a nenhuma das dimensões existentes no modelo. Utilizando-se este tipo de esquema, os atributos podem ser agrupados em uma única dimensão artificial que contém todas as combinações válidas destes atributos, sendo que a tabela fato contém uma chave para conectar-se logicamente a esta dimensão, da mesma forma que se conecta às outras dimensões do modelo.

As vantagens encontradas neste tipo de implementação são várias: redução do número de dimensões, que leva à redução do tamanho global dos índices e da complexidade do SQL gerado, maior integridade dos dados e os vários atributos juntos ou dimensões pouco usadas compartilham de um único ponto de entrada na tabela de fatos.

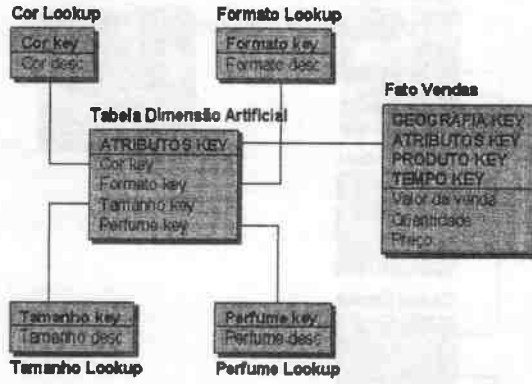

Figura 28 – Exemplo do esquema Snowflake Attribute agrupando os atributos Cor, Formato, Tamanho e Perfume, características dos produtos vendidos.

Sem este tipo de esquema, cada um destes atributos ou dimensões seria mantido como uma chave estrangeira na tabela fato, resultando em um ponto de entrada que não faria parte da chave primária e, provavelmente, sem índice físico para facilitar o acesso. Esta situação, retratada na figura 29, deve ser evitada.

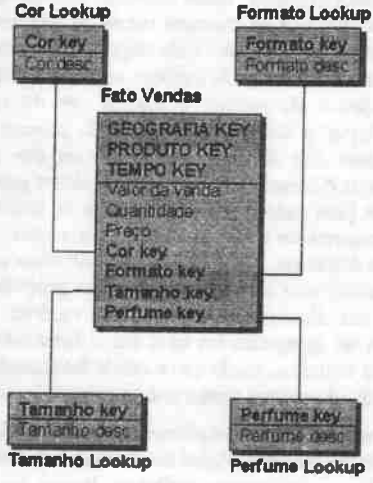

Figura 29 – Implementação alternativa sem a utilização do esquema Snowflake Attribute.

3.5 Agregações das informações

Apesar dos dados no data warehouse serem armazenados segundo a granularidade definida, muitas das consultas realizadas necessitam, além das informações detalhadas, de informações

sumariadas ao longo das dimensões. A informação armazenada no nível de detalhe é importante, porém o acesso à informação em níveis sumariados permite aos analistas de negócio terem uma visão global do modelo de negócios analisado. Estas consultas, partindo de uma base onde existem apenas os dados de nível básico, ou seja, do nível mais detalhado, se tiverem que sumariar os dados no momento da execução, sobrecarregarão todo o processo de análise. De acordo com [Kim96], é necessário um determinado conjunto de vários agregados precomputados para acelerar cada uma das consultas, sendo que o efeito sobre o desempenho é considerável, chegando a obter reduções de dez a mil vezes no tempo de processamento. Por esta razão, a utilização de dados previamente sumariados é um recurso bastante eficiente para controlar o desempenho do data warehouse. Nesta seção, estaremos analisando aspectos da definição, implementação e utilização destes valores agregados.

3.5.1 Definindo os agregados

A escolha dos agregados a serem previamente armazenados no data warehouse, já que armazenar todos os agregados é impraticável, não é uma tarefa fácil. É necessária uma análise das consultas que serão executadas pelos usuários, identificando os agrupamentos mais utilizados, além de constante acompanhamento após sua implantação.

Uma forma de se descrever as agregações a serem aplicadas na base é como um "agregado *n*-direcional", onde *n* indica o número de dimensões sumariadas como resultado da agregação. Um exemplo de um agregado *uni*-direcional seria sumariar uma dimensão (em algum nível da hierarquia) enquanto que as outras dimensões ficam em seu nível atômico, mais detalhado. Um agregado *bi*-direcional sumaria um nível da hierarquia para duas dimensões deixando as outras dimensões em seu nível atômico. Se utilizarmos o exemplo citado na figura 18, que representa um esquema *Star* clássico, com três dimensões *Geografia*, *Produto* e *Tempo* e, se quisermos implementá-las em nosso data warehouse, informações sumariadas para totais de *Categoria* na dimensão *Produto*, totais de *Estado* na dimensão *Geografia* e totais mensais na dimensão *Tempo*, teremos sete tipos de agregados diferentes:

1. Agregado unidirecional: totais de *categoria* por *loja* por *dia*;
2. Agregado unidirecional: totais de *cidade* por *item de produto* por *dia*;
3. Agregado unidirecional: totais mensais por *item de produto* por *loja*;
4. Agregado bidirecional: totais de *categoria* por totais de *cidade* por *dia*;
5. Agregado bidirecional: totais de *categoria* por totais mensais por *loja*;
6. Agregado bidirecional: totais de *cidades* por totais mensais por *item de produto*;
7. Agregado tridirecional: totais de *categoria* por totais de *cidade* por totais mensais.

Neste exemplo fica claro que não estamos representando todos os possíveis agregados para as dimensões do modelo. Como já dissemos, isto seria impraticável, dada a grande quantidade de dados a ser administrada. Ainda assim, para os agregados escolhidos é necessária uma avaliação do tamanho do resultado a ser gerado, levando-se em conta a explosão do número de linhas e a dispersão das informações armazenadas no data warehouse. Uma tabela de fatos de nível básico, para o exemplo que estamos analisando, é normalmente bastante dispersa no preenchimento das chaves [Tan96]. Para entender este aspecto, temos que lembrar que apenas uma parte dos produtos são vendidos por dia em cada uma das lojas. Porém, ao calcularmos o tamanho do

conjunto de agregados, devemos levar em conta que sua criação aumenta drasticamente a taxa de ocupação, aumentando bastante o tamanho do data warehouse. Para comprovarmos este fato, vamos analisar um exemplo prático em um data warehouse fictício. Vamos supor que no modelo exemplo que estamos analisando, existem 10.000 itens produtos diferentes, 1.000 lojas e o período de tempo armazenado é de 2 anos, equivalente a 730 dias. Além disso, neste data warehouse exemplo, apenas aproximadamente 10 porcento dos produtos são vendidos em uma determinada loja em um determinado dia. Isto significa que a tabela de fatos ocupa apenas 10 porcento do que ocuparia caso todos os produtos fossem vendidos em todas as lojas, todos os dias. Entretanto, com a criação dos agregados esta taxa sobe dramaticamente. Vamos supor também que no caso de produtos, 10.000 itens levam a 2.000 categorias de produto, as 1.000 lojas serão agrupadas em 100 cidades e o período de tempo total será sumariado em 24 períodos mensais. Além disso, vamos considerar a dispersão e o número de linhas a tabela de fatos, de acordo com a tabela abaixo:

Tabela	Produto	Geografia	Tempo	Dispersão	# Linhas
Base: item de produto por loja por dia	10.000	1.000	730	10%	730 milhões
Unidir.: categoria por loja por dia	2.000	1.000	730	50%	730 milhões
Unidir.: item por cidade por dia	10.000	100	730	50%	365 milhões
Unidir.: item por loja por mês	10.000	1.000	24	50%	120 milhões
Bidir.: categoria por cidade por dia	2.000	100	730	80%	116 milhões
Bidir.: categoria por loja por mês	2.000	1.000	24	80%	38 milhões
Bidir.: item por cidade por mês	10.000	100	24	80%	19 milhões
Tridir.: categoria por cidade por mês	2.000	100	24	100%	4,8 milhões
				Total aproximado:	2.122 milhões

Se considerarmos o tamanho inicial da tabela de fatos base, que é de aproximadamente 730 milhões de linhas, e o número de linhas após a criação destes sete níveis de agregados, verificamos que o aumento foi de 200%. O fator de dispersão pode ser difícil de prever e a escolha dos agregados deve ser feita com bastante cuidado. Ainda assim, de acordo com [Tan96], a forma mais eficiente para controlar a explosão de agregados, mas ainda assim beneficiar-se de seu valor, é garantir que, em média, cada agregado resuma no mínimo 20 e, de preferência, 20 ou mais itens de nível básico.

É importante lembrar que um determinado agregado pode servir para acelerar uma consulta que requeira um agregado de nível superior, ou seja, se existe um agregado de nível intermediário na hierarquia, este pode ser utilizado, não sendo necessário executar a consulta agregando o nível mais detalhado.

3.5.2 Implementando os agregados

A implementação dos valores agregados no data warehouse conta com várias opções de desenho. É possível manter os fatos sumariados na mesma tabela fato original ou separados por tipo de agregado, o mesmo ocorre com as dimensões. A forma de se implementar os agregados vai depender do esquema implementado para o data warehouse e do número de linhas geradas pelos agregados.

Para um esquema do tipo *Star*, onde para cada categoria de informações é gerada uma estrela composta por uma tabela fato e uma tabela para cada dimensão, os agregados podem residir na tabela de fatos original e seus identificadores já estão presentes nas tabelas de dimensão. Cada linha da tabela fato é identificada por uma chave composta, como já visto anteriormente, que

identifica, nas dimensões, os elementos associados aos valores armazenados na fato. Antes da criação dos agregados, a tabela fato contém apenas valores associados ao nível mais detalhado de cada dimensão. Assim, com a inclusão dos valores agregados na tabela fato, estes deverão relacionar-se com os outros níveis da hierarquia das dimensões. Isto é possível, já que cada tabela dimensão, no esquema *Star*, possui uma linha descrevendo cada elemento da hierarquia, sendo que o campo *Nível* identifica esta estrutura hierárquica. É claro que, se os agregados são representados nas tabelas de dimensão e fatos originais, por meio desta estrutura de campos *Nível*, todas as consultas feitas a esse esquema devem restringir o campo *Nível* a um único valor, caso contrário, ocorrerá contagem dupla [Kim96], envolvendo valores relacionados a níveis de agregação diferentes. A figura 30 mostra este relacionamento, representando uma dimensão *Geografia* e uma tabela fato que possui valores para os vários níveis da dimensão.

Tabela Fato:

	Tempo key	Produto key	Geografia key	Valor	Quantidade
(1)	980715	101	4030	348,00	140
(2)	980716	101	4030	287,00	114
(3)	980716	101	4010	443,00	170
(4)	980716	102	4010	690,00	158
(5)	980717	101	2010	5.800,00	2320
(6)	980718	102	2010	7.540,00	1720
(7)	980718	101	2010	6.860,00	2740
(8)	980716	101	3010	2.840,00	1184
(9)	980716	103	3010	5.834,00	4059
	...	...	...	...	...

Tabela Dimensão Geografia:

Geografia key	Geografia Descrição	Região	Estado	Cidade	Loja	Nível
1010	Região Sul	1				Região
1020	Região Sudeste	2				Região
2010	SP	2	10			Estado
2020	RJ	2	20			Estado
3010	São Paulo	2	10	101		Cidade
3020	Campinas	2	10	102		Cidade
4010	Loja ABC	2	10	101	1001	Loja
4020	Loja DEF	2	10	102	1002	Loja
4030	Loja GHI	2	10	102	1003	Loja
...	...	...	...	...	...	...

Figura 30 – Representação de uma única tabela fato contendo informações de nível detalhado e de agregados associados à dimensão Geografia.

Como podemos verificar, na figura 30, a tabela fato continua sendo única, contendo as linhas de fatos para os níveis detalhados e agregados da dimensão. No exemplo, foram representados parcialmente dois agregados unidirecionais, sendo:

- Agregado: *Item de produto* por *dia* por *Estado* e
- Agregado: *Item de protudo* por *dia* por *Cidade*.

Portanto, na tabela fato acima, as linhas (1), (2), (3) e (4) representam valores associados aos níveis detalhados *Item de produto*, *Dia* e *Loja*, enquanto que as linhas (5), (6), (7), (8) e (9) representam os agregados.

A grande desvantagem desta implementação é a utilização do campo *Nível* que, como citado anteriormente, pode acarretar contagem dupla, se não for utilizado corretamente.

Uma alternativa seria manter os fatos agregados em tabelas distintas, ou seja, uma tabela de fatos para cada agregado. [Kim96] cita algumas das vantagens desta alternativa de implementação:

- Não ocorrerá contagem dupla em qualquer aplicação que utilize as tabelas;
- Os vários tipos de agregados podem ser criados, eliminados, carregados e indexados separadamente quando estão em tabelas diferentes. Como eles provavelmente serão introduzidos no banco de dados em momentos diferentes, o uso de tabelas separadas permite um gerenciamento incremental.

Em um esquema do tipo *Snowflake*, a implementação mais adequada é, obviamente, a utilização de tabelas fato separadas por agregado, já que as dimensões são também separadas. Não há motivos para implementarmos o atributo *Nível* em um esquema do tipo *Snowflake*, unicamente para mantermos os agregados em apenas uma tabela fato.

3.5.3 Utilizando os agregados com um novo componente: o navegador de agregados

A melhor alternativa para a utilização dos agregados é, sem dúvida, o pré-armazenamento dos agregados mais utilizados, sendo que algumas vezes a agregação, por não existir um agregado compatível, é efetuada pela consulta durante sua execução.

Fica claro que o controle deste procedimento não deve ficar a cargo do usuário final, pois a criação ou eliminação de algum agregado levaria a uma série de procedimentos administrativos impraticáveis. Por esta razão, em um data warehouse projetado corretamente, os usuários finais e desenvolvedores de aplicações jamais verão quaisquer dessas tabelas de agregados [Kim96]. Torna-se necessária, portanto, a definição de um componente de software que escolha a tabela de agregados mais apropriada durante uma consulta, baseado, normalmente, em informações existentes no metadados do data warehouse. Assim, este componente chamado por [Kim96] de navegador e por [IA99], de *Aggregate Aware*, para citar apenas dois exemplos, é capaz de criar as combinações de agregação, que não estejam presentes no data warehouse, de forma dinâmica durante a execução da consulta. É importante que este componente seja capaz de criar estas agregações dinâmicas partindo do agregado predefinido mais adequado para a consulta e otimizando ao máximo o desempenho destas agregações.

A utilização deste navegador não elimina as preocupações relacionadas aos agregados. Como estas agregações dinâmicas são mais lentas que o acesso a um agregado armazenado, recomenda-se uma análise profunda para a definição destes agregados, seguida de acompanhamento ininterrupto de sua utilização.

3.6 O processo de carga na implementação estática

Uma das leis imutáveis do data warehousing, conforme [AV98], é sobre a complexidade do processo de carga. O autor nos diz que a parte mais complexa na construção de um data warehouse é a carga dos dados. Vários aspectos contribuem para aumentar a complexidade desta tarefa: baixa qualidade e falta de dados adequados, péssima documentação dos sistemas, múltiplas, redundantes e conflitantes fontes de dados. Muitas características complexas dos dados

ficam "escondidas", vindo à tona somente no momento do processo de carga do data warehouse. A utilização de ferramentas que automatizam este processo pode facilitar o trabalho, porém a complexidade semântica que envolve o processo de transformação dos dados não pode ser minimizada. O processo de carga é, na grande maioria das vezes, redundante e desnecessariamente complicado, tornando-se um ponto crítico para a introdução de erros.

A complexidade desta etapa surge devido à diversidade e abrangência das tarefas envolvidas, recuperando, convertendo e migrando os dados a partir das diversas fontes, executando sua transformação de modo a garantir que a base de dados resultante compreenda sempre informações precisas, integradas, válidas e disponíveis em tempo hábil.

O processo de extração conta com várias dificuldades operacionais como, por exemplo, encontrar uma janela de tempo adequada para a execução do conjunto de programas necessários para o processo, normalmente em *batch*, principalmente se a atualização do data warehouse for diária. Outro problema a ser enfrentado refere-se ao ponto de sincronismo entre os vários sistemas transacionais, que são atualizados em diferentes intervalos de tempo e que servirão de entrada para este processo de extração.

Uma vez que a periodicidade do processo de carga esteja definida, é necessário escolher um método para **identificar e capturar as informações que foram atualizadas** nas bases de origem desde a execução da última carga. Existem algumas opções que podem ser empregadas para esta tarefa e, de acordo com [Kel94], as mais importantes são:

- Utilização de *Logs*: a maioria dos gerenciadores de bancos de dados mantém *logs* ou *journals* que podem ser utilizados para identificar as alterações ocorridas nas bases de dados. Estas alterações, uma vez identificadas, devem ser escritas em um arquivo diferente que será utilizado para migrar as alterações para o ambiente de data warehouse. Esta alternativa produz bons resultados se as atualizações a serem capturadas são obtidas de forma simples e direta. Porém, se os dados necessários para a atualização do data warehouse estiverem em um número considerável de arquivos ou bases de dados separados, dificultando o sincronismo das *logs*, este processo começa a aumentar em complexidade. Adicionalmente, podemos citar os problemas relativos às alterações necessárias no programas de carga devido a novas versões dos gerenciadores de bancos de dados que podem levar a modificações na estrutura das *logs*.
- Comparação das informações: apesar de conceitualmente simples, é uma opção bastante trabalhosa para executar. Esta opção consiste em desenvolver um programa que leia os dados nas bases fonte e no data warehouse, identificando as alterações ocorridas desde a última execução da carga. Este método é caro, no que se refere aos recursos de máquina necessários para comparar os dados e pode ser demorado dependendo do tamanho das bases pesquisadas e dos recursos disponíveis para seu processamento.
- Reengenharia dos sistemas aplicativos transacionais: esta opção não se mostra muito atrativa, já que requer que as aplicações em produção sejam alteradas com o objetivo de armazenar, em arquivos destinados para este fim, as modificações ocorridas nos dados fonte. Ocorre, porém, que a maioria das aplicações não pode sofrer facilmente um processo de reengenharia sendo que esta opção, em muitas situações, só poderia

ser aplicada nas novas aplicações que estivessem sendo desenvolvidas. Neste caso, as necessidades do data warehouse passariam a fazer parte da especificação da aplicação.

Após sua captura, os dados devem passar por uma etapa de **transformação**, já que, simplesmente obter os dados do ambiente fonte e passá-los para o data warehouse não é suficiente. É necessário, com o objetivo de otimizar o potencial do data warehouse, transformar as entidades obtidas em novas composições de entidades que são requeridas para processo de geração de informação relevante para o usuário.

Além da transformação, os dados passam por um processo de **enriquecimento**, que é, normalmente, um produto da integração de dados e ocorre quando um atributo adicional é acrescentado a uma entidade. Se um dado externo está sendo acrescentado ao data warehouse, uma entidade *Cliente*, por exemplo, pode ser "enriquecida" com este novo atributo que foi selecionado de fontes econômicas, demográficas, financeiras, etc. Estas fontes podem ser valiosas para o "enriquecimento" das informações, porém mais valiosas ainda, podem ser as próprias aplicações do data warehouse cujos atributos podem ser derivados dos padrões de comportamento analisados.

Uma outra etapa necessária para a migração dos dados envolve um **mecanismo de transporte** partindo do ambiente operacional para o data warehouse. Estes dois ambientes podem estar localizados em distintas plataformas, além da possibilidade de estarem fisicamente remotas levando a uma maior dificuldade na transferência dos dados. Por ser inconcebível um meio de transporte de dados que não seja eletrônico, a implementação do mecanismo de transporte pode envolver uma avaliação de todas as opções disponíveis, já que diferentes fabricantes podem oferecer diferentes opções proprietárias que custos, capacidades e limitações bastante distintas. Ao considerar a velocidade da transferência, deve-se levar em conta cada aspecto envolvido no processo, seja na plataforma de origem ou na plataforma destino. Esta análise é particularmente importante para determinar, no contexto considerado, o tempo disponível para a execução desta etapa nas plataformas envolvidas.

Na seqüência do processo de carga um fator muito importante está relacionado a garantir a **integridade dos dados** que estão sendo adicionados ao data warehouse. Este controle de integridade deve ser implementado de forma a garantir dois aspectos. O primeiro deles é a garantia de que os dados que foram extraídos dos sistemas de origem são exatamente os mesmos que estão sendo carregados no data warehouse. O segundo aspecto deve garantir que os dados que estão sendo carregados estão consistentes com os dados que foram pedidos para as bases de origem em um determinado instante no tempo.

A **reformatação dos dados** pode também ser uma etapa necessária neste processo, uma vez que os ambientes de origem e o data warehouse podem estar em ambientes computacionais heterogêneos, que muitas vezes causam problemas relacionados ao formato dos dados. O problema mais comum refere-se às diferenças entre os formatos de arquivos em EBCDIC, encontrados nos ambientes IBM e os formatos ASCII utilizados pela maioria das outras plataformas.

Como último passo do processo, encontramos a etapa de **carga dos dados** que depende, obviamente, do volume de dados envolvido no processo. Pode ser que o tempo necessário para a carga dos dados adicionado ao tempo requerido para transferi-los, cause um impacto negativo na disponibilidade do data warehouse para os usuários. Portanto, devem ser considerados todos os aspectos que possam dificultar este processo. Um destes aspectos é a quantidade de índices

criados nas tabelas do data warehouse. Uma grande quantidade de índices adequados vai agilizar a execução das consultas, objetivo primário de um data warehouse, porém este alto nível de indexação pode diminuir sensivelmente a velocidade do processo de carga.

É fácil perceber, após uma análise das etapas necessárias ao processo de carga, a alta complexidade desta tarefa que, se não for definida com bastante cuidado, pode levar à introdução de erros, comprometendo todo o processo de tomada de decisão executado pelos usuários. Além disso, qualquer alteração na arquitetura dos sistemas transacionais e nas bases de origem deve ser detalhadamente verificada, já que pode causar alterações em várias etapas do processo de carga.

3.7 Conclusão

Com a implementação desta abordagem estática, o sincronismo entre o data warehouse e os sistemas transacionais fica dependente da periodicidade do processo de carga. Normalmente, a periodicidade da sincronização implementada é, no mínimo, diária. Para um grande conjunto de aplicações em data warehouse, esta periodicidade é satisfatória. Porém, para determinadas aplicações, que requerem a propagação dos dados da base transacional para o data warehouse de forma imediata, é necessária uma forma dinâmica para implementar este sincronismo. Outro aspecto a considerar é que, durante o processo de carga descrito na seção anterior, o data warehouse fica indisponível para consultas. Se considerarmos as características de uma economia globalizada, onde empresas localizadas em diferentes países possam estar utilizando o mesmo data warehouse para os processos de tomada de decisão, este período de indisponibilidade é totalmente inaceitável.

Face aos aspectos expostos, é necessário buscar uma implementação mais dinâmica do data warehouse, que minimize os problemas apresentados acima. Esta implementação será descrita no próximo capítulo deste trabalho.

4 A abordagem dinâmica utilizando visões materializadas

Partindo das definições já citadas anteriormente neste texto, podemos dizer que o data warehouse é um conjunto de dados integrados, a partir de diversas fontes heterogêneas, em uma grande base de dados [GM96]. Este data warehouse sumaria os dados, detalhados e armazenados, ao longo de diversas dimensões do negócio e mantém estes dados sumariados para o processamento das consultas efetuadas através de ferramentas de suporte à decisão e de ferramentas OLAP.

Uma outra forma de implementar um data warehouse é defini-lo como um conjunto de *visões materializadas* que integram os dados a partir de múltiplas fontes heterogêneas, e eventualmente distribuídas, de informação [DEB95].

Uma visão é uma relação derivada, definida em termos de relações base, que é computada todas as vezes que uma referência a ela é feita. Uma visão é dita materializada quando ela é realmente armazenada na base de dados em vez de ser computada a partir das relações base em resposta a consultas [QGMW97]. Uma visão materializada pode ser vista como um cache – uma cópia dos dados que pode ser acessada rapidamente

Além de definir o próprio data warehouse como um conjunto de visões materializadas baseadas nos dados dos ambientes operacionais [QGMW97, Gup97], estas visões também podem ser utilizadas com o objetivo de otimizar consultas complexas [Qua97], sendo que, para isto, devemos definir um conjunto compartilhado de visões, criteriosamente escolhidas a partir de uma análise das consultas mais freqüentes executadas no data warehouse. Estas visões, ao serem materializadas, agilizariam o acesso aos dados necessários para a realização das consultas no data warehouse[YKL96], aspecto crítico e diferencial nas aplicações onde a quantidade de consultas é alta e as visões são complexas ou são definidas sobre dados em bases remotas.

Estes dois temas têm apresentado grande relevância, já que a utilização dos data warehouses vem aumentando significativamente e inclui aspectos como otimização das consultas, replicação dos dados e a construção e otimização das ferramentas de suporte à decisão e OLAP. Vários trabalhos vêm sendo realizados envolvendo a utilização das visões materializadas em sistemas de data warehouse, com o objetivo de conferir a estas enormes bases de dados, melhor integração das informações, facilidades no processo de carga e, obviamente, otimização das consultas efetuadas.

Como a tecnologia de visões materializadas pode apoiar todo o processo de implantação e utilização dos data warehouses? Quais as vantagens em se utilizar este tipo de visões? Quais os limites encontrados nesta abordagem? Estas serão algumas das questões respondidas ao longo das próximas seções.

A utilização das visões materializadas pode conferir ao data warehouse um dinamismo não encontrado na abordagem estática discutida anteriormente. Com o uso das visões, o data warehouse pode manter-se atualizado e sincronizado com as bases de dados transacionais originais, em intervalos de tempo bem menores, já que não dependerá do processo de carga periódico para isto. Tipicamente, uma visão é mantida imediatamente, como parte da transação que atualiza as tabelas bases, quando da inserção ou eliminação de alguma linha destas tabelas. Esta forma de atualizar as visões causa, porém, uma sobrecarga ao processo de atualização das tabelas base. Uma outra opção é protelar esta manutenção por um certo período de tempo e ativá-

la em intervalos predefinidos ou sob demanda. É importante, no processo de atualização das visões, levar em conta dois fatores básicos: minimizar o tempo em que a visão fica "bloqueada" para acesso (durante sua manutenção) e minimizar a sobrecarga imposta às transações de atualização das tabelas base considerando que este processo passa por, basicamente, duas etapas. A primeira etapa consiste da definição das alterações que devem ser aplicadas nas visões e a segunda, na aplicação destas atualizações.

Fica claro que desta abordagem mais dinâmica do data warehouse, que utiliza visões materializadas, surgem vários problemas. Um deles é a decisão sobre a selecão das visões mais adequadas para a implantação do data warehouse. É necessário analisar os custos de manutenção em função dos benefícios na utilização de cada visão. Outro aspecto que deve ser analisado se refere à capacidade de automanutenção das visões, minimizando ao máximo o acesso às bases remotas para manter o data warehouse sincronizado com as fontes de informação originais. Além disso, as técnicas de manutenção das visões dependem da definição de quando serão aplicadas as alterações. Todos estes problemas serão também discutidos nas próximas seções, incluindo as soluções desenvolvidas para viabilizar a utilização desta abordagem nos data warehouses.

4.1 O uso das visões materializadas no data warehouse

Como já citado, uma visão materializada é uma consulta cujo resultado já está computado e armazenado na base de dados [Qua97]. As consultas que puderem utilizar as visões já armazenadas podem ser executadas de forma muito mais rápida, sendo que, para consultas complexas envolvendo grandes volumes de dados, esta alternativa favorece dramaticamente os resultados: de horas ou dias para segundos ou minutos. De fato, as visões materializadas são vistas como uma das principais opções para o controle do desempenho de um data warehouse [Kim96].

A desvantagem das visões materializadas é que as alterações feitas aos dados base, a partir dos quais uma visão materializada é definida, torna a visão desatualizada. Para que a visão possa estar novamente sincronizada aos dados base, será necessário recriar a visão a partir dos dados de origem ou então, atualizá-la incrementalmente [Qua97].

Outro aspecto importante é que estas atualizações podem ser executadas imediatamente, tão logo a alteração é recebida, ou podem ser proteladas, de tal forma que todas as alterações ocorridas nos dados base serão reunidas e aplicadas em processos batch.

O uso de visões materializadas nos data warehouses, conforme descrito nos parágrafos acima, requer, portanto, a análise de alguns tópicos relacionados, como:

1. a seleção das visões mais adequadas levando-se em conta uma análise dos custos de manutenção e os benefícios de cada visão e os algoritmos disponíveis;
2. aspectos da materialização e utilização das visões para responder às consultas;
3. os mecanismos que permitem a propagação correta quando da atualização das fontes de dados base, para vários tipos de visões, preferencialmente de forma online;
4. a manutenção das visões de forma dinâmica e incremental;
5. a criação de algoritmos que permitam a manutenção das visões de maneira autônoma, utilizando o conceito de *self maintainable views*.

A seguir descreveremos as visões materializadas e alguns aspectos referentes à sua definição, seleção, utilização e manutenção.

4.2 Selecionando as visões a serem materializadas

[GM95] define uma visão como uma relação derivada, em termos das relações base armazenadas. Uma visão é, portanto, uma função que parte de um conjunto de tabelas base para uma tabela derivada, sendo que esta função é recalculada todas as vezes que é referenciada.

Uma visão pode ser materializada, através do armazenamento das tuplas da visão na base de dados. Com esta materialização, é possível criar índices que tornarão o acesso a estas visões materializadas muito mais rápido que o recálculo, que é executado todas as vezes que as visões são referenciadas.

Conforme descrito em [YKL96], um data warehouse, devido aos diferentes tipos de análises, pode conter múltiplas visões. Quando estas visões estão relacionadas umas com as outras, isto é, quando estão definidas sobre partes sobrepostas dos dados base, então pode ser mais eficiente materializar apenas certas visões compartilhadas, ou porções dos dados base, em vez de materializar todas as visões. Assim, surge o primeiro aspecto a ser analisado, que se refere à escolha das visões que serão materializadas. Esta escolha se baseia na determinação de um conjunto de visões, de uso compartilhado no data warehouse, de modo a combinar bom desempenho com baixo custo de manutenção. O objetivo é selecionar um conjunto apropriado de visões que minimize o tempo total de reposta das consultas e o custo de manutenção das visões selecionadas, dado um certo limite de recursos, como por exemplo, tempo de materialização, espaço para armazenamento etc.

Vários trabalhos tratam deste tema, como por exemplo, [HRU96] que apresenta e analisa algoritmos para a seleção das visões em um caso especial de "cubos de dados". [GHRU96] estende os resultados para a seleção de visões e índices em cubos de dados, porém ambos ignoram os custos de manutenção das visões. Uma estrutura teórica é definida em [Gup97] para o problema geral de seleção de visões, apresentando uma heurística para alguns casos especiais importantes de problemas que ocorrem na prática. [YKL96] define aspectos de uma metodologia para desenho das visões materializadas, por exemplo, como selecionar um conjunto de resultados intermediários das consultas a serem materializados de forma que o custo total seja mínimo. Apresenta também um algoritmo para escolha deste conjunto, levando em conta as freqüências das consultas e as freqüências das atualizações nos dados base. Todo o processo de escolha das visões está baseado em um plano de processamento das visões envolvidas (MVPP – Multiple View Processing Plan), gerado por um algoritmo, que parte dos planos individuais de cada consulta envolvida. O trabalho apresentado por [YKL96] analisa o desenho da visão materializada em termos de desempenho, levando em conta também os recursos utilizados na manutenção da visão. A discussão é apresentada em termos do modelo relacional com operações *select*, *project* e *join*, sendo que a abordagem utilizada pode ser estendida para operações mais complexas, como consultas com agregações e consultas recursivas.

4.2.1 Uma descrição resumida do trabalho de [YKL96]

Nesta seção serão apresentados os conceitos do trabalho realizado por [YKL96]. A apresentação destes conceitos estará baseada em uma base de dados exemplo, que contém as seguintes relações:

Product[6] (Pid, name, Did)

Division (Did, name, city)

Order (Pid, Cid, quantity, date)

Customer (Cid, name, city)

Part (Tid, name, Pid, supplier)

Para simplificar os diagramas, **Pd, Div, Ord, Cust** e **Pt** representam, respectivamente, as relações acima. Além disso, assume-se que estas relações encontram-se todas no mesmo local e, portanto, serão desconsiderados os custos de comunicação de dados para os cálculos que se seguirão.

Supondo que temos as seguintes consultas, freqüentemente utilizadas no acesso ao data warehouse:

Consulta 1:

```
Select Pd.name
From   Pd, Div
Where Div.city = "LA" and Pd.Did=Div.Did
```

Consulta 2:

```
Select Pt.name
From Pd, Pt, Div
Where Div.city = "LA" and Pd.Did=Div.Did
                      and Pt.Pid=Pd.Pid
```

Para cada uma das consultas será gerado um grafo de processamento, que representa seu plano de acesso individual, conforme representado na figura abaixo:

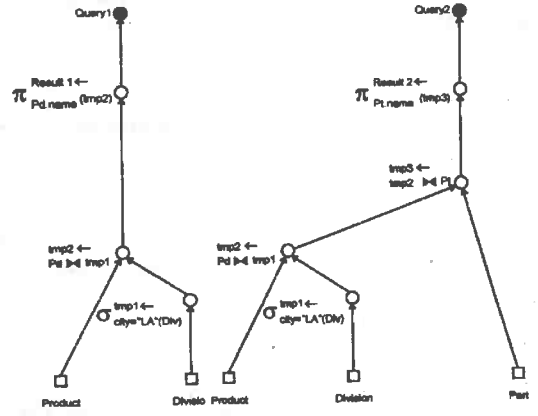

Figura 31 – Plano de acesso individual para as consultas 1 e 2 (query1 e query2).

[6] Os nomes dos atributos e das relações envolvidas no exemplo serão mantidos no idioma original.

Uma das alternativas para obter um rápido tempo de respostas para as consultas acima, seria materializar alguns dos nós intermediários de cada plano de acesso individual, sendo que, para o cálculo do custo total, os custos de manutenção das visões também deveriam ser levados em conta. O nó intermediário tmp2 da query1 é equivalente ao da query2, na figura 31 e são chamados de sub-expressões comuns. Os dois planos individuais podem ser combinados de modo a formar um único plano conforme mostra a figura 32, abaixo:

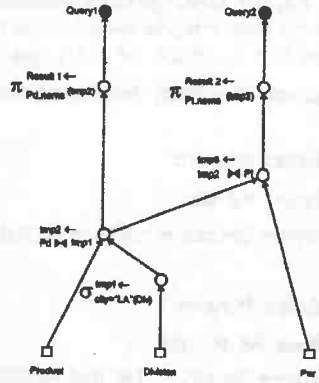

Figura 32 – Plano de acesso combinado para query1 e query2.

Fica claro que, se o nó identificado como tmp1 for materializado, pode ser utilizado pelas duas consultas, em vez do acesso às relações base Product, Division e Part, diminuindo assim seus custos de execução. Além disso, o custo de manutenção de apenas um nó tmp1 será menor que o de dois tmp1s. Desta forma, haverá ganhos em termos de custo total do acesso global e da manutenção da visão.

Agora, vamos supor duas outras consultas freqüentes ao data warehouse:

Consulta 3:

```
Select Cust.name, Pd.name, quantity
From   Pd, Div, Ord, Cust
Where Div.city = "LA" and Pd.Did=Div.Did
                      and Pd.Pid=Ord.Pid and Ord,Cid=Cust.Cid
                      and date>7/1/96
```

Consulta 4:

```
Select Cust.city, date
From Ord, Cust
Where quantity>100 and Ord.Cid=Cust.Cid
```

A figura 33 representa um plano de acesso global para as quatro consultas, de forma que os planos de acesso locais ou individuais foram combinados com base nas operações compartilhadas dos dados comuns. Este plano global é chamado de **Multiple View Processing Plan (MVPP)**. Após a criação deste plano global, pode-se decidir quais são os nós a serem materializados, de forma que o custo da consulta de o da manutenção da visão sejam mínimos.

Figura 33 – Um MVPP para o exemplo.

Obviamente, no grafo, existem várias opções para a escolha do conjunto de visões materializadas, a saber: (1) materializar todas as consultas; (2) materializar alguns dos nós intermediários, por exemplo, tmp1, tmp2, tmp4 etc; (3) manter virtuais todos os nós que não sejam folhas. Para a decisão do melhor conjunto, o custo de cada alternativa deve ser calculado em termos do processamento da consulta e da manutenção da visão.

O trabalho apresentado em [YKL96] define dois algoritmos para implementar uma solução para o problema:

1. **Algoritmo para a definição de múltiplos MVPP:**

   Normalmente para uma consulta existem vários planos de processamento, sendo que, entre eles existe um plano considerado ótimo. Da mesma forma, pode-se ter múltiplos MVPPs baseados em diferentes combinações destes planos individuais. Para reduzir o espaço da pesquisa, o trabalho parte de planos individuais ótimos e os ordena, baseado nas freqüências das consultas e seus custos. Uma vez que a ordem está definida, o algoritmo trabalha com os vários planos individuais, combinando suas sub-expressões comuns chegando a um conjunto de k planos globais ou k MVPPs. Cada MVPP gerado tem um custo total que será analisado pelo segundo algoritmo descrito em seguida.

2. **Algoritmo para selecionar os nós intermediários a serem materializados:**

   Dado um MVPP, o objetivo é definir um conjunto de visões materializadas tal que o custo total do processamento da consulta e da manutenção da visão seja mínimo, tentando comparar os custos de cada combinação de nós possível. O trabalho apresentado propõe um algoritmo que compara o custo total de cada MVPP gerado, selecionando o de menor custo.

4.3 Caracterizando alguns aspectos da manutenção das visões

Por se comportar como um cache, uma visão materializada oferece acesso rápido aos dados, sendo que esta velocidade pode ser crítica em aplicações onde a quantidade de consultas é alta e a complexidade das visões leva à impossibilidade de recálculo da visão em todo acesso.

Assim, como um cache que se torna obsoleto quando seus dados base sofrem atualização, as visões materializadas também ficam desatualizadas quando as bases de dados de origem são modificadas. Neste caso, é necessária a execução de um processo de manutenção sendo que, na maioria dos casos, é desperdício de recursos proceder com seu recálculo a partir do zero [GM95]. Isto porque apenas uma parte da visão muda em resposta às alterações das bases originais e, portanto, é possível alterar o conteúdo da visão materializada, de forma incremental. Obviamente, isto é apenas heurística. Se tivermos, por exemplo, a eliminação da relação base como um todo, será mais barato recalcular a visão, que depende desta relação eliminada, de forma total, do que apenas aplicar as alterações. Vários trabalhos descrevem algoritmos que permitem a manutenção incremental das visões materializadas, com diferentes domínios de aplicabilidade, como [SI84, CW91, HD92, GMS93].

[GM95] propõe uma classificação do problema de manutenção das visões, através de quatro dimensões, ao longo das quais este problema pode ser analisado, a saber:

- Dimensão da Informação: Refere-se à quantidade de informação disponível para a manutenção da visão. Na análise sob esta dimensão, pergunta-se se o acesso é possível a toda ou apenas parte das relações base e se existe acesso à visão materializada, se são conhecidas regras de integridade e chaves. A quantidade de informação utilizada é ortogonal à "incrementabilidade" da manutenção da visão. "Incrementabilidade" se refere à capacidade de se computar ou calcular apenas aquela parte da visão que foi alterada. Nesta dimensão, portanto, analisamos os dados utilizados para computar as alterações na visão.
- Dimensão da Modificação: Nesta dimensão são analisadas as modificações que o algoritmo de manutenção da visão pode tratar. Questiona-se, então, se inserções ou eliminações de tuplas nas tabelas base são suportadas pelo algoritmo, se as atualizações nas tuplas são tratadas diretamente ou se são modeladas como eliminações seguidas por inserções e aspectos relacionados às alterações na definição da visão.
- Dimensão da Linguagem: Esta dimensão está relacionada a questões sobre a forma com que a visão está expressa, ou seja, foi definida como uma consulta do tipo *select-project-join* (também conhecida como visão SPJ ou como consulta conjuntiva) ou baseada em algum outro subconjunto da álgebra relacional, se utiliza a SQL ou um subconjunto da SQL, se a visão possui duplicidades e se utiliza agregação e recursão.
- Dimensão das Instâncias: Esta dimensão divide-se em dois tipos: informações relativas às instâncias da base de dados e às instâncias da modificação. É importante saber se o algoritmo de manutenção da visão funciona para todas as instâncias da base de dados ou apenas para algumas e se funciona para todas as instâncias da modificação ou apenas para algumas.

Para esclarecer a utilização desta classificação, [GM95] propõe alguns exemplos, que serão transcritos abaixo:

Exemplo 1: Dimensões da Informação e Modificação

Considere a relação:

$$\text{part(part\_no, part\_cost, contract)}$$

que lista o custo de uma peça, negociado em um contrato. Note que uma peça pode ter preços diferentes por contrato. Considere também a visão expensive_parts definida como:

$$\text{expensive\_parts(part\_no)} = \Pi_{\text{part\_no}}\, \sigma_{\text{part\_cost} > 1000}(\text{part})$$

Esta visão contém números de peça **distintos** para as peças que custam mais que $1000 presentes em, ao menos, um contrato (a operação de projeção descarta as duplicidades). Considere a manutenção da visão quando uma tupla for inserida na relação part. Se a tupla inserida tiver part_cost ≤ 1000, então a visão não sofrerá alterações. Entretanto, suponha que part($p1$, 5000, $c1$) foi inserida, tendo custo>1000. Algoritmos distintos podem ser definidos, dependendo da informação disponível para determinar se $p1$ deve ser ou não inserido na visão:

- Apenas a visão materializada está disponível: utiliza-se a antiga visão materializada para determinar se a peça de número part_no já está presente na visão. Se já estiver, não há alteração na materialização, caso contrário, deve-se inserir a peça $p1$ na materialização.
- Apenas a relação base part está disponível: utiliza-se a relação part para verificar se alguma tupla existente na relação tem o mesmo número part_no, porém com custo igual ou maior. Se tal tupla já existir, então a nova tupla inserida não contribui para a visão.
- É conhecido que part_no é a chave: infere-se que a peça de part_no não está na visão e, portanto, deve ser inserida.

Um outro problema de manutenção das visões está relacionado a eliminações utilizando apenas a visão materializada. Considere-se a eliminação da tupla part($p1$, 2000, $c12$). Está claro que a peça $p1$ tem que estar na materialização, porém não se pode eliminar $p1$ da visão, já que alguma outra tupla, como part($p1$, 3000, $c13$), pode ter contribuído em levar a peça $p1$ para a visão. A existência (ou inexistência) desta tupla não pode ser provada utilizando-se apenas a visão. Desta forma, não existe um algoritmo que possa resolver este problema de manutenção para eliminações, utilizando apenas a visão materializada. É importante notar que, se a relação part estivesse disponível, ou se fosse conhecida a restrição da chave ou ainda, se a quantidade de tuplas derivadas estivessem disponíveis, então a visão poderia ser mantida.

Com respeito à dimensão da informação, deve-se notar que tanto a definição da visão quanto a alteração real, estão sempre disponíveis para o processo de manutenção e, em relação à dimensão da modificação, as atualizações são tratadas, tipicamente, como uma eliminação seguida por uma inserção.

Exemplo 2: Dimensões da Linguagem e das Instâncias

O exemplo anterior considerou uma visão, cuja definição consiste de operações de seleção e projeção. Para este segundo exemplo, a linguagem de definição da visão será estendida para implementar também a operação *join* , sendo que a visão supp_parts é definida como um *equijoin* entre as relações supp(supp_no, part_no, price) e part($\times_{part\_no}$ representa um *equijoin* no atributo part_no):

$$supp\_parts(part\_no) = \Pi part\_no(supp \times_{part\_no} part)$$

A visão contém os números distintos das peças que são fornecidas por, no mínimo, um fornecedor (a operação de projeção descarta as duplicidades). Para o processo de manutenção deve-se considerar apenas a utilização do conteúdo antigo da visão supp_parts. A inserção da peça part($p1$, 5000, $c15$), será analisada neste exemplo. Se supp_parts já contiver a peça com part_no $p1$, então esta inserção não afetará a visão existente. Entretanto, se supp_parts não contiver $p1$, então o efeito da inserção não pode ser determinado utilizando-se apenas a visão.

É importante lembrar que a visão expensive_parts, do exemplo anterior, sofreu o processo de manutenção em decorrência das inserções em part, utilizando apenas a visão. Diferentemente, a operação *join* torna impossível a manutenção de supp_parts, como consequência das inserções em part, utilizando-se apenas a visão. Com isso, a visão supp_parts pode ser mantida se já contiver a peça com part_no $p1$, caso contrário, não. Portanto, a capacidade de sofrer manutenção de uma visão depende também de instâncias particulares da base de dados e da modificação.

4.4 Mantendo as visões de forma incremental

Como uma visão materializada é uma relação derivada e definida a partir de relações base, operações de inserção, eliminação e atualização nestas bases de origem fazem com que a visão se torne desatualizada. Recriar a visão como um todo, principalmente em data warehouses onde as relações existentes contêm um número muito grande de elementos, pode ser um processo caro, demorado e ineficiente, dado que, na maior parte dos casos, a janela de tempo disponível para a manutenção do data warehouse é limitada. Uma alternativa bastante utilizada é aplicar na visão, apenas as alterações. Este tipo de manutenção é chamado de manutenção incremental das visões.

A maioria das técnicas de manutenção define as visões como uma fórmula matemática e obtém uma expressão que representa o processo de manutenção. O exemplo abaixo ilustra estas técnicas, conforme [GM95]:

Exemplo 3:

Seja a relação base link(S, D), de tal forma que link(a, b) é verdadeiro, se existe uma ligação que parte do nó a e chega no nó b. A visão hop é definida de tal forma que hop(c, d) é verdadeiro, se c está conectado a d utilizando duas ligações, através de um nó intermediário:

$$D: \quad \text{hop}(X, Y) = \Pi_{X,Y}(\text{link}(X, V) \times_{V=W} \text{link}(W, Y))$$

Seja um conjunto de tuplas Δ(link), inserido na relação link. As inserções Δ(hop) que devem ser efetuadas na visão hop, de forma correspondente, podem ser computadas matematicamente, pela definição da diferenciação D, para se obter a seguinte expressão:

$$\Delta(\text{hop}) = \Pi_{X,Y}((\Delta(\text{link})(X, V) \times_{V=W} \text{link}(W, Y)) \cup (\text{link}(X, V) \times_{V=W} \Delta(\text{link})(W, Y)) \cup (\Delta(\text{link})(X, V) \times_{V=W} \Delta(\text{link})(W, Y)))$$

No exemplo 3 citado acima, se as tuplas forem eliminadas da relação link, a mesma expressão pode computar as eliminações da visão hop. Se as tuplas forem inseridas e eliminadas da relação link, então Δ(hop) pode ser obtido através do processamento do conjunto de eliminações Δ^-(hop) e do conjunto de inserções Δ^+(hop), separadamente [QW91, HD92]. Em [GMS93], para casos especiais, inserções e eliminações podem ser tratados em um único passo.

O trabalho apresentado em [GMS93] apresenta dois algoritmos para manutenção incremental que aplica nas visões materializadas as alterações (inserções, eliminações e atualizações) ocorridas nas relações base, sendo que as visões podem utilizar uniões, negações, agregações (como SUM, MIN) e recursões. Os dois algoritmos utilizam a definição da visão para produzir regras que implementarão as alterações nas visões utilizando, para isto, as alterações ocorridas nas relações base e as visões materializadas na situação anterior às modificações.

O primeiro algoritmo descrito em [GMS93] é um algoritmo de *counting*. Este algoritmo pode ser utilizado em visões com ou sem linhas duplicadas. Sua idéia básica é manter um contador do número de derivações para cada tupla da visão. Para descrevê-lo de forma mais detalhada, dada sua importância, será utilizado como base, o exemplo 3 citado acima, que utiliza a relação base link e a visão hop.

Exemplo 4: A definição da visão hop, em SQL é:

```
CREATE VIEW hop(S,D) as
        (select distinct l1.S, l2.D from link l1, link l2 where l1.D = l2.S)
```

Seja a relação link = {(*a*,*b*), (*b*,*c*), (*b*,*e*), (*a*,*d*), (*d*,*c*)} e, como resultado, a visão hop {(*a*,*c*), (*a*,*e*)}. A tupla hop (*a*,*e*) tem ocorrência única, ou seja, é derivada uma única vez de link, sendo que, a tupla hop (*a*,*c*) possui duas derivações. Se a visão tivesse duplicidade, ou seja, não tivesse sido definida com o operador **distinct**, então hop (*a*,*e*) teria ***count*** igual a 1 e hop (*a*,*c*) teria ***count*** igual a 2. O algoritmo de *counting* simula a duplicidade na visão e armazena estes contadores.

Vamos supor que a tupla link (*a*,*b*) seja eliminada. Neste caso hop seria recomputada como {(*a*,*c*)}. O algoritmo de *counting* infere que uma derivação de cada uma das tuplas hop (*a*,*c*) e hop (*a*,*e*) deve ser eliminada. Baseado nos contadores armazenados, infere também que hop (*a*,*c*) possui ainda uma derivação e portanto, apenas hop (*a*,*e*), que não possui mais nenhuma derivação, deve ser eliminada.

O segundo algoritmo descrito em [GMS93] não pode ser utilizado em visões com duplicidade de tuplas, ou seja, em visões definidas sem o operador **distinct**. Este algoritmo, de nome **DRed** (*Deletion and Rederivation*) calcula as alterações nas visões em três passos. No primeiro passo, o algoritmo faz uma super estimativa das tuplas derivadas de devem ser eliminadas. Uma tupla *t* está nesta super estimativa se as alterações feitas nas relações base invalidam qualquer ocorrência de *t*. Em um segundo passo, esta superestimativa sofre a remoção daquelas tuplas que tenham uma ocorrência ou derivação alternativa na nova base de dados. Finalmente, as novas tuplas que devem ser inseridas são computadas utilizando-se a visão materializada parcialmente atualizada e as inserções feitas nas relações base. Para ilustrar este algoritmo, vamos considerar o seguinte exemplo:

Exemplo 5:

Seja a visão hop, definida no exemplo 4 e a eliminação da tupla link (*a*,*b*). O algoritmo **DRed** primeiramente elimina as tuplas hop (*a*,*c*) e hop (*a*,*e*), já que ambas dependem da tupla eliminada. O algoritmo procura, então, pelas derivações alternativas de cada uma das tuplas eliminadas. Assim, no segundo passo, a tupla hop (*a*,*c*) é novamente derivada e inserida na visão materializada. O terceiro passo do algoritmo é vazio, para este exemplo, já que não existem tuplas a serem inseridas na tabela link.

O processo de manutenção incremental das visões depende da quantidade de informações disponíveis. Conforme descrito em [GM95], vários trabalhos foram desenvolvidos apresentando algoritmos para a manutenção das visões em função da quantidade de informações disponíveis e do tipo de visão.

Foram analisados os casos onde toda a informação está disponível para o processo de manutenção, ou seja, todas as relações base e as visões materializadas estão disponíveis durante o processo de manutenção. Estes trabalhos consideram também, características da formação das visões. Segue abaixo uma breve descrição dos vários algoritmos definidos nestes trabalhos, agrupados pela quantidade de informação disponível e pelo tipo de visão que pode ser tratada.

1. **Utilizando informação completa:** A maioria dos trabalhos em manutenção de visões assume que todas as relações base e as visões materializadas estão disponíveis para o processo de manutenção, considerando características como agregações, duplicidades, recursão e *outer join*. As técnicas diferem quanto à linguagem de definição utilizada, quanto ao uso das chaves e regras de integridade e quanto à capacidade de tratar inserções e eliminações de forma separada ou em apenas um passo. As atualizações são modeladas e tratadas como uma eliminação seguida de uma inserção. Todas as técnicas funcionam para todas as instâncias da base de dados tanto para inserções como para eliminações de tuplas.

   1.1. **Visões não recursivas:** as técnicas apropriadas neste caso incluem o algoritmo de *counting* (descrito acima) apresentado por [GMS93], além de outros algoritmos que também utilizam contadores, como [SI84] que cria estruturas com ponteiros que partem da tupla de origem para suas tuplas derivadas. [BLT86] utiliza os contadores exatamente como o algoritmo *counting*, porém apenas para visões do tipo SPJ, computando inserções e eliminações separadamente. O algoritmo de manutenção por diferenciação algébrica, introduzido por [Pai84] e utilizado posteriormente em [QW91] para manutenção de visões, diferencia expressões algébricas para derivar a expressão relacional que determina a alteração em uma visão SPJ. Este tipo de algoritmo gera duas expressões para cada visão: uma para definir as inserções na visão e outro, para definir as eliminações. O algoritmo Ceri-Widom [CW91] define regras de produção para manter as visões SQL que não possuam duplicidades, agregações e negação e aquelas onde os atributos das visões determinam funcionalmente a chave da relação base que está sendo atualizada.

   1.2. **Visões com *outer-join*:** a manutenção de visões com *outer-join* é discutida em [GJM94], cujo algoritmo redefine a visão, obtendo duas instruções (uma com *left-outer-join* e outra com *right-outer-join*) para calcular as alterações a serem implementadas.

   1.3. **Visões recursivas:** Os algoritmos citados nesta seção aplicam-se também a visões não recursivas. O primeiro deles é o algoritmo DRed [GMS93], descrito acima. Vários outros algoritmos se aplicam a este tipo de visão, estando entre eles: o algoritmo PF (Propagation/Filtration) definido em [HD92], que é muito similar ao DRed, sendo que, para visões não recursivas, o DRed sempre funciona melhor; o algoritmo Kuchenhoff [Kuc91] que cria regras para calcular a diferença entre estados consecutivos da base de dados e o algoritmo Urpi-Olive [UO92] que cria regras de transição mostrando como cada modificação na relação base se traduz em uma modificação em cada relação derivada.

2. **Utilizando informação parcial:** As visões podem ser mantidas utilizando-se somente um subconjunto das relações envolvidas na definição da visão. De forma diferente da utilização da informação completa, quando a manutenção utiliza apenas informação parcial, nem sempre é possível realizar o processo de manutenção da visão. A manutenção vai depender de aspectos como o tipo de modificação, ou seja, se é uma inserção, uma eliminação ou

atualização. Portanto, os algoritmos neste caso devem se preocupar se a visão pode ser mantida e então, como efetuar a manutenção, sendo que em alguns casos, mesmo que não exista o algoritmo para eliminação+inserção, é possível existir o algoritmo para atualização.

2.1. **Nenhuma informação disponível:** Muitos trabalhos têm sido feitos para determinar quando uma modificação na base não acarreta alteração na visão. Esta situação é conhecida como "o problema da atualização irrelevante". [BLT86, BCL89, Elk90, LS93] determinam atualizações irrelevantes em várias condições diferentes.

2.2. **Utilizando a visão materializada:** ***Self-Maintenance.*** As visões podem ser mantidas utilizando-se apenas a visão materializada e as regras implementadas através das chaves. Estas visões são tratadas por [GJM94], que apresenta vários resultados em *self-maintenance* em visões do tipo SPJ e com *outer-join*, respondendo a inserções, eliminações e atualizações. Este trabalho define uma visão *self-maintainable* com respeito ao tipo de modificação, não considerando as visões com *self-joins* ou *outer-joins*, as visões que não usam informações das chaves e as que não consideram a capacidade de auto manutenção com respeito a todas as instâncias de modificações.

3. **Utilizando a visão materializada e algumas relações base: Referência parcial.** O problema da manutenção com referência parcial é manter a visão, dados apenas um subconjunto das relações base e a visão materializada. Dois subproblemas interessantes surgem desta abordagem: o primeiro é quando estão disponíveis a visão e todas as relações base, com exceção da relação que sofreu a modificação e o segundo, quando estão disponíveis a visão e a relação modificada. Vários trabalhos, como [JMS95, GB95, Gup94], tratam de variações desta situação.

4.5 Conclusão

A manutenção dinâmica das visões materializadas, de maneira similar à definição do processo de carga estático, não é uma tarefa simples. Muitas das etapas envolvidas na carga estática estão presentes na manutenção dinâmica, diluídas nos processos de atualização de cada visão materializada. Existem algumas opções relacionadas ao momento em que as visões são atualizadas. Basicamente, a atualização pode ser imediata, ou seja, no momento em que os dados base são atualizados, ou protelada, sendo que, neste caso, devem ser considerados os aspectos necessários para minimizar o tempo que as visões ficam indisponíveis para o acesso por parte dos usuários.

Normalmente, as bases de dados fonte, para o data warehouse, podem ser sistemas legados ou sistemas que, de uma forma geral, não conhecem as visões materializadas implementadas no ambiente analítico. As aplicações no ambiente fonte devem informar ao data warehouse quando ocorre uma atualização, por exemplo, a contratação de um novo funcionário ou um paciente que pagou sua conta hospitalar. Entretanto, estas aplicações não têm condições de determinar quais são os dados necessários para se implementar as atualizações nas visões materializadas do data warehouse. Quando a informação sobre uma atualização chega ao data warehouse, temos que buscar os dados adicionais, nas bases fonte, para atualizar as visões corretamente e de forma incremental. Por esta razão, o warehouse deverá emitir algumas consultas para as bases fonte após a ocorrência da atualização. Existem trabalhos já realizados, que implementam vários algoritmos distintos, para garantir que as visões materializadas no data warehouse sejam atualizadas de forma correta, garantindo a consistência das visões, no menor tempo possível. A

escolha do algoritmo mais adequado depende de uma série de fatores. Aspectos como: o tipo da visão implementada, sua dependência das bases de dados fonte durante o processo de atualização, incluindo sua capacidade de se auto manter sem a necessidade de acesso às bases originais, a distribuição dos ambientes fonte e do data warehouse e a concorrência no momento da atualização devem ser levados em conta na escolha do algoritmo de propagação das atualizações mais adequado.

Fica claro que, assim como no modelo de data warehouse estático, com processos de carga periódicos, conforme descrito na seção 3.7 deste trabalho, a atualização do data warehouse em uma implementação mais dinâmica também é a etapa mais complexa e crítica de se implementar.

Ocorre que, para alguns tipos de aplicações transacionais em conjunto com as aplicações analíticas a abordagem estática é preferível, em detrimento da dinâmica e, para outros tipos, ocorre o inverso.

Como um data warehouse pode compreender várias áreas de negócios distintas, associadas a diferentes aplicações transacionais e usuários com necessidades de análise específicas, o modelo resultante pode não ser homogêneo, ou seja, apenas estático ou apenas dinâmico.

A determinação das características da implementação mais adequada de um determinado modelo analítico estará baseada em uma análise baseada em dois conjuntos de informação:

1. Classificação dos domínios das aplicações transacionais, levando-se em conta o modelo de negócios envolvido e as características operacionais da implementação e utilização das informações geradas pela aplicação;
2. Classificação da utilização do ambiente analítico, levando-se em conta as características das informações a serem disponibilizadas e as formas de sua utilização.

Com o levantamento destas classificações, poderão ser avaliados os aspectos do comportamento transacional dos ambientes operacional e analítico, sendo que a análise conjunta destas características resultará em diretrizes para que os projetistas de data warehouse possam definir o modelo mais adequado para sua implementação.

Como um data warehouse pode compreender várias áreas de negócios distintas, associadas a diferentes aplicações transacionais e usuários com necessidades de análise específicas, o modelo resultante pode não ser homogêneo, ou seja, apenas estático ou apenas dinâmico. A análise das classificações, descritas acima, poderão levar, em muitos casos, à definição de um modelo híbrido de data warehouse, onde determinadas porções terão características estáticas enquanto que outras sofrerão manutenção dinâmica. Devemos considerar, também, que a proporção entre as partes dinâmicas e estáticas do data warehouse pode ser alterada em função da evolução das aplicações.

Referências

[AV98] C. Adamson, M. Venerable. *Data Warehouse Design Solutions*, John Wiley & Sons, 1998.

[BCL89] J.A. Blakeley, N. Coburnm, P. Larson. "Updating derived relations: detecting irrelevant and autonomously computable updates". *AM Transactions on Database Systems*, 14(3):369-400, 1989.

[BLT86] J.A. Blakeley, P. Larson, F. Tompa. "Efficiently updating materialized views". *Proceedings of ACM SIGMOD*, 61-71, Washington D.C., 1986..

[BPT97] E. Baralis, S. Paraboschi, E. Teniente. "Materialized view selection in a Multidimensional Database". *Proceedings of the 23rd VLDB Conference*, Atenas, Grécia, 1997.

[CW91] S. Ceri, J. Widom. "Deriving incremental production rules for incremental view maintenance". *Proceedings of 17th VLDB*, 577-589, Barcelona, Espanha, 1991.

[DEB95] *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2), junho 1995.

[Elk90] C. Elkan."Independence of logic database queries and updates". *Proceedings of 9th ACM Symposium on Principles of Database Systems*. 154-160, 1990.

[GB95] A. Gupta, J. A. Blakeley. *Maintaining views using materialized views.* Documento não publicado.

[GHRU96] H. Gupta, V. Harinarayan, A. Rajaraman, J. Ullman. "Index selection in OLAP", *Proceedings 13th ICDE*, 208-219, Manchester, GB, 1997.

[GM95] A.Gupta, I. S. Mumick. "Maintenance of materialized views: problems, techniques, and applications", *IEEE Data Engineering Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):3-19, Junho, 1995.

[GM96] A. Gupta e I. S. Mumick. "What is the data warehousing problem? (Are materialized views the answer?)". *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), Índia, 1996.

[GMR98] M. Golfarelli, D. Maio, S. Rizzi. "Conceptual Design of Data Warehouses from E/R Schemes", *Proceedings of the Hawaii International Conference on System Sciences*, Kona, Hawaii, USA, 1998.

[GMS93] A.Gupta, I. S. Mumick, V. S. Subrahmanian. "Maintaining views incrementally", *Proceedings ACM SIGMOD International Conference on Management of Data*, 157-166, Washington, D.C., 1993.

[Gup94] A.Gupta. *Partial information based integrity constraint checking*. Tese de doutoramento da Stanford University (CS-TR-95-1534), 1994.

[Gup97] H. Gupta. "Selection of views to materialize in a Data Warehouse", *Proceedings Sixth International Conference on Database Theory ICDT*, 98-112, Delphi, 1997.

[HD92] J. Harrison, S. Dietrich. "Maintenance of materialized views in a deductive database: an update propagation approach". *Proceedings Workshop on Deductive Databases, JICSLP*, 1992.

[HRU96] V. Harinarayan, A. Rajaraman, J. Ullman. "Implementing data cubes efficiently". *Proceedings ACM Sigmod Intl. Conf. on Mngt. of Data*, 205-216, Montreal, 1996.

[IA99] *Information Advantage, Decision Path™ Implementation Methodology.* "MyEureka!™ data warehouse requirements guide", 1999.

[IH97] W. H. Inmon, R. D. Hackathorn. *Como usar o Data Warehouse*. Infobook, Rio de Janeiro, 1997.

[JMS95] H. V. Jagadish, I. S. Mumick, A. Silberschatz. "View maintenance issues in the chronicle data model". *14th PODS*, pág. 113-124, 1995.

[Kel94] S. Kelly. *Data Warehousing The Route to Mass Customization*, John Wiley & Sons, 1994.

[Kim96] R. Kimball. *The data warehouse toolkit.* John Wiley & Sons, 1996.

[LS93] A.Y. Levy, Y. Sagiv. "Queries independent of updates". *19th VLDB*, pág. 171-181, 1993.

[MF01] L. A. Mantovani, J. E. Ferreira. *Uma alternativa para simplificação do sincronismo de dados históricos dos Ambientes Operacionais e Analíticos.* Dissertação de mestrado do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, São Carlos, 2001.

[Pai84] R. Paige. "Applications of finite differencing to database integrity control and query/transaction optimization". *Advances in Databse Theory*, pág. 170-209, Plenum Press, New York, 1984.

[QGMW97] D. Quass, A. Gupta, I. S. Mumick, J. Widom. *Making views self-maintainable for data warehousing.* 1997.

[Qua97] D. Quass. *Materialized views in data warehouses.* Tese de doutoramento do Departamento de Ciência da Computação da Stanford University. Agosto 1997.

[QW97] D. Quass, J. Widom. "On-line Warehouse View Maintenace". *Proceedings of ACM SIGMOD 1997 – International Conference on Management of data*, pág. 393-404, 1997.

[QW91] X. Qian, G. Wiederhold. "Incremental recomputation of active relational expressions". *IEEE Transactions on Knowledge and Data*

Engineering, 3(3): 337-341, 1991.

[Rad96] N.Raden. "Technology Tutorial: Modeling a Data Warehouse", disponível: http://techweb.cmp.com/iw/564/64oldat.htm, 1996.

[SBHD99] C. Sapia, M. Blaschka, G. Höfling, B. Dinter. "Extending the E/R model for the multidimensional paradigm". "Advances in database technologies", *Journal of Computer Science and Information Management*, vol. 2, N. 3, 1999.

[SI84] O. Shmueli, A. Itai. *Maintenance of views. Sigmod Record*, 14(2):240-255, 1984.

[SMKK98] S. Samtani, M. Mohania, V. Kumar, Y. Kambayashi. *ER Workshops*, 1998, pág. 81-92.

[Tan97] R. Tanler. *The Intranet Data Warehouse*, John Wiley & Sons, 1997.

[YKL96] J. Yang, K. Karlapalem, Q. Li. *A framework for designing materialized views in data warehousing environment.* Technical Report HKUST-CS96-35. Outubro 1996.

[ZGMHW94] Y. Zhuge, H. Garcia-Molina, J. Hammer e J. Widom. *View maintenance in a warehousing environment.* Technical report, Stanford University. Disponível: ftp://db.stanford.edu como pub/zhuge/1994/anomaly-full.ps. Outubro de 1994.

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1997 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail (mac@ime.usp.br).

Flávio Soares Corrêa da Silva e Daniela Vasconcelos Carbogim
A TWO-SORTED INTERPRETATION FOR ANNOTATED LOGIC
RT-MAC-9801, fevereiro de 1998, 17 pp.

Flávio Soares Corrêa da Silva, Wamberto Weber Vasconcelos, Jaume Agustí, David Robertson e Ana Cristina V. de Melo.
WHY ONTOLOGIES ARE NOT ENOUGH FOR KNOWLEDGE SHARING
RT-MAC-9802, outubro de 1998, 15 pp.

J. C.de Pina e J. Soares
ON THE INTEGER CONE OF THE BASES OF A MATROID
RT-MAC-9803, novembro de 1998, 16 pp.

K. Okuda and S.W.Song
REVISITING HAMILTONIAM DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9804, dezembro 1998, 17pp.

Markus Endler
AGENTES MÓVEIS: UM TUTORIAL
RT-MAC-9805, dezembro 1998, 19pp.

Carlos Alberto de Bragança Pereira, Fabio Nakano e Julio Michael Stern
A DYNAMIC SOFTWARE CERTIFICATION AN VERIFICATION PROCEDURE
RT-MAC-9901, março 1999, 21pp.

Carlos E. Ferreira e Dilma M. Silva
BCC DA USP: UM NOVO CURSO PARA OS DESAFIOS DO NOVO MILÊNIO
RT-MAC-9902, abril 1999, 12pp.

Ronaldo Fumio Hashimoto and Junior Barrera
A SIMPLE ALGORITHM FOR DECOMPOSING CONVEX STRUCTURING ELEMENTS
RT-MAC-9903, abril 1999, 24 pp.

Jorge Euler, Maria do Carmo Noronha e Dilma Menezes da Silva
ESTUDO DE CASO: DESEMPENHO DEFICIENTE DO SISTEMA OPERACIONAL LINUX PARA CARGA MISTA DE APLICAÇÕES.
RT-MAC-9904, maio 1999, 27 pp.

Carlos Humes Junior e Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9905, maio 1999, pp.

Carlos Humes Junior and Paulo José da Silva e Silva
STRICT CONVEX REGULARIZATIONS, PROXIMAL POINTS AND AUGMENTED LAGRANGIANS
RT-MAC-9906, maio 1999, 21 pp.

Ronaldo Fumio Hashimoto, Junior Barrera, Carlos Eduardo Ferreira
A COMBINATORIAL OPTIMIZATION TECHNIQUE FOR THE SEQUENTIAL DECOMPOSITION OF EROSIONS AND DILATIONS
RT-MAC-9907, maio 1999, 30 pp.

Carlos Humes Junior and Marcelo Queiroz
ON THE PROJECTED PAIRWISE MULTICOMMODITY FLOW POLYHEDRON
RT-MAC-9908, maio 1999, 18 pp.

Carlos Humes Junior and Marcelo Queiroz
TWO HEURISTICS FOR THE CONTINUOUS CAPACITY AND FLOW ASSIGNMENT GLOBAL OPTIMIZATION
RT-MAC-9909, maio 1999, 32 pp.

Carlos Humes Junior and Paulo José da Silva e Silva
AN INEXACT CLASSICAL PROXIMAL POINT ALGORITHM VIEWED AS A DESCENT METHOD IN THE OPTIMIZATION CASE
RT-MAC-9910, julho 1999, 13 pp.

Markus Endler and Dilma M. Silva and Kunio Okuda
A RELIABLE CONNECTIONLESS PROTOCOL FOR MOBILE CLIENTS
RT-MAC-9911, setembro 1999, 17 pp.

David Robertson, Fávio S. Corrêa da Silva, Jaume Agustí and Wamberto W. Vasconcelos
A LIGHTWEIGHT CAPABILITY COMMUNICATION MECHANISM
RT-MAC-9912, novembro 1999, 14 pp.

Flávio S. Corrêa da Silva, Jaume Agustí, Roberto Cássio de Araújo and Ana Cristina V. de Melo
KNOWLEDGE SHARING BETWEEN A PROBABILISTIC LOGIC AND BAYESIAN BELIEF NETWORKS
RT-MAC-9913, novembro 1999, 13 pp.

Ronaldo F. Hashimoto, Junior Barrera and Edward R. Dougherty
FINDING SOLUTIONS FOR THE DILATION FACTORIZATION EQUATION
RT-MAC-9914, novembro 1999, 20 pp.

Marcelo Finger and WanbertoVasconcelos
SHARING RESOURCE-SENSITIVE KNOWLEDGE USING COMBINATOR LOGICS
RT- MAC-2000-01, março 2000, 13pp.

Marcos Alves e Markus Endler
PARTICIONAMENTO TRANSPARENTE DE AMBIENTES VIRTUAIS DISTRIBUÍDOS
RT- MAC-2000-02, abril 2000, 21pp.

Paulo Silva, Marcelo Queiroz and Carlos Humes Junior
A NOTE ON "STABILITY OF CLEARING OPEN LOOP POLICIES IN MANUFACTURING SYSTEMS"
RT- MAC-2000-03, abril 2000, 12 pp.

Carlos Alberto de Bragança Pereira and Julio Michael Stern
FULL BAYESIAN SIGNIFICANCE TEST: THE BEHRENS-FISHER AND COEFFICIENTS OF VARIATION PROBLEMS
RT-MAC-2000-04, agosto 2000, 20 pp.

Telba Zalkind Irony, Marcelo Lauretto, Carlos Alberto de Bragança Pereira and Julio Michael Stern
A WEIBULL WEAROUT TEST: FULL BAYESIAN APPROACH
RT-MAC-2000-05, agosto 2000, 18 pp.

Carlos Alberto de Bragança Pereira and Julio Michael Stern
INTRINSIC REGULARIZATION IN MODEL SELECTION USING THE FULL BAYESIAN SIGNIFICANCE TEST
RT-MAC-2000-06, outubro 2000, 18 pp.

Douglas Moreto and Markus Endler
EVALUATING COMPOSITE EVENTS USING SHARED TREES
RT-MAC-2001-01, janeiro 2001, 26 pp.

Vera Nagamura and Markus Endler
COORDINATING MOBILE AGENTS THROUGH THE BROADCAST CHANNEL
RT-MAC-2001-02, janeiro 2001, 21 pp.

Júlio Michael Stern
THE FULLY BAYESIAN SIGNIFICANCE TEST FOR THE COVARIANCE PROBLEM
RT-MAC-2001-03, fevereiro 2001, 15 pp.

Marcelo Finger and Renata Wassermann
TABLEAUX FOR APPROXIMATE REASONING
RT- MAC-2001-04, março 2001, 22 pp.

Julio Michael Stern
FULL BAYESIAN SIGNIFICANCE TESTS FOR MULTIVARIATE NORMAL STRUCTURE MODELS
RT-MAC-2001-05, junho 2001, 20 pp.

Paulo Sérgio Naddeo Dias Lopes and Hernán Astudillo
VIEWPOINTS IN REQUIREMENTS ENGINEERING
RT-MAC-2001-06, julho 2001, 19 pp.

Fabio Kon
O SOFTWARE ABERTO E A QUESTÃO SOCIAL
RT- MAC-2001-07, setembro 2001, 15 pp.

Isabel Cristina Italiano, João Eduardo Ferreira and Osvaldo Kotaro Takai
ASPECTOS CONCEITUAIS EM DATA WAREHOUSE
RT – MAC-2001-08, setembro 2001, 65 pp.