UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA – CCT

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

DISSERTAÇÃO DE MESTRADO

ATUALIZAÇÃO EFICIENTE DE VISÕES MATERIALIZADAS EM DATA WAREHOUSING

Débora R. Arnaud L. Formiga

CAMPINA GRANDE, Agosto de 99

Débora R. Arnaud L. Formiga

Atualização Eficiente de Visões Materializadas em Data Warehousing

Dissertação submetida ao Curso de Mestrado em Informática do Centro de Ciências e Tecnologia da Universidade Federal da Paraíba, como requisito parcial para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Banco de Dados

Marcus Costa Sampaio
Orientador

Campina Grande, Agosto de 99

F725A

Formiga, Débora R. Arnaud Lima

Atualização Eficiente de Visões Materializadas em Data Warehousing

Dissertação de Mestrado, Universidade Federal da Paraíba, CCT, COPIN, Campina Grande, PB, 1999

Orientador: Marcus Costa Sampaio

1. Banco de Dados

2. Data Warehousing

3. Visões Materializadas

CDU – 681.3.07B

F725a Formiga, Debora R. Arnaud L.
Atualizacao eficiente de visoes materializadas em data warehousing / Debora R. Arnaud L. Fomiga. - Campina Grande, 1999.
145 f.

Dissertaca (Mestrado em Informatica) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia.

1. Banco de Dados 2. Data Warehousing 3. Visoes Materializadas 4. Dissertacao - Informatica I. Sampaio, Marcus Costa II. Universidade Federal da Paraiba - Campina Grande (PB)

CDU 004.65(043)

ATUALIZAÇÃO EFICIENTE DE VISÕES MATERIALIZADAS EM DATA WAREHOUSING

DÉBORA REINALDO ARNAUD LIMA FORMIGA

DISSERTAÇÃO APROVADA EM 25.08.1999

PROF. MARCUS COSTA SAMPAIO, Dr.
Orientador

PROF. ULRICH SCHIEL, Ph.D
Examinador

PROFª VÂNIA MARIA PONTE VIDAL, Drª
Examinadora

PROF. GUSTAVO HENRIQUE M. B. MOTA, M.Sc
Examinador

CAMPINA GRANDE – PB

Resumo

Um *Data Warehouse* (DW) é um grande repositório de dados de apoio a decisões gerenciais, "alimentado" por fontes de informação diversas, distribuídas e muitas vezes heterogêneas, que são os sistemas operacionais ou de produção das organizações. Uma característica importante de um DW é o fato de ser temporal, o que o torna muito volumoso. Um típico relatório extraído de um DW, chamado de relatório OLAP (*On-Line Analytical Processing*) apresenta séries temporais de informações sumariadas por diferentes critérios de agregação (dimensões do negócio). Para que o custo de processamento de relatórios OLAP seja aceitável, adota-se a abordagem chamada de *Visões Materializadas*, que são cálculos pré-computados, fisicamente armazenados no DW. É usando visões materializadas que o software de gerência de DW pode processar as consultas OLAP com bom desempenho.

Visões materializadas são o tema central desta dissertação, mais especificamente a problemática de sua atualização. Em intervalos regulares, o DW, incluindo as visões materializadas, precisa ser atualizado, para nele incorporar as modificações ocorridas nas fontes. Este processo pode ser extremamente custoso, exigindo um grande esforço de pesquisa, no sentido da melhoria do desempenho dos algoritmos de atualização de DWs. Dentro deste contexto, apre-

sentamos um algoritmo para a atualização eficiente de visões materializadas em DWs.

Abstract

A Data Warehouse (DW) is a great data storage toll and support to the management decisions that are "fed" by diverse information sources that are distributed. These constitute the operational systems or production systems of the organizations. One of the important characteristics of a DW is that it is temporal, wich makes it of high volume. A typical report from a DW, called OLAP (Online Analytical Processing) report presents temporal series of summarized information using different grouping criteria (business dimensions). In order to maintain the costs of OLAP acceptable, an approach called *Materialized Views* was adopted. This approach store calculated summaries physically in the DW. By using these materialized views, the managing software is able to process the OLAP consultation with a good or acceptable performance. Materialized views are the central topic of this research, mainly, the problem of updating. In regular intervals the DW, with the materialized views included, need to be uptaded so that the changes in fonts can be incorporated. This can be a costly process, requiring great effort in research, in order to improve the performance of updating algorithms of the DWs. In this context, we present an algorithm for the efficient updating of materialized views in DWs.

Agradecimentos

A DEUS por ter me dado a coragem e a fé para iniciar uma nova etapa de conhecimentos;

À minha mãe pelo apoio irrestrito nas horas em que quase pensei em desistir motivada pelo cansaço;

Ao meu marido que só soube me dar amor e compreensão quando muitas vezes deixei de estar ao seu lado para trabalhar neste projeto;

Aos meus amigos e irmãos pelo carinho e força que tanto me impulsionaram a chegar até aqui;

Ao meu orientador pela compreensão, pelo respeito às minhas idéias e minhas limitações e por tantos conhecimentos passados ao longo desses anos;

A Aninha e Vera, do DSC-COPIN, que foram muito atenciosas e prestativas;

À CAPES, pelo apoio financeiro, através de Bolsa de Estudos.

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Sumário

Capítulo 1 Introdução

O ponto central desta dissertação é um algoritmo para atualização eficiente e incremental de visões materializadas em bancos de dados de apoio à decisão. Nesta introdução, fazemos considerações sobre bancos de dados de apoio à decisão e sua problemática, damos a motivação e os objetivos da dissertação e apresentamos a estrutura da mesma, descrevendo sucintamente cada capítulo.

1.1 Data Warehouse e Visões Materializadas

Um *Data Warehouse* (DW) é um grande repositório de dados gerenciais derivados de bases de dados operacionais ou de produção (fontes), espalhadas por uma empresa. Transcendendo um DW, um ambiente de *Data Warehousing* (DWing) tem múltiplas funcionalidades, como (1) coletar os dados das bases operacionais; (2) integrá-los no DW segundo um modelo lógico apro-

priado; e (3) a partir do DW, fornecer informações aos *decision makers*, através de ferramentas de fácil uso.

As informações armazenadas em um DW são principalmente agregações de dados operacionais, ao longo de um período de tempo. Estas características são essenciais aos gerentes que desejam analisar dados e descobrir tendências e anomalias nos negócios de sua empresa.

Consultas com alto nível de agregação das informações, em que os usuários podem analisar tendências de negócios ao longo do tempo, são chamadas de aplicações OLAP (*On-Line Analytical Processing*) [22]. Os grandes DWs para aplicações OLAP podem ter de centenas de gigabytes até terabytes de tamanho. Consultas OLAP são em geral complexas, acessando milhares de registros. Como os DWs são freqüentemente relacionais, e dado o seu gigantismo, sérios problemas de desempenho podem advir se os DWs forem construídos à maneira dos bancos de dados (BDs) operacionais. A discussão sobre as vantagens e inconveniências dos modelos de dados específicos para DWs está fora do escopo deste trabalho.

Uma estratégia interessante para minimizar os problemas de desempenho de aplicações OLAP é a utilização de *visões materializadas* (VMs)[18]. Como a expressão sugere, trata-se de definir visões sobre os dados de um DW; porém, ao contrário das visões em BDs operacionais, elas existem fisicamente no DW. Visões materializadas armazenam níveis mais altos de agregação de dados em relação aos dados básicos do DW. Se uma visão materializada é apropriadamente utilizada numa consulta, diminui significativamente o número de registros a serem lidos no processamento desta. Além disso, pode-se esperar uma redução importante nos cálculos a serem realizados pela consulta (funções de agregação, principalmente), uma vez que muitos deles estão pré-computados na(s) visão(ões) materializada(s).

Apesar do processo de atualização de um DW, e consequentemente de suas VMs, acontecer geralmente uma vez por dia, na maioria dos casos à noite, e em regime *off-line*, é necessário que se gaste o menor tempo possível nesse processo, pois o DW deve permanecer o maior tempo possível disponível a seus usuários. Para dar uma idéia da importância de tornar o DW quase sempre disponível, imagine um DW a nível mundial, com

os diversos fusos horários. Assim, se em uma região o horário de consultas OLAP não for de pico, em uma outra poderia sê-lo.

Observe-se, então, que um DW não se apresenta atualizado *stricto sensu* aos seus usuários; tal fato, antes de poder ser considerado um problema, pode ser uma necessidade, uma vez que é desejável que um DW tenha *consistência temporal*, ou seja, durante uma sessão OLAP com interação usuário-sistema (sessão geralmente muito longa), o DW não deve mudar de estado.

Com relação aos modelos de dados para DW, a tecnologia relacional, devido ao seu alto grau de maturidade e disseminação, vem se impondo também neste campo. Entretanto, o projeto de um DW relacional é bastante diferente do projeto de um banco de dados relacional tradicional. Ele requer um modelo que possa destacar as diversas dimensões e os fatos de um negócio, ou seja, o modelo deve ser multidimensional.

Como mencionado anteriormente, as VMs são de grande importância em um projeto de DW, pois sua materialização pode reduzir bastante o tempo de resposta das consultas dos usuários, principalmente em se tratando de consultas conhecidas previamente. A atualização periódica de um DW deve ocorrer em um

breve intervalo de tempo, de forma que os usuários não sejam prejudicados por sua indisponibilidade durante o processo de atualização. Porém, a atualização eficiente de um DW é uma tarefa difícil, devido ao seu grande volume de dados, fontes de dados situadas em locais remotos, inconsistências entre as fontes, etc.

Para agilizar o processo de atualização do DW, pode-se considerar apenas os dados dos sistemas de produção mais recentes ou que foram adicionados após a última atualização do DW, ou seja, ao invés de utilizar todos os dados, praticamente uma nova carga do DW, podemos filtrar apenas os novos dados das fontes de informação. Com isso, o tempo de atualização do DW pode ter uma redução bastante significativa.

Infelizmente, fazer atualização incremental de um DW é uma tarefa bastante complexa. Enumeremos alguns dos problemas: (p1) como saber quais os dados das fontes que mudaram?; (p2) como filtrar informações relevantes ao DW e às visões materializadas?; e (p3) como decompor o processo de atualização em subprocessos, de modo a explorar o paralelismo?

1.2 Objetivos da Dissertação

O objetivo desta dissertação é apresentar um algoritmo para a atualização eficiente de visões materializadas em DWs, cujas idéias principais residem na filtragem de informações relevantes às visões, e na exploração de processamento paralelo.

1.3 Contribuição da Dissertação

Outros algoritmos existentes na literatura [3],[8],[9],[14],[17],[18],[28], com o propósito de atualizar um DW e suas VMs, ou não consideram os dados mais recentes das fontes (ou não são incrementais), ou são incrementais, porém não selecionam as informações mais relevantes à atualização do DW. Nossa proposta é atualizar o DW e suas VMs a partir dos novos dados inseridos nas fontes de informação, considerando dentre esses dados aqueles que podem afetar o estado das visões.

1.4 Estrutura e Conteúdo da Dissertação

Esta dissertação é constituída de sete capítulos, incluindo esta introdução, e um anexo. Descrevêmo-los sucintamente, a seguir.

O Capítulo 2 é uma breve introdução a DW e DWing, em que definimos sua terminologia básica, e as diferenças entre aplicações OLAP e OLTP. Discorremos também sobre análse multidimensional, o ambiente e a arquitetura funcional de um DWing, os elementos de projeto de um DW, e finalmente sobre o processo de carga de dados em um DW.

No Capítulo 3, enfocamos as visões materializadas, e tecemos um comentário sobre a problemática da atualização de um DW.

No Capítulo 4 apresentamos alguns trabalhos relacionados à esta dissertação e que contribuíram para maturação das idéias presentes nesta dissertação.

O Capítulo 5 é o principal da dissertação, apresentando o algoritmo *Atualiza_DW*, para a atualização eficiente de DWs. O algoritmo permite manter as tabelas do DW e suas VMs, procurando fazê-lo de forma otimizada, a fim de aumentar a disponibilidade do DW.

Concluímos a dissertação (Capítulo 6) com as conclusões e algumas considerações gerais sobre o trabalho e sugestões de trabalhos futuros.

No anexo à dissertação, encontram-se o pseudocódigo do algoritmo *Atualiza_DW* e o script de criação do DW de testes do algoritmo.

Capítulo

2 Conceitos Básicos de DWing

Data Warehouse é uma das áreas de sistemas de informação de grande interesse atualmente. Neste capítulo, definimos a terminologia básica da área e mostramos que um DW é parte de uma estrutura maior, que inclui bancos de dados operacionais, sistemas de arquivos, software de extração e integração de dados, e ferramentas de apoio à decisão.

2.1 Data Warehousing e Data Warehouse

Desde o advento da computação comercial nos anos 50, os analistas de negócios têm procurado formas de prover a informação de que eles precisam, de modo a melhor controlar seus negócios. Por décadas, os sistemas de gerência de informações eram pouco mais que enormes relatórios de várias atividades corporativas. Nos anos 60, o advento dos sistemas de gerência de bancos de dados (SGBDs) permitiu a centralização dos dados

corporativos, facilitando consideravelmente o acesso às informações. Seria então o surgimento dos BDs relacionais.

A complicação do problema foi o fato das companhias não terem realmente um armazenamento de dados centralizado. Ao contrário, elas tinham bancos de dados diferentes, gerenciados por diferentes SGBDs, e apoiando múltiplas aplicações: inventário, recursos humanos, etc. Quando os gerentes precisavam de respostas baseadas em dados espalhados nesses múltiplos bancos de dados, eles freqüentemente ficavam desapontados, visto que era bastante difícil obter as respostas de forma eficiente e funcional.

A necessidade de integração de bancos de dados múltiplos, distribuídos e heterogêneos e outras fontes de informação tem levado a um enorme esforço de pesquisa nos últimos anos [3],[8],[9],[14],[17],[18],[28]. Há duas abordagens para o problema de integração de dados.

A primeira abordagem de integração é chamada *Mediador*, e refere-se aos módulos de software que decompõem consultas e combinam seus resultados. Apesar dessa abordagem garantir informação em tempo-real, o processamento de consultas é in-

exoravelmente ineficiente e demorado, se as fontes de informação são muitas e dispersas.

A segunda abordagem e alternativa ao enfoque *Mediador* é a abordagem antecipada para integração de dados, chamada *Data Warehousing*[1]. Sua idéia básica é "disponibilizar" o resumo das informações de forma centralizada para que os analistas de negócio possam tomar decisões sem ter que levar em conta as diferentes fontes de informação.

O repositório centralizado de informações gerenciais constitui o *Data Warehouse*, que é mantido por aplicativos que extraem periodicamente dados dos BDs de produção, resume-os e integra-os no DW.

Um DWing, constitui-se, portanto, em um ambiente que envolve um conjunto de etapas de extração, tradução, filtragem e integração de dados, além de um conjunto de ferramentas para a realização de consultas sobre o seu repositório de dados. Uma descrição mais detalhada do ambiente de Dwing será apresentada na sessão 2.4.

---

1 Armazenagem de Dados, em português. Preferimos manter a expressão em inglês, já consagrada na literatura sobre BDs.

Como os relatórios gerenciais freqüentemente exibem tendências de um negócio ao longo de um período de tempo, uma importante característica dos DWs é a manutenção de dados históricos ou temporais. Outra importante característica dos DWs vem do fato de que as informações dos relatórios gerenciais são agregadas ou sumariadas em diferentes níveis. Por esta razão, os dados de um DW não devem ser uma mera cópia dos "dados de produção", mas devem ser *derivados* deles, com a derivação consistindo principalmente na sua agregação em diferentes níveis.

Um DW é voltado para aplicações OLAP, cujos requisitos funcionais e de desempenho são diferentes dos das aplicações tradicionais de processamento transacional *on-line* (OLTP – *On-Line Transactional Processing*).

No próximo tópico discorremos mais sobre aplicações OLAP e OLTP.

2.2 DWs x BDs Convencionais

Aplicações transacionais tradicionais (ou transações) e aplicações de DWing (ou aplicações OLAP) são pólos a parte em

seus requisitos de projeto e características operacionais. É importante entender essas diferenças para evitar a armadilha de projetar um Data Warehousing como se ele fosse uma aplicação de processamento de transação on-line (OLTP).

Um transação geralmente é de curta duração, acessa poucos objetos do BD, e seu tempo de resposta pode ser medido em segundos. Diferentemente, uma aplicação OLAP tem acesso a muitos objetos do DW para obter as respostas das consultas. Além disso, o tempo de resposta pode ser medido em minutos ou horas para essas consultas, e mesmo um número reduzido de usuários pode sobrecarregar uma máquina realmente potente.

Os BDs operacionais tendem a ser bem menores que os DWs. A razão para isto é que, embora os dados de um DW sejam agregados comparativamente aos dados de um BD convencional, os dados históricos que devem ser mantidos em um DW são um enorme consumidor de espaço. Não se pode deixar de levar também em conta que os dados de um DW são sumariados em diferentes níveis (visões materializadas): esta "explosão" de agregados é outro fator de crescimento do volume de um DW. Assim, um BD convencional de 50 gigabytes poderia ser consid-

erado muito grande enquanto que um DW de 50 gigabytes seria apenas um modesto DW!

Um DW é atualizado em "*batch*" periodicamente, em oposição à atualização contínua dos BDs operacionais. Cada atualização nas fontes dos sistemas de produção vem como parte de um fluxo de entrada que pode ser processado e adicionado ao DW. Tipicamente, isto é feito cada noite ou uma vez por mês, embora outros períodos (tais como quinzenalmente) são também usados.

Terminamos esta seção com a Tabela 1, que contrasta as aplicações OLAP com as aplicações OLTP. A primeira coluna mostra as características utilizadas para diferenciar os dois tipos de aplicações. As colunas seguintes representam os respectivos valores correspondente a cada tipo de aplicação e característica.

Característica	OLAP	OLTP
Objetivo	Tomada de Decisão	Controle Operacional
Operação Típica	Análise	Atualização de Dados
Complexidade das Operações	Grande	Pequena
Nível de Agregação dos Dados	Alto	O mais baixo
Dados Históricos	Sim	Não
Freqüência das Operações	Moderada	Alta
Usuário	Gestores de Negócio	Pessoal Operacional
Sistemas	Sistemas de Apoio à Decisão	Sistemas de Produção

Tabela 1 Características OLAP x OLTP

2.3 Análise Multidimensional

A tomada de decisão por um gerente de uma organização é muitas vezes baseada na *análise multidimensional.* Uma *dimensão* representa uma agregação de dados numéricos. A essência da análise multidimensional consiste em fazer-se comparações, ao longo do tempo, de dados sumariados ou agregados por diferentes dimensões, a fim de se descobrir tendências do negócio da organização. Tendências são geralmente comparações de dados em um certo nível de agregação, através do tempo.

O usuário pode querer mais detalhes ou menos detalhes de um relatório exibindo tendências de um negócio, ou pode ainda querer combinar relatórios logicamente interligados. A capacidade de "navegar" dentro das dimensões de um negócio é um aspecto fundamental da análise multidimensional.

Dentre as operações básicas de suporte à análise multidimensional destacamos *drill down, drill up* e *drill across.*

Drill Down é a seleção iterativa de dados sumariados, onde cada iteração apresenta dados com um nível maior de detalhes.

Considere um relatório de vendas anuais, inicialmente. Como resultado de operações *drill down*, o relatório passa a ser de vendas mensais, em seguida de vendas semanais, e assim por diante. Neste exemplo, as operações *drill down* se realizam ao longo da dimensão tempo, da maior granularidade de tempo para a menor.

A operação *drill up* "navega" no sentido oposto ao do da operação *drill down*. Seja um relatório de vendas diárias, inicialmente. Como resultado de operações *drill up*, o relatório para a ser de vendas semanais, em seguida de vendas mensais, e assim por diante. Neste exemplo, as operações *drill up* se realizam ao longo da dimensão tempo, da menor granularidade de tempo para a maior.

A operação que combina relatórios diferentes contendo dimensões comuns em um simples relatório é chamada *drill across*. A título de ilustração, pode-se desejar combinar dois relatórios logicamente relacionados, relatório de vendas de produtos e relatório de distribuição dos mesmos produtos, para decidir-se por aumentar ou diminuir o ritmo de fabricação dos produtos.

Bancos de dados de suporte à análise multidimensional são chamados de bancos de dados multidimensionais. Os DWs devem, obviamente, ser bancos de dados multidimensionais. A modelagem de bancos de dados multidimensionais é o assunto da próxima seção.

2.3.1 Modelagem de Bancos de Dados Multidimensionais

A modelagem multidimensional é uma técnica de projeto bastante diferente da modelagem relacional pura, pois os dados são representados, na maioria das vezes, de forma cúbica. Dessa forma, alguns conceitos existentes nos projetos de bancos de dados tradicionais, principalmente no que se refere a parte relacional, foram tratados de forma bastante diferente da tradicional. Neste tópico, iremos apresentar as duas abordagens (relacional e multidimensional) de forma a esclarecer e apresentar suas principais características.

2.3.1.1 Modelo Relacional para Aplicações OLTP

O projeto de BDs multidimensionais é bastante diferente do projeto de BDs de produção para aplicações OLTP, pois os BDs OLTP são projetados para facilitar a atualização dos seus dados, e sua modelagem visa evitar a redundância de dados, para

que as transações possam modificar os dados rapidamente. Como os BDs OLTP são geralmente relacionais, seus esquemas de dados consistem de diversas tabelas relacionadas logicamente, como mostra a Figura 1.

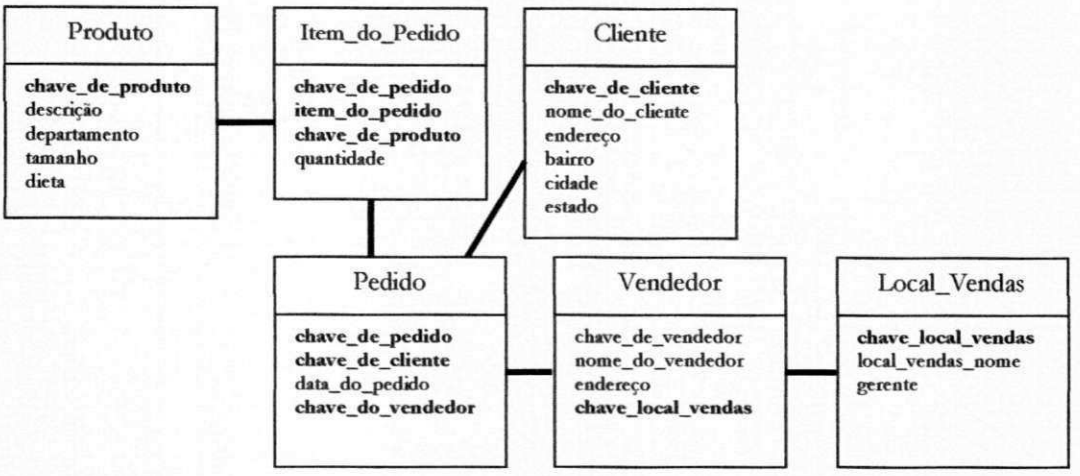

Figura 1 Esquema Relacional de um BD OLTP

Além disso, as tabelas são normalizadas, ou seja, contêm somente as redundâncias absolutamente necessárias à consistência entre elas. Por exemplo, na tabela `Item_do_Pedido`, o item de dados `Chave_de_Pedido` (chave externa[2]) é repetido do item de dados da tabela `Pedido` (chave primária[3]). Desta forma, a tabela `Item_de_Pedido` contém todos os itens de determinado

[2] Chave externa é o item de dados que identifica a ligação lógica de uma tabela com outra
[3] Chave primária é o item de dados que identifica unicamente uma linha da tabela

pedido da tabela `Pedido`, através da ligação lógica `chave_primária-chave_externa`.

Um outro fato relacionado aos esquemas de BDs OLTP é que eles não são tão simples, necessitando geralmente de uma análise mais apurada para sua completa compreensão.

2.3.1.2 Modelo Multidimensional Puro

Visões multidimensionais (ver Figura 2, pág. 33) de dados permitem aos usuários verem os detalhes e agregados de medidas de negócios em um DWing pelos atributos dessas medidas. Medidas ou fatos são os números que quantificam os valores do negócio, tais como vendas em dólar, vendas unitárias, ou o número de empregados. O nível mais alto de detalhe é o nível mínimo de agregação apropriado, como por exemplo, dia, semana ou mês.

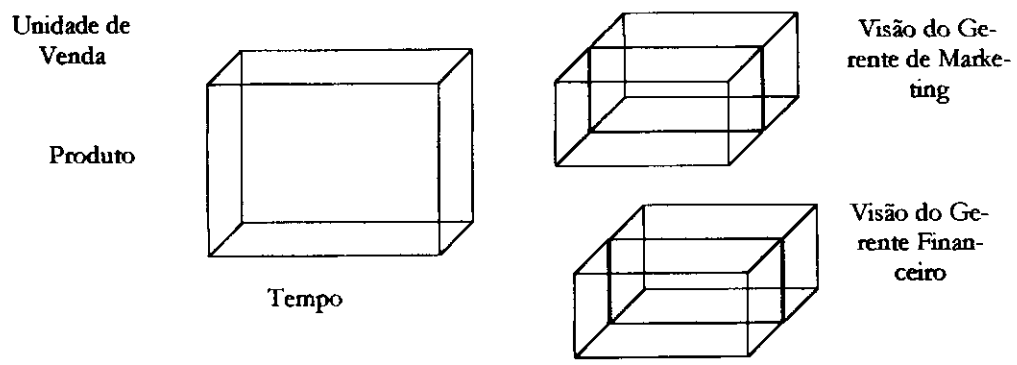

Figura 2 - Visões Multidimensionais

Há duas razões principais para a adoção de um BD Multidimensional: velocidade e operadores especiais para fazer análise multidimensional. Entretanto, BDs relacionais em conjunção com ferramentas de análise multidimensional podem suprir a velocidade e os operadores necessários sem sacrificar a generalidade do modelo relacional.

As matrizes bidimensionais ou planilhas são um bom exemplo da modelagem de BDs multidimensionais. Ao compararmos as planilhas podemos notar que é muito mais compreensível, com os valores das dimensões bem destacados e organizados.

Infelizmente, na prática o número de visões pode ser bastante alto. Visualizar 3 dimensões, forma chamada de cubo, é muito difícil. Acima de 3 dimensões, forma chamada de Array Multidimensional, é praticamente impossível. Por outro lado, o modelo relacional que tem uma tecnologia bastante amadurecida em termos de desempenho e confiabilidade, e bastante difundida no mercado mundial, não podia ser simplesmente ignorada. A solução encontrada foi representar arrays multidimensionais por

meio de tabelas relacionais. O tal esquema é chamado de esquema relacional com estrutura em estrela.

2.3.1.3 Modelo Relacional em Estrela

Os requisitos de esquemas de dados para DWs são diferentes dos de esquemas de dados para BDs OLTP, pois as dimensões dos primeiros devem estar bem destacadas para serem facilmente compreendidas a partir de uma simples análise.

Todo esquema relacional consiste em um conjunto de tabelas, assim o esquema relacional com estrutura em estrela não poderia ser diferente. Entretanto, notamos nestes esquemas dois tipos de tabelas: uma tabela *central* de medidas numéricas ou *fatos*, chamada *tabela de fatos*, em torno da qual existem *tabelas de dimensão*, representando as dimensões de um negócio. Existe, portanto, uma ligação lógica entre as tabelas de fatos e de dimensões, através das chaves das dimensões, que passam a compor a chave primária da tabela de fatos. As tabelas de dimensão têm geralmente cardinalidade pequena, quando comparadas com a da tabela de fatos. A figura 3 mostra um esquema em estrela para um DW de vendas.

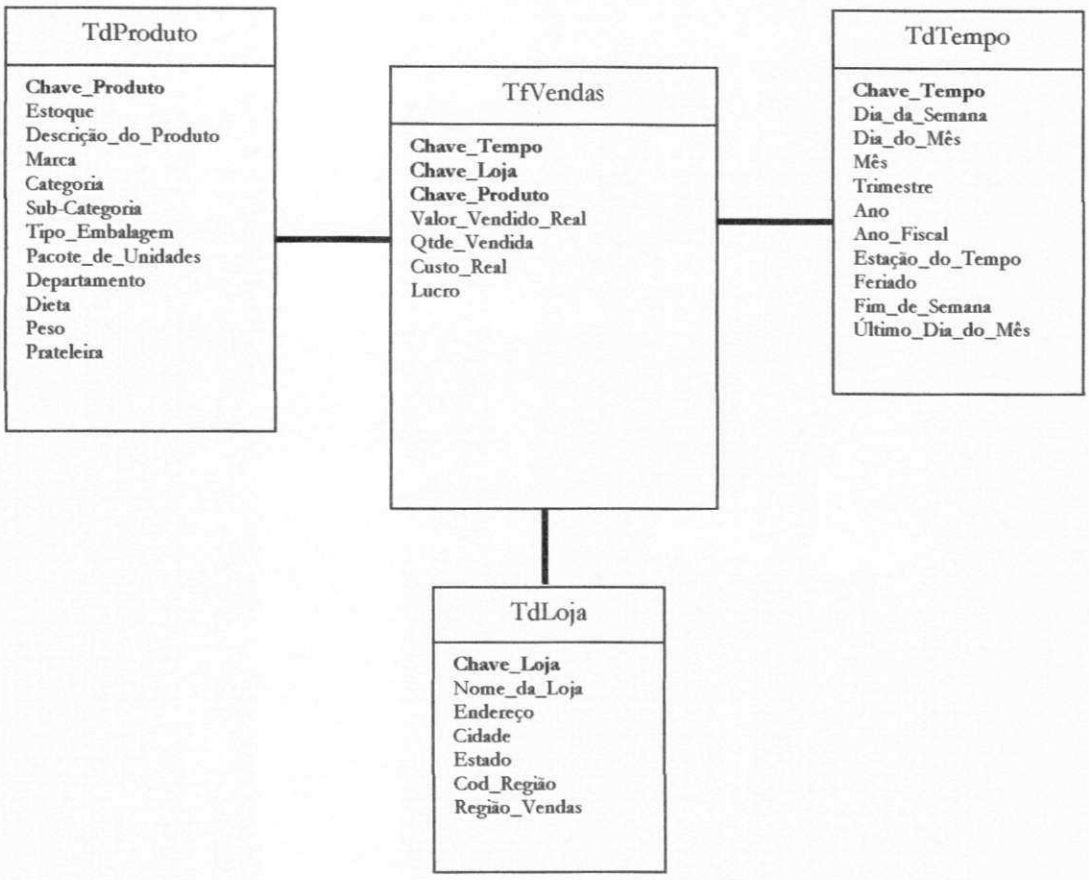

Figura 3 Esquema Relacional com Estrutura em Estrela para um DW de vendas

Na Figura 3, podemos ver a tabela de fatos `TfVendas`, e as tabelas de dimensão `TdProduto`, `TdLoja` e `TdTempo`.

Além do esquema acima, chamado DW primário ou básico, um DW pode conter DWs secundários ou agregados, ou ainda, visões materializadas. Uma abordagem conceitual sobre visões materializadas será mostrada no capítulo 3, pois VMs constituem o foco principal desta dissertação, bem como a problemática de sua atualização.

A seguir, damos mais detalhes sobre tabelas de fatos e de dimensão.

2.3.1.3.1 Tabela de Fatos

Cada linha da tabela de fatos representa as medidas numéricas associadas ao negócio da empresa, ou fatos, para uma combinação dos valores das chaves primárias das tabelas de dimensão. Como mencionado anteriormente, o conjunto das chaves primárias das tabelas de dimensão compõem a chave primária da tabela de fatos, no exemplo `Chave_Tempo`, `Chave_Loja`, `Chave_Produto`.

Uma importante consideração sobre tabelas de fatos é que, geralmente, elas são temporais, guardando históricos de fatos de 5, 10 anos ou mais, apesar de ser mais freqüente guardar fatos do dia-a-dia. Desta forma, elas são absolutamente essenciais à geração , de maneira simples, de relatórios que exibem tendências de um negócio. Dado o seu aspecto temporal, a cardinalidade das tabelas de fatos geralmente é enorme, e ela é determinante do tamanho de um DW. Impõem-se então, que as tabelas de fatos sejam normalizadas, como as tabelas de BDs OLTPs, para evitar redundância desnecessária.

Além da chave primária, os demais atributos de uma tabela de fatos contêm, a rigor, valores aditivos, semi-aditivos ou não-aditivos:

- Atributos aditivos: seus valores podem ser somados ao longo de todas as dimensões. Exemplos: total de um produto por região, total do produto por loja, total do produto por um período de tempo, etc. Na figura 3, os atributos `valor_vendido_real` e `custo_real` são aditivos.

- Atributos semi-aditivos: a soma de seus valores ao longo de uma dimensão não tem significado em si, porém pode ser útil a outras operações aditivas. Exemplo: saldos de uma conta não devem ser somados ao longo do tempo, entretanto, a média dos saldos num certo período de tempo é uma informação bastante útil.

- Atributos não-aditivos: não podem ser somados, ou sua soma não faz sentido. Geralmente, valores percentuais são não-aditivos. Na figura 3, o atributo percentagem de lucro `((valor_vendido_real / custo_real) / valor_vendido_real)` é não-aditivo.

Outro importante aspecto do projeto de um esquema em estrela é a questão da granularidade da tabela de fatos. A granularidade diz respeito ao nível de detalhe dos fatos nela contidos. Quanto mais detalhe, mais baixo o nível de granularidade, e mais volumosa é a tabela de fatos. Quanto menos detalhe, mais alto o nível de granularidade, e menos volumosa, comparativamente, é a tabela de fatos.

2.3.1.3.2 Tabelas de Dimensão

As tabelas de dimensão devem conter atributos que sejam significativos para a análise de um negócio, que permitem restringir o acesso à tabela de fatos, e ainda, que sirvam de critério de agregação dos registros da tabela de fatos. Não sendo temporais, elas influem minimamente no volume de um DW, e nunca se deve comprometer o projeto das tabelas de dimensão com o fim de gerar economia de espaço. Ou seja, as tabelas de dimensão são tabelas de modificações lentas ao longo do tempo, daí a denominação SCD (Slowly Change Dimension). Em conseqüência, as tabelas de dimensão não devem ser necessariamente normalizadas.

As tabelas de dimensão estão mais sujeitas às concessões de não-normalização. Existem correntes (modelagem denominada *Star Schema*) que advogam a não-normalização das tabelas de dimensão. Outras (modelagem *Snowflake Schema*) exigem a normalização das mesmas. Observe que, na tabela `TdLoja` existe o código da região (`cod_região`) e nome da loja. No nosso exemplo, adotamos a estratégia de *Star Schema*. Na modelagem *Snowflake Schema*, as tabelas de dimensão são definidas em camadas, daí o nome *flocos de neve (sic)*.

As tabelas de fatos, na maioria das vezes, são sempre normalizadas em ambas as propostas.

Além das tabelas de dimensão terem modificações lentas ao longo do tempo, há uma outra razão para que elas não devam ser completamente normalizadas. Para restringir o acesso à tabela de fatos, os usuários fazem primeiro *browsing* nas tabelas de dimensão. Segundo [14], em um dia de interação usuário-sistema, 80% das operações são *browsing* em tabelas de dimensão. Se as tabelas de dimensão fossem normalizadas, muito fragmentadas portanto, isto dificultaria em muito as operações *browsing*.

Tabelas de dimensão não normalizadas e geralmente apresentam relacionamentos hierárquicos (relacionamentos 1:N) entre seus atributos. Observe a tabela de dimensão `TdLoja`, do esquema da figura 3. Ela contem a hierarquia de atributos *loja-cidade-estado* (uma cidade pode ter muitas lojas e uma loja existe em uma única cidade; um estado tem várias cidades e uma cidade existe em único estado). Uma outra hierarquia é *loja-região_vendas-cod_região.*

A existência de hierarquias permite definir as operações *drill up* e *drill down* naturalmente ou implicitamente. Por exemplo, de vendas por loja para vendas por cidade é uma operação *drill up*, enquanto que, de vendas por estado para vendas por cidade é uma operação *drill down.* Entretanto, estas operações não se restrigem a hierarquias de atributos, podendo ser generalizadas[1].

Com relação a valores de chaves de dimensão, recomenda-se que eles não tenham nenhum significado em si (valores internos, valores substitutos ou *surrogates keys*), sua semântica sendo definida pelos valores dos demais atributos[1]. A explicação é que valores de chaves devem servir unicamente para as ligações

lógicas entre os registros das tabelas de dimensão e os da tabela de fatos.

Terminamos esta seção com um pequeno comentário sobre a dimensão Tempo, quase sempre presente em um DW.

2.3.1.3.3 A Dimensão Tempo

A representação da dimensão Tempo através de uma tabela pode parecer desnecessária, já que é possível definir um atributo do tipo Data (DD/MM/AAAA) para a tabela de fatos. Entretanto, uma data esconde muito de sua semântica. Por exemplo, 21/10/1999 representa explicitamente apenas três pontos de tempo, o dia 21, e mês de outubro e ano de 1999. Outros pontos do tempo relacionados a esta data, que poderiam ser de muito interesse dos usuários, como semana, trimestre, feriado, etc., ficam completamente obscurecidos.

Um dos requisitos fundamentais de um projeto de DW é a sua clareza. Particularmente, a informação histórica (séries temporais) deve estar bem explícita. Para isso, todos os pontos relevantes do tempo, citados anteriormente, devem ser explícitos através dos atributos da dimensão Tempo.

2.4 Ambiente de Data Warehousing

Como mencionado anteriormente, um ambiente de DWing não é simples, se comparado a um ambiente de BDs Operacionais.

Das características assinaladas na seção que trata das diferenças entre DWs e BDs convencionais (ver Tabela 1, pág. 28), é fácil concluir que as fontes de dificuldades em DWing são muito diferentes daquelas dos sistemas de produção.

Primeira dificuldade: o tamanho do DW. O que pode ser simples com 100 megabytes de dados, torna-se difícil com 100 gigabytes de dados. Como o volume de dados é muito grande, o tempo de resposta de um DW não pode ser comparado ao tempo de resposta de sistemas de produção, uma vez que eles possuem um volume muito menor de dados e o acesso é feito a poucos dados.

Segunda dificuldade: o projeto do DW. No *back-end* (etapas de extração, tradução, filtragem e integração) do DW, os bancos de dados fontes podem mudar, novas fontes podem ser adicionadas, ou fontes antigas removidas. Acrescente-se a isto o problema da integração de dados de várias fontes, bem como a

manutenção de dados históricos. No *front-end* (ferramentas OLAP) do DW, os requisitos dos usuários gerenciais são muito mais complexos, envolvendo grande volume de dados e cálculos. Assim, o projeto de um DW necessita ser orgânico, em que ele constantemente envolve novas demandas de usuários.

Em alguns casos, dada a complexidade e gigantismo de projetos de ambientes de DWing, as organizações partem para o conceito de *Data Marts*, como veremos na próxima seção.

2.4.1 Data Marts

Data Marts não constituem tema principal desta dissertação, porém tecemos um breve comentário por se tratar de um conceito ligado a DW. Eles (ver Figura 4, pág. 44) são pequenos DWs e são construídos para enfocar necessidades particulares. Por exemplo, o Departamento de Marketing de uma empresa poderia apenas se preocupar com dados sobre clientes, produtos e vendas, e ter este subconjunto do DW como o tema do *Data Mart* (enfoque *top-down*).

Em alguns casos, o tamanho do empreendimento pode resultar em um DW muito grande e complexo, que pode requerer anos para sua construção. Então, constroem-se os *Data Marts*

primeiro, para interpretá-los mais tarde no DW (enfoque *botton-up*). A intenção é utilizar a proposta *bottom-up* para a construção do empreendimento de DW em oposição a proposta *top-down* como descrita acima.

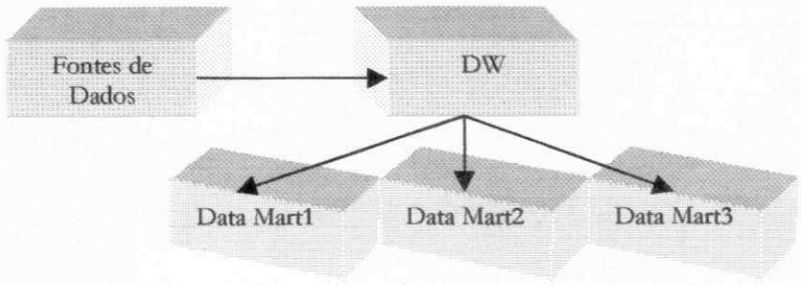

Figura 4 Enfoque *top-down* de construção de *Data Marts*

Entretanto, há um problema potencial. É que os *Data Marts* apresentam dificuldade de integração se eles não são projetados para uma futura ligação entre eles. As mesmas inconsistências que infestam um DW, construído de sistemas de produção ou outros *legacy system* surgir de *Data Marts* sem coordenação. Infelizmente, planejar *Data Marts* pode ser quase tão difícil quanto construir DWs.

2.4.2 Arquitetura

A arquitetura mais comum de um ambiente de DWing está apresentada na Figura 5:

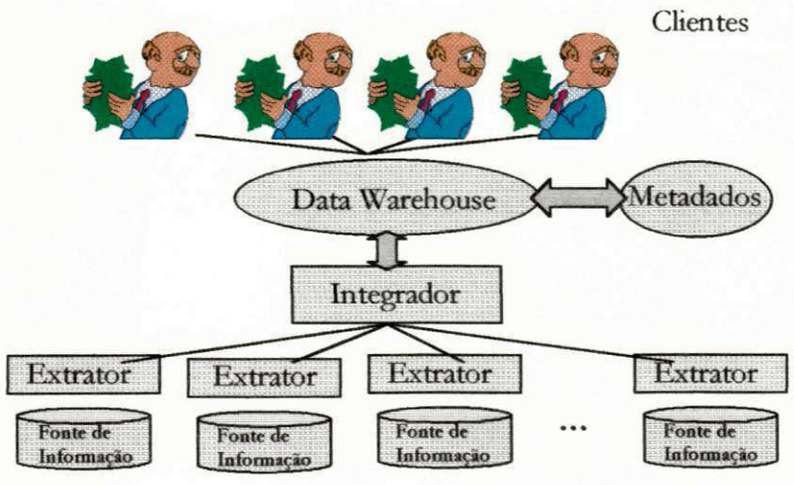

Figura 5 Arquitetura funcional de um Data Warehousing

Nela, podemos notar os seguintes elementos:

- **Fontes de Informação (BDs Operacionais)**, que são os BDs Operacionais, dados externos (extraídos da Internet, por exemplo), ou seja, todos as informações importantes e necessárias para "alimentar" o DW;

- **Extratores**, que são módulos agregados a cada fonte, responsáveis pela identificação ou detecção de mudanças nas fontes,

através de regras ativas, e pela extração dos dados, notificando estas ao DW;

- **Integrador**, responsável por receber as atualizações nos dados fontes, integrar esses dados, muitas vezes incompatíveis, criar o DW básico e as visões materializadas (DW básico constitui-se em tabelas de fatos e dimensões), manter a integração do esquema, e armazenar os dados,

- **Repositório de Metadados**, que é literalmente um "banco de dados sobre os dados", ou seja, ele armazena informações técnicas que descrevem a estrutura física dos dados, tanto das relações dos BDs Operacionais, quanto das relações da estrutura do DW e definições das VMs;

- **Data Warehouse**, que é o repositório central do ambiente de Data Warehousing. Nele estão as informações básicas (tabelas de fatos), as tabelas de dimensão, além das visões materializadas;

- **Clientes**, responsáveis por realizar consultas sobre os dados a fim de analisá-los e descobrir informações antes implícitas.

Na arquitetura apresentada em [25], existem outros módulos responsáveis pelas tarefas de detecção e transformação dos dados fontes para o DW, e atualização em tempo real do mesmo. Entretanto, como raramente é necessária contemplar a necessidade de atualização imediata, neste caso, a atualização em tempo real do DW, é até indesejável. Isto porque, na maioria dos casos, não é necessário que os dados de um DW estejam atualizados *stricto sensu* : bem ao contrário, o DW deve ter consistência temporal [14], isto é, uma consulta OLAP, se repetida várias vezes ao longo de um período de tempo (por exemplo, um dia), deve reproduzir o mesmo conjunto resposta.

2.5 Elementos de Projeto de um DW

Um projeto de um DW se inicia como os projetos de bancos de dados convencionais. Uma área de negócio é devidamente escolhida, baseada nas prioridades emergentes da organização. As áreas usuais para condução de projetos iniciais de DW são clientes, finanças, vendas, produção, etc., sempre associadas às prioridades reais de negócio da organização. Escolhida a área de negócio, parte-se para definir aquele(s) que será(ão) o(s) proc-

esso(s) alvo(s) do projeto de DW: varejo, entrega, controle de pedidos, estatísticas de venda, assinaturas, etc.

Até aí, o processo é o mesmo dos projetos de BDs convencionais. A partir desse ponto, acontecem algumas mudanças. Tomemos como base o esquema relacional para um DW de vendas mostrado na Figura 3.

A primeira etapa do processo é a determinação da *granularidade* desejada para o processo a ser controlado. Essa análise é de fundamental importância, pois define, de forma combinatória, os níveis dimensionais que serão usados para o armazenamento dos dados. Por exemplo, numa rede de supermercado que deseja controlar o comportamento de vendas de produtos por região ao longo do tempo, podemos ter várias possibilidades: a tripla *produto-loja-dia* seria uma alternativa, como também seriam possíveis outras variações, como "produto-loja-mês" (nesse caso, a variação na dimensão tempo, de dia para mês) ou "produto-região-dia" (aqui com variação na dimensão geografia, de loja para região). Os fatores que definem a escolha da granularidade estão relacionados com o volume de dados a ser mantido e com o processamento necessário para produzi-lo.

O volume pode ser estimado através de uma média de combinação entre as dimensões. Por exemplo, suponha que em cada loja, na média, num determinado dia, são vendidos 10.000 produtos. Se tivermos 200 lojas e desejarmos armazenar os dados por dois anos (365x2), teríamos algo em torno de 10.000x200x730, ou 1,46 bilhões de registros. Se considerarmos que teremos três chaves (chave_produto, chave_loja, chave_tempo, cada qual com 5 bytes) e três valores numéricos (cada qual com 4 bytes), teremos 1,46 bilhões de registros vezes 27 bytes (15+12), o que levaria a um montante de 39,4 gigabytes, sem contar gastos com índices e outras tabelas acessórias.

Note que estamos considerando somente uma tabela de fatos. Perceba ainda, que qualquer acréscimo em uma das variáveis dimensionais pode elevar o volume a patamares elevados1. Aqui caberia, entre outros, um questionamento sobre a dimensão temporal. Dois anos é suficiente para se perceberem os padrões escondidos de comportamento de vendas de produtos em lojas, que as ferramentas de apoio a decisão se propõem revelar?

O outro aspecto importante na definição da granularidade é o trabalho para se chegar a ela. Note que, no nosso exemplo, a granularidade foi definida em termos de produto-loja-dia, o que nos faz pressupor que temos um processamento diário de consolidação para transformarmos os nossos registros (transações) de vendas (com os seus itens) em totais por produto e por loja. Isso pode ser feito automaticamente nos vários caixas automáticos (PDVs – Pontos de Vendas) e enviados na janela da noite para uma central de atualização do DW.

Após a definição da granularidade, o passo seguinte é o detalhamento das dimensões. Note que, as dimensões já foram discutidas, até porque a especificação da granularidade, feita no passo anterior, depende delas. No caso, as dimensões definidas foram produto, loja e tempo. Observe que, as dimensões geografia e tempo (loja e tempo, no exemplo) quase sempre estarão presentes nos projetos de DW. O importante nessa etapa é a hierarquia das dimensões e a definição dos atributos restantes de cada dimensão. Por exemplo, na dimensão tempo, podem existir uma ou mais hierarquias como "ano→mês→dia", ou "estação-do-ano→dia". No caso da dimensão produto, podem existir

dissemos, serão normalizadas; com exceção de casos em que a granularidade é muito alta e um registro Fato poderá conter vários valores. Isso poderá acontecer em BDs com medições temporais (valores de 5 em 5 minutos, por exemplo) ou equivalentes. Nesse caso, os valores são armazenados em array e a dificuldade de manipulação dos valores é de certa forma transferida para os programas na estação cliente, pois haverá impedimento do SQL de tratar valores agregados não normalizados.

Observe ainda que, no nosso exemplo, a trinca de chaves da tabela de fatos oferece unicidade às linhas, ou seja, pode-se ter o mesmo produto vendido na mesma loja, no mesmo dia, visto que existe uma totalização por dia, entretanto, esse registro não é capturado porque a granularidade é dia. Uma alternativa, caso a granularidade fosse em nível de transação, seria definir como chave primária a concatenação do número-pedido e o número-item e, dessa forma, o tamanho em bytes do registro da tabela de fatos cresceria algo em torno de 8 bytes, e o tamanho da tabela cresceria muito mais.

2.6 Carga de um Data Warehouse

O processo de migração de dados do banco de dados fonte para o DW é composto de duas operações distintas. A primeira é o problema da carga inicial e em seguida atualização periódica do DW.

Para inicializar um DW, o primeiro passo é extrair os dados desejados dos bancos de dados fontes. Isto pode ser feito através da criação de um programa personalizado para tal tarefa ou utilizando uma ferramenta de extração de dados já existente. O próximo passo é detectar e corrigir os problemas de qualidade.

Os dados devem então ser mapeados para a estrutura de dados do DW. O projeto de um DW irá remover as diferenças entre os nomes dos dados, dos atributos e dos relacionamentos, portanto, é de responsabilidade do projetista estabelecer, cuidadosamente, as correspondências. Por exemplo, múltiplos campos com diferentes nomes irão fornecer valores para um único campo do DW.

Como parte do projeto de um DW, certos cálculos serão definidos, tais como "`preço x = preço * qtde`". Em adição, certas sumariações irão ser pré-definidas, e cálculos mais comuns

não necessitarão ser repetidos. Por exemplo, ao invés de recalcular as vendas diárias e mensais sempre que os dados forem recuperados, os totais podem ser calculados e armazenados para uma resposta mais rápida.

O processo de carga envolve alguns passos de processamento, como ordenação dos dados e realização de alguns cálculos ou agregações. A quantidade de tempo para carregar os dados pode ser bastante longa, mesmo com computação paralela, visto o grande volume dos dados, o número de cálculos e da necessidade de construção de índices.

Todos esses passos precisam ser realizados antes da atualização de um DW. O que torna este problema mais complicado é que esta atualização em lote deve ser realizada em um espaço de tempo limitado e quando o DW estiver *off-line*. Alguns autores[3] defendem a reconstrução do DW cada vez que o mesmo for atualizado. Realmente, se ocorrer uma mudança na estrutura dos dados ou se o banco de dados tem uma certa quantidade de sumariações a serem recalculadas, uma reconstrução pode ser necessária. Porém, o inconveniente é deixar o DW indisponível por

aproximadamente 12 horas ou mais dependendo do tamanho do mesmo.

Portanto, é aconselhável extrair apenas as mudanças ocorridas nas fontes e aplicá-las ao DW. Porém, é uma tarefa complicada, pois extrair apenas os dados modificados não é fácil, usar apenas os dados modificados pode obscurecer problemas de qualidade (tais como referências incorretas), e atualizar o DW, incluindo a realização de cálculos necessários para mudar as agregações, é mais difícil do que carregá-lo (e o tempo pode não ser pequeno).

Apesar das dificuldades, se houver um planejamento adequado no projeto de um DW, a sua atualização incremental terá sucesso.

2.7 Conclusões

O problema da atualização periódica de DWs é o foco desta dissertação. Além disso, para melhorar o desempenho das consultas, faz-se o uso de visões materializadas ou DW secundário (ver Sessão 1.1, pág. 14). Vimos também que o proc-

esso de atualização periódica do DW não é um problema de fácil solução.

Visões Materializadas e a problemática da atualização de um DW são o objeto do próximo capítulo.

Capítulo

3 Visões Materializadas

Da análise cuidadosa das consultas OLAP, pode-se perceber que, em sua imensa maioria, elas impõem a agregação de centenas ou milhares de linhas da tabela de fatos, a um custo muito alto se essas agregações tiverem que ser feitas em tempo-real. Para contornar o problema, surgiu a idéia de se pré-computar agregados (visões materializadas), pelo menos aqueles mais freqüentemente usados pelas consultas OLAP.

Neste capítulo, abordamos as visões materializadas no contexto do DW e do projeto de um DW, e a problemática de sua atualização.

3.1 O que são Visões Materializadas (VMs)?

O tamanho das tabelas de fatos em DWs é uma enorme barreira ao grande objetivo de um projeto de DW, que é o de

permitir que as consultas OLAP aos DWs, geralmente muito complexas, possam ser processadas de maneira eficiente.

Um agregado pré-computado é uma nova tabela de fatos, derivada da tabela de fatos *básica*, e chamado também de *visão materializada*. Ele pode ter suas próprias tabelas de dimensão, chamadas de *dimensões derivadas*.

A definição dos agregados ou VMs representa a fase final da modelagem e são tabelas prontas, trabalhadas em várias dimensões e que facilitam os acessos aos dados e agilizam os processos decisórios.

Os valores agregados apresentam paradoxalmente uma solução e alguns problemas. Solução, porque constituem tabelas com cálculos pré-computados, que agilizam os processos decisórios, já que são utilizadas para responder as consultas dos usuários, geralmente em busca destes cálculos. Problemas, pois agridem os preceitos canônicos de não-redundância estabelecidos nos projetos de bancos de dados. Além disso, obviamente, gastam mais espaço, pois exigirão uma coleção de tabelas fato ou dimensão, agora dedicadas ao armazenamento de dados, digamos, "pré-cozidos". Entretanto, o seu uso pode reduzir

bastante o tempo de resposta de uma consulta ao DW, feita pelo usuário, o que pode justificar os problemas apresentados.

Os critérios para definição das VMs passam pela análise dos principais tipos de informação necessários pelo usuário e frequentemente utilizadas, e pela dificuldade de obtê-los diretamente das tabelas de fatos. Suponha o projeto de DW já discutido aqui anteriormente e constituído das seguintes tabelas:

- TdLoja (#chave_loja, nome_da_loja, endereço, cidade, estado, cod_região, região_vendas).

- TdProduto (#chave_produto, estoque, descrição_do_produto, marca, categoria, sub-categoria, tipo_embalagem, pacote_de_unidades, departamento, dieta, peso, prateleira).

- TdTempo (#chave_tempo, dia_da_semana, dia_do_mês, mês, ano, ano_fiscal, estação_do_tempo, feriado, fim_de_semana, último_dia_do_mês).

- TfVendas (#chave_loja, #chave_produto, #chave_tempo, valor_vendido_real, custo_real, lucro, qtde_vendida).

Neste caso, teríamos as seguintes possibilidades de tabelas com valores agregados, que comporiam o esquema final do DW:

• Loja – Por loja, todas as lojas e por região, num total de três alternativas.

• Tempo – Por dia, por mês, por estação, num total de três alternativas.

• Produto – Por produto, todos os produtos e por marca, total de três alternativas.

Dessa forma, o número de VMs passíveis de serem definidas seria a combinação das alternativas acima, totalizando 27. A escolha deverá ser, obviamente, por aquelas que oferecerem maior disponibilidade de informações e também o melhor coeficiente de redução. Isso significa que, quanto maior for a cardinalidade da(s) coluna(s) escolhida(s) para compor a VM, menor efeito trará a sua utilização. Em outras palavras, significa que, se as colunas escolhidas tiverem muitos valores diferentes, os registros agregados serão em grande número e poderão não satisfazer em termos de performance, quando comparados com a tabela de fatos. Comandos SQL, como o mostrado abaixo, poderão ajudar

na análise e viabilidade de se produzirem as VMs, permitindo estabelecer o histograma de seus valores.

Exemplo, verificar a distribuição dos valores agregados por loja:

```
SELECT NOME_DA_LOJA, COUNT(*)
FROM TdLOJA AS L, TfVENDAS AS V
WHERE V.CHAVE_LOJA = L.CHAVE_LOJA
GROUP BY NOME_DA_LOJA
```

Código 1 Valores Agregados por Loja

As VMs exigem alguns cuidados nas suas definições. O principal é a definição dos valores aditivos que estarão habitando os agregados, pois lembre-se de que nem todas as métricas armazenadas na tabela de fatos são aditivas em todas as dimensões (algumas são semi-aditivas ou não-aditivas). Isso significa que os atributos das VMs poderão ser diferentes dos da tabela de fatos. Outro cuidado é definir criteriosamente a precisão dos valores aditivos das VMs, que deverão ser maiores do que os usados nos respectivos valores da tabela de fatos. Caso contrário, experimentaremos *"overflow"* em operações de adição. Outro aspecto muito importante se relaciona com a definição do DW secundário (tabela de fatos agregada e tabelas de dimensão agre-

gadas) em estruturas fisicamente diferentes do DW primário. Embora esse número grande de tabelas possa parecer um complicador para o usuário, alguns fatores atenuantes, como o *Navegador de Agregados* (Aggregate Navigator), estão surgindo para criar transparência no uso dessas informações consolidadas de natureza gerencial.

Na realidade, o *navegador de agregados* é mais uma camada colocada entre a ferramenta OLAP (que solicita os dados do DW básico ou primário) e o servidor de DW. Esse navegador realiza, transparentemente, a tradução das solicitações, convertendo consultas SQLs que referem-se a tabela de fatos e dimensões do DW primário nas tabelas equivalentes do DW secundário apropriado. Além disso, essas ferramentas oferecem no seu arsenal algumas funções que permitem melhor definir as VMs (baseados em análises de histograma), e monitoram e registram o uso de cada VM, dando indicações de sua efetiva utilidade nos processos de tomada de decisão. Dentre os produtos que oferecem essas características se destacam a família MetaCube (Informix), Microstrategy e Information Advantage [23].

Uma forma de se entender melhor as VMs é através dos comandos SQL que as produzem. Por exemplo, podemos definir algumas VMs, como abaixo:

1. Vendas por loja: Visão vmVendas_por_Loja

```
INSERT INTO vmVENDAS_POR_LOJA AS
SELECT NOME_DA_LOJA AS LOJA,
SUM(VALOR_VENDIDO_REAL) AS VALOR,
SUM(CUSTO_REAL) AS CUSTO
FROM TdLOJA AS L, TfVENDAS AS V
WHERE L.CHAVE_LOJA = V.CHAVE_LOJA
GROUP BY NOME_DA_LOJA
```

Código 2 Vendas por Loja

equivale à agregação por lojas de todos os produtos, todos os dias.

2. Vendas por loja e por mês: Visão vmVendas_por_Loja_Mês

```
INSERT INTO vmVENDAS_POR_LOJA_MÊS AS
SELECT NOME_DA_LOJA AS LOJA, MÊS,
SUM(VALOR_VENDIDO_REAL) AS VALOR,
SUM(CUSTO_REAL) AS CUSTO
FROM TdLOJA AS L, TfVENDAS AS V,
TdTEMPO AS T
WHERE L.CHAVE_LOJA = V.CHAVE_LOJA AND
T.CHAVE_TEMPO = V.CHAVE_TEMPO
GROUP BY NOME_DA_LOJA, MÊS
```

Código 3 Vendas por Loja e por Mês

equivale à agregação por loja, por mês, para todos os produtos.

3. Vendas por região de venda, por mês, por categoria: Visão vm_Vendas_por_Reg_Mês_Categoria

```
INSERT INTO
vmVENDAS_POR_REG_MÊS_CATEGORIA AS
SELECT REGIÃO_VENDAS, MÊS, CATEGORIA,
SUM(VALOR_VENDIDO_REAL) AS VALOR,
SUM(CUSTO_REAL) AS CUSTO
FROM TdLOJA AS L, TfVENDAS AS V,
TdPRODUTO AS P, TdTEMPO AS T
WHERE L.CHAVE_LOJA = V.CHAVE_LOJA AND
T.CHAVE_TEMPO = V.CHAVE_TEMPO AND
P.CHAVE_PRODUTO = V.CHAVE_PRODUTO
GROUP BY REGIÃO_VENDAS, MÊS, CATEGORIA
```

Código 4 Vendas por Região, por Mês e por Categoria

equivale à agregação por região de venda, por mês e categoria de produtos.

4. Vendas por Produto no mês de Outubro de 1999 nas lojas Iguatemi: Visão vm_Vendas_por_Produto_Out_1999_Iguatemi

```
INSERT INTO
vmVENDAS_POR_PRODUTO_OUT_1999_IGUATEMI
AS
SELECT DESCRIÇÃO_DO_PRODUTO AS PRODUTO,
SUM(QTDE_VENDIDA) AS VENDAS
FROM TdLOJA AS L, TfVENDAS AS V,
TdPRODUTO AS P, TdTEMPO AS T
WHERE L.CHAVE_LOJA = V.CHAVE_LOJA AND
L.NOME_DA_LOJA LIKE
'%'||'IGUATEMI'||'%' AND T.CHAVE_TEMPO
= V.CHAVE_TEMPO AND T.MÊS = 'OUTUBRO'
AND T.ANO = '1999' AND
P.CHAVE_PRODUTO = V.CHAVE_PRODUTO
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 5 Vendas por Produto, no mês de Outubro de 1999, nas lojas Iguatemi

equivale à agregação por produto, nas lojas Iguatemi, no mês de Outubro do ano de 1999.

5. Últimas Vendas: Visão vmÚltimas_Vendas_Iguatemi-JPessoa

```
INSERT INTO vmÚLTIMAS_VENDAS_IGUATEMI-
JPESSOA AS SELECT DESCRIÇÃO_DO_PRODUTO
AS PRODUTO, MAX(CHAVE_TEMPO) AS
ÚLTIMADATA, SUM(QTDE_VENDIDA) AS VENDAS
FROM TfVENDAS ASV, TdPRODUTO AS P,
TdLOJA AS L
WHERE V.CHAVE_LOJA = L.CHAVE_LOJA
AND L.NOME_DA_LOJA = 'IGUATEMI-JPESSOA'
AND V.CHAVE_PRODUTO = P.CHAVE_PRODUTO
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 6 Últimas Vendas da loja Iguatemi-João Pessoa

equivale à agregação por loja das vendas mais recentes da loja Iguatemi-JPessoa.

O projeto e a criação das VMs deverão ser revestidos de alguns cuidados operacionais:

- As VMs deverão compor um modelo separado, com a definição da tabelas de fatos e dimensões (ver Figura 6, pág. 66). Isso evitará contenções mútuas no momento da sua carga ou atualização. A Figura 6 apresenta as tabelas do DW básico e as visões materializadas definidas acima com as respectivas ligações entre elas.

- A carga/atualização dessas tabelas deverá ser analisada com relação à janela (normalmente noturna) disponível para seu processamento. Pelo volume e número de VMs, é importante considerar processos paralelos de carga/atualização, ou uso de máquinas com sistemas operacionais que explorem paralelismo ou mesmo, a análise criteriosa de SGBDs relacionais que paralelizem comandos SQL.

3.2 A Problemática da Atualização de um DW

Como a existência de VMs é importante para o desempenho das consultas OLAP, e o tempo em que o DW está sendo atualizado deve ser o menor possível para que logo esteja disponível, a eficiência do processo de manutenção do DW e suas VMs é de grande importância em ambientes de DWing.

Afinal, estamos falando de tabelas (tabelas de fatos) que poderão conter muitos milhões de registros, portanto qualquer concessão que venha aumentar a velocidade de acesso é vista como positiva.

Um processo bastante utilizado é realizar o processamento envolvendo todos os dados dos BDs de produção. Esse processo

pode ser bastante vantajoso se considerarmos que ele pode fazer uma "limpeza" no DW e suas VMs, pois a cada novo processamento considera um novo estado dos BDs de produção; refletindo assim, as mudanças ocorridas, inclusive mudanças nas estruturas das tabelas. Entretanto, o tempo que esse processo pode levar é praticamente inaceitável!

Dessa forma, um importante problema quanto à manutenção de VMs é como atualizá-las de forma eficiente sem acessar as tabelas de fatos, que são enormes, e considerando apenas as últimas atualizações nos BDs de produção.

3.3 Conclusões

O foco desta dissertação é a atualização eficiente de VMs em DWs. Vimos que os processos de atualização de DWs e VMs existentes são insuficientes ou não correspondem as necessidades dos usuários, pois geralmente é feita uma nova carga dos dados existentes nos BDs de produção. Dessa forma, se faz necessário a existência de um processo de atualização mais próximo da necessidade do usuário que deseja ter maior tempo possível o DW de sua organização.

O próximo capítulo apresenta algumas considerações sobre os trabalhos relacionados a esta dissertação, e que foram de grande importância para a maturidade das idéias expostas.

Capítulo

4 Manutenção Incremental de DWs

Neste capítulo, apresentamos dois trabalhos relacionados a esta dissertação, que apresentam métodos de manutenção incremental de VMs.

4.1 A proposta de Mumick[14]

A proposta intitulada *Manutenção de Cubos de Dados e Tabelas Sumário em um Data Warehouse* é um método de manutenção de visões materializadas, chamado método de *tabela delta-sumário*, e é utilizado para resolver dois problemas da manutenção de VMs em um DW: (1) como manter eficientemente uma VM enquanto reduz o tempo de processamento para essa manutenção, e (2) como manter um grande conjunto de VMs a partir das tabelas básicas. Além disso, muito do trabalho necessário para manutenção de uma VM pelo método delta-sumário pode ser reutilizado

por outras VMs, de tal forma que um conjunto de VMs pode ser mantido eficientemente. Há muitas publicações referentes à escolha e materialização de VMs, mas esta proposta é a primeira que trata de sua manutenção.

Como ocorrem mudanças nas fontes de informação, as VMs devem ser atualizadas para refletir a mudança de estado das fontes. Essa atualização pode ser feita a partir da recomputação dos dados ou usando técnicas de manutenção incremental. Usando a proposta de manutenção incremental, o DW e suas VMs podem ser atualizados imediatamente às mudanças nas fontes de dados, ou quando uma grande quantidade de atualizações ocorrem nas fontes. Na manutenção imediata cada atualização significa um *overhead* para atualização do DW. Esse *overhead* aumenta com o número de visões e sua complexidade. Outro problema com a manutenção imediata em um DW é que ele está sendo acessado. Por essas razões, o DW é, geralmente, atualizado de modo *batch*, com as mudanças nas fontes de dados sendo recebidas durante o dia, por exemplo, e aplicadas ao DW (tabelas básicas e visões materializadas) à noite, durante o qual o DW não está sendo acessado pelos usuários. O DW precisa estar

inacessível aos usuários durante sua atualização, devido ao grande número de atualizações que precisam ser aplicadas. Já que o DW necessita estar disponível aos usuários rapidamente, o tempo para manutenção freqüentemente é um fator limitante. Uma vez que as VMs têm um grande impacto na performance de consultas OLAP, sua manutenção eficiente é crucial.

Usando técnicas de manutenção incremental, é possível aumentar o número de VMs em um DW, ou alternativamente, diminuir o tempo que o DW não está disponível aos usuários. Essa proposta apresenta as seguintes contribuições:

- Técnicas de manutenção incremental são apresentadas, propondo um novo paradigma, chamado método de tabelas delta-sumário, para manutenção de VMs.

- Uma estratégia geral para reduzir o tempo de atualização necessário para manutenção ao dividir o processo em duas funções *propagate* e *refresh*. A primeira ocorre antes do processamento da atualização, e a segunda durante o mesmo.

- O processamento requerido para manter uma VM geralmente pode ser reutilizado para manter outras VMs. Assim,

um conjunto de VMs pode ser mantido mais eficientemente, melhor do que manter cada VM isolada.

4.1.1 Exemplo de Motivação

Considere um exemplo de informações de varejo, com dados de pontos-de-venda de centenas de armazéns. O dado do ponto-de-venda é armazenado no DW em uma grande tabela chamada `vendas`, que é a tabela de fatos e que contém um registro para cada item vendido nas transações de vendas. Cada registro tem o seguinte formato:

vendas(#armazemID, #itemID, data, qtde, preço)

Os atributos do registro são o identificador do armazém, do item vendido, a data da venda, a quantidade de itens vendidos, e o preço. A tabela `vendas` pode conter registros duplicados, por exemplo, quando um item é vendido em diferentes transações no mesmo armazém e na mesma data.

Em adição, as tabelas de dimensão são armazém e item e suas chaves são armazemID e itemID, respectivamente.

armazem(#armazemID, cidade, região)

item(#itemID, nome, categoria, preço)

Notamos as hierarquias de dimensões em ambas as tabelas. Por exemplo, na tabela armazém, o campo armazemID funcionalmente determina cidade, que, por sua vez, determina região. O mesmo ocorre em item, onde itemID funcionalmente determina nome, categoria e preço. De modo a responder mais rapidamente as consultas, foram definidas algumas VMs. O Código 7 mostra a definição de quatro VMs.

```
CREATE VIEW AID_vendas(armazemID, itemID, data, TotalCount,
TotalQuantidade) AS
SELECT armazemID, itemID, data, COUNT(*) AS TotalCount,
SUM(qtde) AS TotalQuantidade
FROM vendas
GROUP BY armazemID, itemID, data

CREATE VIEW aCD_vendas(cidade, data, TotalCount,
TotalQuantidade) AS
SELECT cidade, data, COUNT(*) AS TotalCount, SUM(qtde) AS
TotalQuantidade
FROM vendas, armazem
WHERE vendas.armazemID = armazem.armazemID
GROUP BY cidade, data

CREATE VIEW AiC_vendas(armazemID, categoria, TotalCount,
VendaRecente, TotalQuantidade) AS
SELECT armazemID, categoria, COUNT(*) AS TotalCount,
MIN(data) AS EarliestSale, SUM(qtde) AS TotalQuantidade
FROM vendas, item
WHERE vendas.itemID = item.itemID
GROUP BY armazemID, categoria

CREATE VIEW aR_vendas(região, TotalCount, TotalQuantidade)
AS
SELECT região, COUNT(*) AS TotalCount, SUM(qtde) AS
TotalQuantidade
FROM vendas, armazem
WHERE vendas.armazemID = armazem.armazemID
GROUP BY região
```

Código 7 Definição de Visões Materializadas da proposta de Mumick

As visões acima apresentam agregações em vários níveis. Abaixo, definimo-las:

- AID_vendas: representa a quantidade vendida e as vendas por armazém, item e data;
- aCD_vendas: representa a quantidade vendida e as vendas por cidade e data;
- AiC_vendas: representa a mais recente quantidade vendida por loja e categoria de item;
- aR_vendas: representa a quantidade vendida por região.

Os nomes das VMs foram escolhidos para refletir os atributos agregados. O caracter A representa armazemID, I representa itemID, e D representa data. A notação aC representa a agregação cidade por armazém, aR representa armazém por região, e iC representa a um item por categoria. Por exemplo, o nome AiC_vendas implica que armazemID e categoria são os atributos da cláusula *group by* na definição da VM.

A medida que as vendas acontecem, mudanças representando os novos dados são acrescidos ao DW. Pelas razões mencionadas anteriormente, muitos DWs não aplicam essas mudanças imediatamente. Ao invés disso, elas são armazenas e aplicadas ao DW e VMs, geralmente, à noite em um processo *batch*. Isso porque as análises e consultas ao DW devem mostrar uma consistência temporal do dado ao longo do dia.

Embora seja mais freqüente que as mudanças envolvam apenas inserções, para esta proposta assumiremos que as mesmas envolvem tanto inserções quanto remoções. De modo a manter corretamente uma VM na presença de remoções é necessário incluir uma função agregada COUNT(*) na mesma. Com isso é possível determinar quando todas os registros em grupo foram removidos (ou seja, quando COUNT(*) para um grupo retorna 0), implicando em remover o registro do grupo da VM. Incluímos COUNT(*) explicitamente nas VMs anteriores, mas isso pode ser adicionado implicitamente quando a VM é materializada no DW.

Para simplificar a apresentação, assumimos nesta proposta que a manutenção é realizada em resposta as mudanças ocorridas

apenas na tabela de fatos, e que colunas adicionadas não incluem valores nulos.

4.1.1.1 Mantendo uma VM

Nesta sessão, mostraremos um exemplo do método de tabela delta-sumário na manutenção da visão materializada AID_vendas (ver Figura 7, pág. 104). Na próxima seção, mostraremos que muito trabalho usado para manter essa VM poderá ser reutilizado para manter outra VM da figura.

Um aspecto importante do algoritmo é que o processo de manutenção é dividido em duas funções: *propagate* e *refresh*. O trabalho de computação do método acontece na função *propagate*, que não bloqueia o acesso às VMs de modo que o DW continua sendo submetido a consultas pelos usuários. Este estado se mantém até a função *refresh*, durante a qual as VMs são atualizadas.

Propagate: A funçao *propagate* envolve a criação de tabelas delta-sumário a partir do conjunto de novos registros obtidos das fontes de informação. Essas tabelas representam as modificações que irão ocorrer nas VMs devido a atualizações na tabela de fatos. O conjunto de inserções será armazenado na tabela ven-

Para computar a tabela delta-sumário, primeiro realizamos a projeção nos registros inseridos e removidos tal que temos 1 para count e qtde para quantidade dos registros inseridos, e o valor negativo desses valores para os registros removidos. Então, faremos a união desse resultado e o agregaremos, agrupando pelos mesmos atributos da cláusula *group by* da VM. A agregação resultante representa as mudanças correspondendo aos valores das funções agregadas da VM.

Refresh: A função *refresh* aplica as mudanças representadas na tabela delta-sumário à VM. A função mostrada no Código 9, recebe como entrada a tabela delta-sumário ds_AID_vendas, a VM AID_vendas, e atualiza a mesma para refletir as modificações da tabela delta-sumário. Para simplificar, assumiremos aqui que não há valores nulos na tabela de fatos vendas.

A função *refresh* foi otimizada para executar eficientemente. Cada registro da tabela delta-sumário causa uma simples atualização na VM.

```
Para cada registro δr em ds_AID_vendas:
        Faz o registro r = (SELECT * FROM AID_vendas d,
                            WHERE d.armazemID = δr.armazemID AND  d.data
                            = δr.data  AND d.itemID = δr.itemID)
```

```
Se não encontrar r
        Insira o registro δr em AID_vendas
Senão
        Se δr.ds_Count + r.TotalCount = 0,
                Remova o registro r de AID_vendas
        Senão
                Atualize o registro r.TotalCount += δr.ds_Count,
                                r.TotalQuantidade +=
δr.ds_Quantidade
```

Código 9 Função Refresh para AID_vendas

4.1.1.2 Mantendo múltiplas VMs

Agora mostraremos a função *propagate* que cria tabelas delta-sumário para as outras VMs do nosso exemplo. Agora enfatizamos a manutenção eficiente de múltiplas VMs paralelamente, o que permite conseguir uma maior otimização, ao contrário da manutenção individual. A otimização ocorre porque a tabela delta-sumário computada para manter uma VM pode ser utilizada para computar tabelas delta-sumário de outras VMs. Se uma tabela delta-sumário já envolve alguma agregação, é mais fácil que ela seja menor que o conjunto de mudanças, então ao usar essa tabela para computar outras tabelas delta-sumário acessaremos um número menor de registros, o que é mais eficiente do que computar cada tabela delta-sumário diretamente das mudanças.

As consultas definindo as tabelas delta-sumário para aCD_vendas, AiC_vendas, e aR_vendas são mostradas no

Código 10. As tabelas delta-sumário para aCD_vendas e AiC_vendas referenciam a tabela delta-sumário para AID_vendas, e a tabela delta-sumário para aR_vendas referencia a tabela delta-sumário para aCD_vendas.

```
CREATE VIEW ds_aCD_vendas(cidade, região, data, ds_Count,
ds_Quantidade) AS
SELECT cidade, região, data, sum(ds_Count) AS ds_Count,
sum(ds_Quantidade) as ds_Quantidade
FROM ds_AID_vendas, armazem
WHERE ds_AID_vendas.armazemID = armazem.armazemID
GROUP BY cidade, região, data

CREATE VIEW ds_AiC_vendas(armazemID, categoria, ds_Count,
ds_VendaRecente, ds_Quantidade) AS
SELECT armazemID, categoria, sum(ds_Count) AS ds_Count,
min(data) AS VendaRecente, sum(ds_Quantidade) as
ds_Quantidade
FROM ds_AID_vendas, item
WHERE ds_AID_vendas.itemID = item.itemID
GROUP BY armazemID, categoria

CREATE VIEW ds_aR_vendas(região, ds_Count, ds_Quantidade) AS
SELECT região, sum(ds_Count) AS ds_Count, sum(ds_Quantidade)
as ds_Quantidade
FROM ds_aCD_vendas
GROUP BY região
```

Código 10 Função Propagate para aCD_vendas, AiC_vendas e sR_vendas

Note que, a tabela delta-sumário ds_aCD_vendas inclui o atributo região, que não é necessário manter aCD_vendas. Ele é incluído porque mais tarde, na definição de ds_aR_vendas, iremos precisar unir ds_aCD_vendas com a tabela de dimensão armazém. A inclusão do atributo região em ds_aCD_vendas não

afeta a manutenção de aCD_vendas porque na hierarquia de dimensão para cidades temos que especificar que cidade, funcionalmente, determina região, isto é, cada cidade pertence a uma única região, então agrupando por (cidade, região, data) resulta no mesmo grupo que agrupando por (cidade, data).

4.1.1.3 Algoritmo básico

Nesta seção mostramos o algoritmo do método delta-sumário e suas etapas.

4.1.1.3.1 Função *Propagate*

A função *propagate* cria uma tabela delta-sumário que contém os registros das mudanças nas VMs. Como essa função não afeta a VM, ela pode continuar sendo acessada pelos usuários enquanto a função é computada. Consequentemente, o objetivo desta função é fazer tanto trabalho quanto possível para que o tempo requerido para a função *refresh* seja minimizado.

4.1.1.3.1.1 Preparando as mudanças

Para facilitar o entendimento, dividimos o trabalho inicial em três etapas, para a definição de visões virtuais: *prepare-changes*, *prepare-insertions* e *prepare-deletions*. A visão virtual *prepare-changes* é

definida simplesmente como a união de *prepare-insertions* e *prepare-deletions*, que são descritas a seguir.

As visões virtuais *prepare-insertions* e *prepare-deletions* retornam as mudanças das funções agregadas causadas pelas inserções e remoções individuais, respectivamente, para os dados básicos. Primeiro aplicam uma projeção das inserções/remoções para os dados básicos, depois aplicam algumas condições de seleções e uniões que aparecem na definição da VM. Os atributos projetados incluem cada atributo da cláusula *group by* da VM, e atributos *aggregate-source* correspondendo à cada função de funções agregadas computadas na VM.

Um atributo *aggregate-source* computa o resultado da expressão na qual a função agregada é aplicada. Por exemplo, se a VM inclui a função agregada sum(a*b), as visões virtuais *prepare-insertions* e *prepare-deletions* poderiam incluir em suas cláusulas select um atributo agregado computando a*b (*prepare-deletions* ,incluindo –(a*b)). Veremos mais tarde que, quando a tabela delta-sumário é computada, os atributos *aggregate-source* são agregados.

Esses atributos são derivados de acordo com a Tabela 2. A coluna intitulada *prepare-insertions* descreve como eles são derivados para a visão *prepare-insertions*; a coluna *prepare-deletions* descreve como eles são derivados para a visão *prepare-deletions*.

	prepare-insertions	prepare-deletions
COUNT(*)	1	-1
COUNT(expr)	0 se expr é nulo, senão 1	0 se expr é nulo, senão –1
SUM(expr)	Expr	-expr
MIN(expr)	Expr	expr
MAX(expr)	Expr	expr

Tabela 2 Derivando atributos *aggregate-source*

Exemplo: Considere a visão AiC_vendas do Código 7. As visões virtuais *prepare-insertions*, *prepare-deletions* e *prepare-changes* para AiC_vendas são mostradas no Código 11. A visão *prepare-insertions* é prefixada pi_, *prepare-deletions* é prefixada pd_, e *prepare-changes* é prefixada pc_. Os atributos *aggregate-source* são _count, _date, e _quantidade, respectivamente.

```
CREATE VIEW pi_AiC_vendas(armazemID, categoria, _count,
_data, _quantidade) AS
SELECT armazemID, categoria, 1 AS _count, data AS _data,
qtde AS _quantidade
FROM vendas_ins, itens
WHERE vendas_ins.itemID = item.itemID

CREATE VIEW pd_AiC_vendas(armazemID, categoria, _count,
_data, _quantidade) AS
SELECT armazemID, categoria, -1 AS _count, data AS _data, -
qtde AS _quantidade
FROM vendas_del, itens
WHERE vendas_del.itemID = item.itemID

CREATE VIEW pc_AiC_vendas(armazemID, categoria, _count,
_data, _quantidade) AS
SELECT * FROM (pi_AiC_vendas UNION ALL pd_AiC_vendas)
```

Código 11 Visões virtuais da função propagate para atualização da visão materializada AID_vendas

4.1.1.3.1.2 Computando a tabela delta-sumário

A tabela delta-sumário é computada pela agregação da visão virtual *prepare-changes*. Esta tabela tem o mesmo esquema da VM, exceto pelos atributos resultantes das funções de agregação nas tabelas delta-sumário. Por essa razão, nomeamos esses atributos com seu nome correspondente na VM prefixado de ds_.

Cada registro da tabela delta-sumário descreve o efeito das mudanças nos dados básicos nas funções agregadas de um registro correspondente na VM (isto é, o registro correspondente na VM tem os mesmos valores para todos os atributos da cláusula *group by* do registro na tabela delta-sumário). Note que, um registro correspondente na VM pode não existir, e neste caso, é ne-

cessário na função *refresh* inserir um registro na VM devido as mudanças representadas na tabela delta-sumário.

A consulta para computar a tabela delta-sumário segue da consulta para a computação da VM, e difere no seguinte:

- A cláusula FROM recebe *prepare-changes*.
- A cláusula WHERE é removida, já que a união entre as tabelas foi feita quando da definição de *prepare-insertions* e *prepare-deletions*.
- As expressões nas quais as funções agregadas são aplicadas são substituídas pelos atributos *aggregate-source* referentes em *prepare-changes*.
- As funções agregadas COUNT são substituídas por SUM.

Exemplo: Considere novamente a visão AiC_vendas do Código 7. A consulta para computar a tabela delta-sumário para AiC_vendas é mostrada abaixo. Ela agrega as mudanças representadas na visão virtual *prepare-changes*, agrupando pelos mesmos atributos da cláusula *group by* da VM.

```
CREATE VIEW ds_AiC_vendas(armazemID, categoria,
ds_Count, ds_RecenteVenda, ds_Quantidade) AS
```

```
SELECT armazemID, categoria, sum(_count) AS ds_Count,
min(data) AS ds_RecenteVenda, sum(_quantidade) AS
ds_Quantidade
FROM pc_Aic_vendas
GROUP BY armazemID, categoria
```

Código 12 Criação da tabela delta-sumário para a visão materializada AiC_vendas

Note que, a tabela delta-sumário para a visão AiC_vendas já foi mostrada e foi definida usando AID_vendas. Aqui, em vez disso, definimos a tabela delta-sumário para AiC_vendas usando as mudanças nos dados básicos.

4.1.1.3.2 Função *Refresh*

A função *refresh* aplica as mudanças representadas na tabela delta-sumário à VM correspondente. Cada registro na tabela delta-sumário causa uma mudança em um único registro correspondente da VM. O registro correspondente na VM é atualizado, removido, ou se não for encontrado, é inserido na mesma.

O algoritmo da função *refresh* é mostrado no Código 13.

4.2 A Proposta de Filtro (Detecção de Dados Relevantes)[3]

Em certos casos, um conjunto de atualizações de uma tabela não afeta o estado de uma visão. Quando isso ocorre, independentemente do estado do banco de dados, chamamos o conjunto de atualizações de irrelevante. É importante prover um mecanismo eficiente para a detecção de atualizações irrelevantes, de tal forma que, a reavaliação da expressão relacional que define uma visão possa ser evitada ou o número de registros considerados reduzido.

Esse mecanismo explora o conhecimento provido tanto da expressão de definição da visão quanto das operações de atualizações do banco de dados. Ele considera apenas as expressões da álgebra relacional formadas pela combinação de seleções, projeções e junções, chamadas *expressões SPJ*.

Considere a visão definida pela expressão

$$v = \pi_X(\sigma_{C(Y)}(R_1 \times R_2 \times \ldots \times R_p))$$

onde **C(Y)** é uma expressão Booleana e **X** e **Y** são conjuntos de variáveis denotando os nomes de alguns atributos para as relações chamadas R_1, R_2, ..., R_p. Os conjuntos **X** e **Y** não são ne-

cessariamente iguais (isto é, nem todos os atributos na projeção participam da condição de seleção e vice versa), e de fato podem ser disjuntos.

Suponha que uma tupla $t = (a_1, a_2, ..., a_q)$ é inserida dentro da relação r_k definida no esquema $R_k[3]$.Seja $Y_1 = R_k \cap Y$, e $Y_2 = Y - Y_1$, então que $Y = Y_1 \cup Y_2$. Vamos modificar a condição de seleção C(Y), ao substituirmos as variáveis Y_1 por seus valores correspondentes $t(Y_1)$. Se a condição modificada C(Y) não é satisfeita independente do estado do banco de dados, então a tupla inserida t em r_k não afeta a visão v.

Considere dois esquemas de relações R = {A,B} e S = {B,C}, e uma visão V definida como V = R ⊗ S. Vamos denotar por r e s instâncias dos esquemas das relações nomeadas R e S, respectivamente, e $v = r \otimes s$. Assuma que uma transação T atualiza as relações r e s.

Caso 1 : $t \in i_r \otimes i_s$ é uma tupla que tem que ser inserida em v.

Caso 2 : $t \in i_r \otimes s$ é uma tupla que tem que ser inserida em v.

Caso 3 : $t \in r \otimes s$ é uma tupla que já existe na visão v.

Geralmente, descrevemos o valor do campo correspondente à tupla resultante de uma operação *Join* de duas tuplas e, similarmente o valor do campo correspondente das tuplas resultantes de operações *Select* ou *Project*, de acordo com a Tabela 3:

r_1	r_2	$r_1 \otimes r_2$			r	$\sigma_{C(Y)}(r)$	$\pi_X(r)$
Insert	insert	Insert			insert	insert	insert
Insert	delete	Ignore			delete	delete	delete
Insert	old	Insert			old	old	old
Delete	insert	Ignore					
Delete	delete	Delete					
Delete	old	Delete					
old	insert	Insert					
old	delete	Delete					
old	old	Old					

Tabela 3 Regras de Filtro

onde as três últimas colunas representam o comportamento do dado com relação as operações *Join*, *Select* e *Project* respectivamente.

Assim, quando tratamos essas operações podemos descobrir os dados que podem realmente afetar determinada visão.

Exemplo : Considere duas relações r e s definidas em R = {A, B} e S = {C, D}, respectivamente, e uma visão v definida como

$$v = \pi_{A, D}(\sigma_{(A<10)\wedge(C>5)\wedge(B=C)}(R \times S)).$$

Isto é, a condição C(A,B,C) = (A<10)^(C>5)^(B=C).

r:	A	B		s:	C	D		v:	A	D
	1	2			2	10			5	20
	5	10			10	20				
	12	15								

Tabela 4 Exemplo de Filtro

Suponha que a tupla (9,10) é inserida na relação *r*. Podemos substituir os valores (9,10) para as variáreis A e B em C(A,B,C) para obter a condição modificada C(9,10,C) = (9<10)^(C>5)^(10=C). A condição de seleção C(9,10,C) é satisfeita, isto é, existem instâncias das relações nomeadas R e S contendo as tuplas (9,10) e (10,δ), para algum valor de δ, tal que C(9,10, δ) = *True*. Portanto, inserir a tupla (9,10) dentro da relação *r* é relevante à visão *v*. Note que, poderia existir algum estado de *s* que não "casaria" com a tupla (10, δ), neste caso, a tupla (9,10) não teria afetado a visão. Entretanto, a forma de verificar isto é checando os conteúdos do banco de dados.

Por outro lado, suponha que a tupla (11,10) é inserida na relação *r*. Após substituirmos os valores (11,10) para as variáveis A e B em C(A,B,C) obtemos

C(11,10,C) = (11<10)^(C>5)^(10=C)

Podemos ver que a condição C não é satisfeita independente do estado do banco de dados. Portanto, inserir a tupla (11,10) na relação *r* é irrelevante à visão *v*.

Essas considerações são parte de [3] que trata da manutenção de VMs. Porém, este trabalho adota a abordagem de atualização on-line ou imediata do DW, ou seja, cada modificação, que afeta o DW, ocorrida nas fontes de informação é tratada e utilizada para atualizar o DW. Apesar disto, as idéias principais de filtro de dados relevantes são úteis e serão consideradas neste trabalho.

4.3 Conclusões

Vimos que os métodos apresentados podem consumir bastante tempo de processamento[14], por considerarem todos os novos dados oriundos das fontes de informação, ou por atualizarem o DW à medida que as fontes de informação sofrem qualquer tipo de mudança[3].

O próximo capítulo apresenta o algoritmo *Atualiza_DW*, que tenta solucionar os principais problemas da atualização de um DW e suas VMs.

Capítulo

5 O Algoritmo *Atualiza_DW* para Atualização Eficiente de VMs

Vimos no capítulo anterior que é possível melhorar a performance do processo de manutenção de VMs e DW, através da atualização incremental do mesmo.

Inicialmente, o método apresentado em [14] não menciona a atualização da(s) tabela(s) de fatos do DW; apenas envolve a manutenção das VMs do esquema de DWing, supondo que o DW primário tenha sido atualizado. Além disso, usa todos os dados envolvidos para essa atualização. É preciso prover um método eficiente para a atualização do DW que, além de considerar o conceito incremental, consiga filtrar, dos dados envolvidos, apenas aqueles que podem afetar o estado das VMs.

Neste capítulo, apresentamos um algoritmo chamado de *Atualiza_DW* para atualização das VMs de um DW, contem-

plando também os dados básicos do mesmo, ou a(s) sua(s) tabela(s) de fatos.

Iniciamos, pela ordem, com a entrada, a saída, e as idéias principais empregadas na definição do algoritmo. Finalizamos com um exemplo ilustrando tudo o que foi explicado. O pseudo-código do algoritmo encontra-se no Anexo a esta dissertação.

5.1 O algoritmo *Atualiza_DW*

Uma vez conhecida a proposta de Mumick, podemos apresentar o algoritmo *Atualiza_DW*, pois a partir dos conceitos lá mencionados, poderemos apresentar as idéias do algoritmo e mostrar como se dá a atualização do DW, através de um exemplo.

5.1.1 Entrada

Inicialmente, teremos que conhecer a tabela que conterá os novos dados fontes acrescidos aos dados operacionais. Ela será criada em tempo de projeto do DW, e povoada a cada dia, supondo que a atualização do DW seja diária. Quando o DW for atualizado, ela será novamente inicializada. Essa tabela será definida de acordo com a estrutura do DW, pelo administrador do

sistema, que considera, principalmente, a(s) tabela(s) de fatos e as VMs. Assim como as tabelas do DW, esta tabela terá informações no repositório de metadados. Durante o dia, ela será constantemente utilizada pelos Extratores/Integrador[5], apresentados na Figura 5, para que os primeiros detectem os novos dados acrescidos às fontes, e o segundo trate da qualidade desses dados. Assim, o algoritmo *Atualiza_DW* já recebe essa enorme tabela na forma que precisamos e com os dados consistentes. Considere o esquema de DW da Figura 3.

Além do esquema do DW, o algoritmo faz constantes consultas ao Repositório de Metadados para conhecer as estruturas das tabelas e visões, além de outras informações lá registradas e necessárias ao processo de manutenção.

A tabela de novos dados fontes será chamada de `tMovimento`, bem adequado ao propósito do esquema de DW do nosso exemplo, pois o mesmo se aplica a uma rede de supermercados, o que denota a quantidade de movimento de produtos ocorrido nas lojas da rede de supermercados durante o dia. Assim, como a chave primária da tabela de fatos do nosso exemplo

[5] Está fora do escopo deste trabalho a filtragem/tratamento da qualidade de dados fontes de um DWing.

é chave_loja, chave_produto, chave_tempo, tMovimento deve conter esses atributos, além de outros mais, a partir dos quais possamos obter as métricas da tabela de fatos, e consequentemente, das visões materializadas.

Assim, a tabela tMovimento tem a seguinte estrutura:

```
tMovimento
(#regis-
tro,loja,produto,data,qtde_vendida,valor_vendido, va-
lor_custo)
```

Note que, a partir de tMovimento poderemos obter os registros de TfVendas, pois a chave da última está em tMovimento e os campos de medida são facilmente obtidos. Note também que, tanto a definição dessa tabela irá depender de uma análise da estrutura das tabelas do DW, como seu próprio nome, no nosso caso tMovimento, pois se aproxima bastante do sentido do DW.

5.1.2 Processamento

Uma vez definida a entrada do algoritmo, podemos apresentar as principais idéias empregadas no nosso processo de manutenção do DW e suas VMs. Portanto, o objetivo do algoritmo é atualizar o DW básico e suas VMs, a partir da tabela tMovimento.

A atualização do DW básico se dá de forma incremental. Para isso, fizemos as seguintes suposições:

- Granularidade (`tMovimento`) = Granularidade (tabela(s) de fatos)
- Qualquer tempo de `tMovimento` > Qualquer tempo de tabela(s) de fatos.

Ou seja, supomos que a massa de novos dados sempre representa um novo valor do tempo e, geralmente, isso é verdade. Assim, a atualização da(s) tabela(s) de fatos é bastante simplificada, constituindo em um grau de agregação igual ou maior que o da tabela `tMovimento`, e ambas representam o movimento diário da rede de supermercados. O conceito de atualização incremental aqui significa que a(s) tabela(s) de fatos sofre(m) um acréscimo de dados e que não há a necessidade de um novo processamento ou uma recomputação para mantê-la(s).

A atualização das visões materializadas também se dá de forma incremental, mas além do acréscimo de dados há também a atualização de valores já existentes. Neste caso, o conceito de atualização incremental significa a não necessidade de um novo processamento ou uma recomputação para a manutenção.

O processo de atualização se dá em três etapas:

5.1.2.1 Preparar

Durante a etapa Preparar, o algoritmo atualiza a(s) tabela(s) de fatos a partir de `tMovimento`, ao mesmo tempo em que gera[6] tabelas auxiliares para as VMs também a partir de `tMovimento`.

Para atualizar uma VM, inicialmente, o algoritmo faz um filtro em `tMovimento`, a fim de identificar aqueles dados que podem mudar o seu estado, ou seja, ele irá tentar descobrir, a partir da definição da VM, quais os dados de `tMovimento` que podem afetar seu estado.

Quando nos referimos aos dados que afetam uma VM, significa dizer que esses dados são relevantes à manutenção da mesma ou afetam seu estado. Por exemplo, se temos uma VM que retorna as vendas de leite nos últimos quinze dias, então se em `tMovimento` tivermos registros de vendas de leite neste intervalo, certamente esses registros afetam o estado da VM. Ou seja, se antes a VM retornava um valor X de vendas de leite, agora, possivelmente, irá retornar X'.

[6] A geração das tabelas auxiliares se dá a cada atualização de modo a manter a consistência entre as mesmas e as VMs correspondentes, devido a mudanças nas estruturas das VMs

Uma vez detectados os dados relevantes à manutenção da VM, o algoritmo armazena-os em uma *tabela auxiliar*. O nome da tabela auxiliar será prefixado *taux* acrescido do nome da visão. Portanto, cada visão poderá ter uma tabela auxiliar a ela associada e contendo aqueles dados de `tMovimento` que afetam seu estado.

As tabelas auxiliares são criadas a partir das seguintes regras:

- Os atributos projetados na visão a ser atualizada serão projetados na tabela auxiliar;
- O nome da tabela de fatos refenciada na cláusula FROM, será substituído por `tMovimento`, fazendo as respectivas correspondências de atributos. As tabelas de dimensão irão aparecer, para a operação *Join*.
- As funções agregadas não aparecem, pois serão aplicadas na próxima etapa.
- Não aparece a cláusula GROUP BY.

Além das tabelas auxiliares, a etapa Preparar cria e/ou povoa[7] uma tabela de associação, chamada *Tabela_Associação_Auxiliar*, contendo o nome de cada VM e a tabela auxiliar correspondente. Essa tabela é definida da seguinte forma:

```
taaTabela_Associação_Auxiliar(nome_visão, nome_tab_aux)
```

Com isso, será mais fácil e rápido manter uma VM, pois o algoritmo irá acessar um conjunto bastante reduzido, e não a tabela `tMovimento`, que é enorme.

Lembre que dissemos que cada VM poderá ter uma tabela auxiliar a ela associada. Isso por que, antes de criar as tabelas auxiliares, o algoritmo verifica dependências entre as VMs[21]. Se uma visão pode ser obtida a partir de outra visão, o algoritmo não terá que criar duas tabelas auxiliares, como veremos mais adiante.

Regras baseadas nas definições das VMs serão utilizadas no processo de detecção de dados relevantes à manutenção, como

[7] A *Tabela_Associação_Auxiliar* é criada uma única vez e povoada a cada atualização devido a criação de novas VMs.

condições de seleção, atributos projetados e atributos chaves da(s) tabela(s) de fatos.

Basicamente, todas as VMs são definidas a partir da(s) tabela(s) de fatos do DW, porém algumas VMs podem ser obtidas a partir de outras. Assim, a etapa Preparar irá detectar essa dependência entre visões, criando e/ou povoando[8] uma tabela de dependências, chamada *Tabela_Dependência_Visão*, contendo o nome da visão e o nome da tabela de fatos da qual pode ser obtida, ou no caso de dependência, o nome da visão que pode originá-la. Neste caso, se uma visão X pode ser obtida da visão Y, então não é necessário criar uma tabela auxiliar para a visão X, pois ela estará automaticamente associada a tabela auxiliar criada para visão Y. A Tabela_Dependência_Visão será definida da seguinte forma:

```
tdvTabela_Dependência_Visão(nome_visao, nome_fatos_ou_visão)
```

Durante essa etapa, o algoritmo acessa as duas tabelas criadas, de associação e dependência, e verifica a quais tabelas as VMs estão associadas e/ou são dependentes. Isto melhora o

[8] A Tabela_Dependência_Visão também é criada uma única vez e povoada a cada atualização devido a criação de novas VMs.

processamento, pois essas informações já estão armazenadas e não precisarão ser descobertas no momento da atualização, o que poderia acarretar algum atraso.

Abaixo, mostramos graficamente a etapa Preparar:

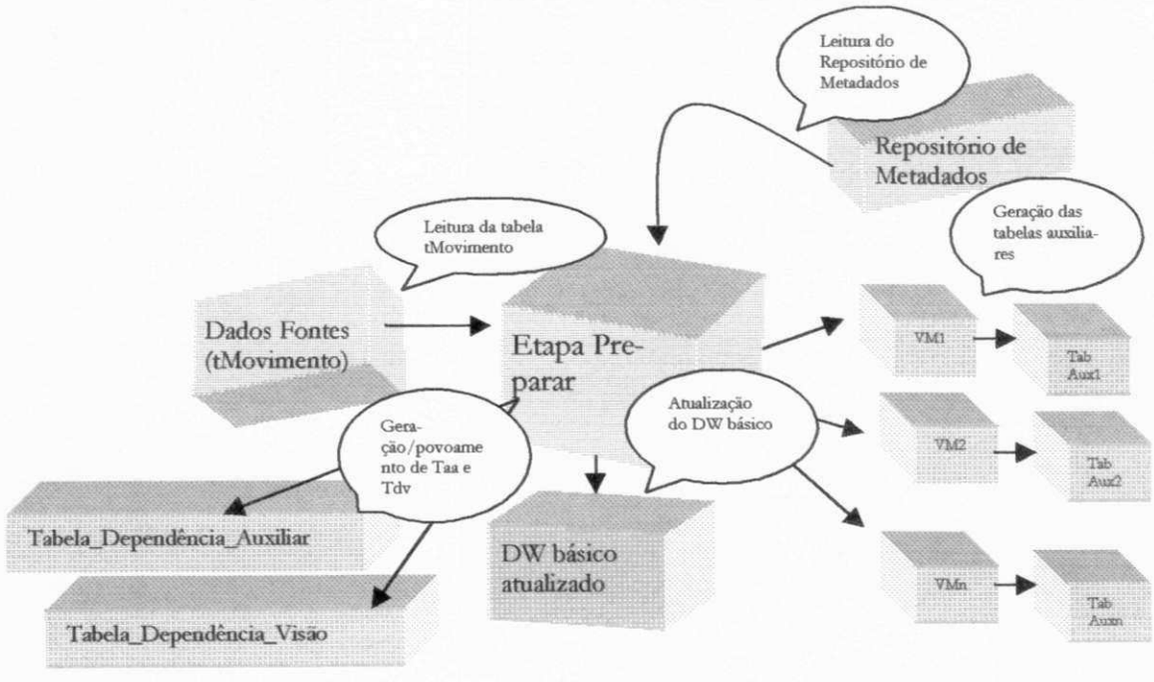

Figura 7 Etapa Preparar

Como podemos ver da Figura 7, os dados fontes são "desviados" de acordo com as definições das visões para as tabelas auxiliares. Além disso, o DW básico é atualizado e as dependências e tabelas auxiliares são registradas nas tabelas de dependência e associação, respectivamente.

Utilizando as regras de filtro apresentadas no capítulo anterior, o algoritmo consegue identificar os dados relevantes à visão, que podem afetar seu estado. Para exemplificar o processo de filtro acima descrito, considere a VM `vmVendas_por_Produto_Out_1999_Iguatemi` (ver Figura 6, pág. 66 e Código 14, pág. 113).

A VM apresenta as operações esperadas, ou seja, atributos projetados, união entre tabelas e condições de seleções. Assim, para a visão `vmVendas_por_Produto_Out_1999_Iguatemi`, será criada uma tabela auxiliar, já que essa VM é obtida da tabela de fatos e não de outra visão. Ela conterá os mesmos atributos da visão, e os registros de `tMovimento` que satisfazem, no caso, as condições de seleção presentes na definição da VM.

Por exemplo, se a tabela `tMovimento` contém o registro *r* = (19/**10**/**1999**,**Iguatemi**-Salvador,Pão,0.50,5,0.15), ele certamente será relevantes para a visão, portanto, a tabela auxiliar receberá o registro *r*. Entretanto, o registro *r'* = (02/**09**/**1999**,**Iguatemi**-Salvador,Pão,0.50,5,0.15) não é relevante para a visão, pois, apesar da loja ser Iguatemi e o ano 1999, o

mês é setembro e não outubro. Assim, esse registro não será considerado para tabela auxiliar da visão.

Após atualizar o DW básico, criar as tabelas auxiliares e as tabelas de associação e de dependências, o algoritmo entra na etapa Propagar, descrita a seguir.

5.1.2.2 Propagar

Nessa etapa, o algoritmo gera as visões auxiliares, que representam a propagação das mudanças ocorridas nas fontes adequadas à definição da visão. Ela recebe a tabela auxiliar relacionada à visão a ser mantida, portanto, contendo apenas aquelas informações que afetam o estado da visão, e cria uma visão ou consulta a partir destes dados, chamada delta-sumário, criada diretamente a partir da tabela auxiliar de forma mais rápida e eficiente e, com a certeza de estar trabalhando com dados corretos e úteis para a visão.

A visão delta-sumário será criada a partir das seguintes regras:

- Os atributos projetados na visão a ser atualizada e também da tabela auxiliar serão projetados na visão delta-sumário.

mesmos produtos, exceto que o preço do Pão estará em promoção e custará R$ 0,05.

Assim, considere a Tabela 20 em Anexo representando `tMovimento` com esses registros.

As linhas em negrito representam os novos dados fontes inseridos em `tMovimento` após a última atualização do DW e VMs.

Abaixo apresentamos alguns atributos e registros das tabelas de dimensão `TdProduto`, `TdLoja` e `TdTempo`:

Chave_Produto	Estoque	Descrição_do_Produto	Marca	Categoria	...
...	...	...	...	...	...
P100000	S	Pão 50grs	A	A	...
P100001	S	Leite	A	B	...
P100002	S	Iogurte	A	B	...
P100003	S	Manteiga	A	B	...
...	...	...	...	...	...

Tabela 5 Tabela de Dimensão TdProduto

Chave_Loja	Nome_da_Loja	Endereço	Cidade	...
...	...	...	...	...
L100000	Iguatemi-Salvador	Rua A	Salvador	...
L100001	Iguatemi-JPessoa	Rua B	João Pessoa	...
L100002	Iguatemi-CGrande	Rua C	Campina Grande	...
...	...	...	...	...

Tabela 6 Tabela de Dimensão TdLoja

Chave_Tempo	Dia_da_semana	Dia_do_Mes	Mês	Trimestre	Ano	
...	...	...	...	...		
T100000	Terça-Feira	19	Outubro	3	1999	...

T100001	Quarta-Feira	20	Outubro	3	1999	...
T100002	Quinta-Feira	21	Outubro	3	1999	...
...	...	...	...	...		

Tabela 7 Tabela de Dimensão TdTempo

Abaixo temos a tabela de fatos `TfVendas` atualizada no dia 20/10/1999:

Tempo	Loja	Produto	Valor_Vendido_Real (R$)	Qtde_Vendida	Custo_Real (R$)	Lucro (%)
T100000	L100000	P100000	3,50	35	1,05	95
T100000	L100000	P100001	2,70	3	2,10	47
T100000	L100000	P100002	9,00	3	6,60	15
T100000	L100001	P100000	10,00	100	3,00	33
T100000	L100001	P100001	4,50	5	3,50	28
T100000	L100001	P100002	3,00	1	2,20	45
T100001	L100000	P100000	10,50	105	3,15	31
T100001	L100000	P100001	9,00	10	7,00	14
T100001	L100001	P100000	20,00	200	6,00	16
T100001	L100001	P100001	8,10	9	6,30	15
T100001	L100001	P100002	18,00	6	13,20	7,5

Tabela 8 Tabela de Fatos TfVendas até o dia 20/10/1999

A visão que utilizaremos para exemplificar a aplicação do algoritmo *Atualiza_DW* é `vmVendas_por_Produto_Out_1999_Iguatemi`, e representa um agrupamento por produto de todas as lojas da rede Iguatemi no mês de Outubro de 1999. Ela contém os seguintes registros até sua última atualização no dia 20/10/1999:

Produto	Vendas
Pão 50grs	440
Leite	27
Iogurte	10

Tabela 9 VM vmVendas_por_Produto_Out_1999_Iguatemi até o dia 20/10/1999

Note que os dados da tabela de fatos e da visão são referentes ao dia 20 de outubro e não contemplam os novos dados, do dia 21. Agora podemos aplicar o algoritmo *Atualiza_DW* para atualizar a tabela de fatos `TfVendas` e a VM `vmVendas_por_Produto_Outubro_1999_Iguatemi`. Iremos mostrar as saídas de cada etapa do algoritmo.

Etapa Preparar: sua saída será a tabela de fatos atualizada (ver Tabela 10, pág. 113), a tabela auxiliar para visão (ver Tabela 21, em anexo) , a tabela de associação auxiliar (ver Tabela 11, pág. 114) e de dependência (ver Tabela 12, pág. 114). Dessa forma temos:

Tempo	Loja	Produto	Valor_Vendido_Real (R$)	Qtde_Vendida	Custo_Real (R$)	Lucro (%)
T100000	L100000	P100000	3,50	35	1,05	95
T100000	L100000	P100001	2,70	3	2,10	47
T100000	L100000	P100002	9,00	3	6,60	15
T100000	L100001	P100000	10,00	100	3,00	33
T100000	L100001	P100001	4,50	5	3,50	28
T100000	L100001	P100002	3,00	1	2,20	45
T100001	L100000	P100000	10,50	105	3,15	31
T100001	L100000	P100001	9,00	10	7,00	14
T100001	L100001	P100000	20,00	200	6,00	16
T100001	L100001	P100001	8,10	9	6,30	15
T100001	L100001	P100002	18,00	6	13,20	7,5
T100002	**L100000**	**P100000**	**20,00**	**200**	**6,00**	**16**

T100002	**L100000**	**P100001**	**40,50**	**45**	**31,50**	**3,17**
T100002	**L100000**	**P100002**	**15,00**	**5**	**11,00**	**9**
T100002	**L100001**	**P100000**	**30,00**	**300**	**9,00**	**11**
T100002	**L100001**	**P100001**	**12,60**	**14**	**9,80**	**10,2**
T100002	**L100001**	**P100002**	**33,00**	**11**	**24,20**	**4,1**
T100002	**L100002**	**P100000**	**35,00**	**700**	**21,00**	**4,7**
T100002	**L100002**	**P100001**	**22,50**	**25**	**17,50**	**5,7**
T100002	**L100002**	**P100002**	**24,00**	**8**	**17,60**	**5,7**
T100002	**L100002**	**P100003**	**16,20**	**12**	**10,80**	**9,2**

Tabela 10 Tabela de Fatos TfVendas atualizada

As linhas em negrito representam os novos dados de `TfVendas`. Note, que ela representa a união de seus valores anteriores e `tMovimento`, com os devidos agrupamentos de atributos, ou seja, cada linha de *TfVendas* representa uma agrupamento dos valores de `tMovimento` para cada valor de chave. Agora, considere novamente a definição da visão `vmVendas_por_Produto_Out_1999_Iguatemi` da Figura 6:

```
CREATE VIEW
vmVENDAS_POR_PRODUTO_OUT_1999_IGUATEMI
AS
SELECT DESCRIÇÃO_DO_PRODUTO AS PRODUTO,
SUM(QTDE_VENDIDA) AS VENDAS
FROM TdLOJA, TfVENDAS, TdPRODUTO,
TdTEMPO
WHERE TdLOJA.CHAVE_LOJA =
TfVENDAS.CHAVE_LOJA AND
TdLOJA.NOME_DA_LOJA LIKE
'%'||'IGUATEMI'||'%' AND
TdTEMPO.CHAVE_TEMPO =
TfVENDAS.CHAVE_TEMPO AND TdTEMPO.MÊS =
'OUTUBRO' AND TdTEMPO.ANO = '1999' AND
TdPRODUTO.CHAVE_PRODUTO =
TfVENDAS.CHAVE_PRODUTO
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 14 Definição da visão vmVendas_por_Produto_Out_1999_Iguatemi

Como a visão acima não pode obtida a partir de outra, então será necessário criar uma tabela auxiliar para ela, e portanto, ela é dependente da tabela de fatos. Assim, cinsidere a Tabela 21 em Anexo.

Note que a tabela auxiliar apresenta os mesmos atributos da visão, não apresenta agrupamento e os dados provêm da tabela `tMovimento`. Agora mostramos as tabelas de associação e dependência criadas:

Visão	Tabela Auxiliar
vmVendas_por_Produto_Out_1999_Iguatemi	taux_vmVendas_por_Produto_Out_1999_Iguatemi

Tabela 11 taaTabela_Associação_Auxiliar

Visão	Tabela Fatos/Tabela Auxiliar
vmVendas_por_Produto_Out_1999_Iguatemi	TfVendas

Tabela 12 tdvTabela_Dependência_Visão

Agora, o algoritmo *Atualiza_DW* entra na próxima etapa.

Etapa Propagar: Com a tabela auxiliar da visão criada, o algoritmo irá criar a partir da tabela auxiliar a visão delta-sumário.

Assim, temos:

```
CREATE VIEW
DELTA_vmVENDAS_POR_PRODUTO_OUT_1999_IGU
ATEMI (PRODUTO, VENDAS) AS
SELECT DESCRIÇÃO_DO_PRODUTO AS PRODUTO,
SUM(QTDE_VENDIDA) AS VENDAS FROM
```

```
TAUX_VMVENDAS_POR_PRODUTO_OUT_1999_IGUA
TEMI
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 15 Definição de delta-sumário para a visão vmVendas_por_Produto_Out_1999_Iguatemi

Note que, processar esta visão a partir da tabela auxiliar é bem mais rápido do que processá-la a partir de tMovimento. Daí, temos:

Produto	Vendas
Pão 50grs	1200
Leite	84
Iogurte	24
Manteiga	12

Tabela 13 deltavmVendas_por_Produto_Out_1999_Iguatemi

Finalmente, o algoritmo passa para a última etapa Aplicar.

Etapa Aplicar: Como a visão delta-sumário foi criada, o algoritmo irá percorrer os registros dessa visão, procurando sua ocorrência na visão. Caso o registro seja encontrado, o valor de Vendas será atualizado, caso contrário ele será inserido na visão.

Note que os três primeiros registros de deltavmVendas_por_Produto_Out_1999_Iguatemi também estão presentes em vmVendas_por_Produto_Out_1999_Iguatemi,

enquanto que o último registro representa um valor novo para a visão. Assim, temos:

Produto	Vendas
Pão 50grs	**1640**
Leite	**111**
Iogurte	**34**
Manteiga	**12**

Tabela 14 vmVendas_por_Produto_Out_1999_Iguatemi atualizada

5.1.4.1 Um Caso Especial[11]

Um caso especial do algoritmo para criação e carga da tabela auxiliar está na presença das funções de agregação MIN e MAX. Considere novamente a definição da visão `vmÚltimas_Vendas _Iguatemi-JPessoa` da Figura 6:

```
CREATE VIEW vmÚLTIMAS_VENDAS_IGUATEMI-
JPESSOA AS SELECT DESCRIÇÃO_DO_PRODUTO
AS PRODUTO, MAX(CHAVE_TEMPO) AS
ÚLTIMADATA, SUM(QTDE_VENDIDA) AS VENDAS
FROM TfVENDAS, TdPRODUTO, TdLOJA
WHERE TfVENDAS.CHAVE_LOJA =
TdLOJA.CHAVE_LOJA
AND TdLoja.NOME_DA_LOJA = 'IGUATEMI-
JPESSOA' AND TfVENDAS.CHAVE_PRODUTO =
TdPRODUTO.CHAVE_PRODUTO
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 16 Definição da visão vmÚltimas_Vendas_Iguatemi-JPessoa

A tabela auxiliar criada, chamada `taux_vmÚltimas_Vendas _Iguatemi-JPessoa`, será um

[11] O caso especial refere-se ao tratamento das funções MIN e MAX. Outras funções de agregação não mencionadas não puderam ser tratadas por questões de tempo.

pouco diferente, pois o algoritmo trata aqui os campos envolvidos nas demais funções agregadas. Assim, ela será constituída dos atributos `Descrição_do_Produto` e data. Após sua criação, estará pronta para receber novos dados.

Dessa forma, para cada nova tupla, o algoritmo testa se o valor do campo `data` é maior que o maior valor de `data` já existente na tabela auxiliar, considerando o índice criado com o atributo `Descrição_do_Produto`, que é chave da tabela original e especificado na cláusula *GROUP BY*. Se o valor for maior, ele o substitui pela data testada, caso contrário, ignora o registro.

Agora, o algoritmo trata o campo com a função agregada SUM, no caso `qtde_vendida`. Ele não aparece na tabela auxiliar da visão por que o algoritmo cria uma tabela auxiliar para ele, composta da chave da tabela auxiliar da visão, e pelo campo `qtde_vendida`, a qual chamaremos `taux_sum_qtde_vendida_vmÚltimas_Vendas_Iguatemi-JPessoa`. Assim, as funções MIN e MAX podem ser tratadas sem perder a integridade das demais funções que aparecem na definição da visão e, o mais importante, sem perder a estrutura

do modelo relacional, ou seja, serão utilizados os próprios recursos da álgebra relacional para encontrar o valor do atributo.

Para melhor entender a explicação acima, iremos partir da visão `vmÚltimas_Vendas_Iguatemi-JPessoa` materializada. Veremos uma simulação de inserção de novos dados fontes e ainda, a criação das tabelas mencionadas acima. Dessa forma, temos a visão `vmÚltimas_Vendas_Iguatemi-JPessoa` materializada, de acordo com a tabela `tMovimento` apresentada anteriormente:

Produto	ÚltimaData	Vendas
Pão 50grs	20/10/1999	200
Leite	20/10/1999	9
Iogurte	20/10/1999	6

Tabela 15 vmÚltimas_Vendas_Iguatemi-JPessoa

Como a visão representa sempre as últimas vendas, os novos registros de `tMovimento` para a loja Iguatemi-JPessoa serão todos utilizados para manter a visão, exceto se existir algum valor de data ainda maior que 21/10/1999. Dessa forma as tabelas auxiliares (ver Tabela 18, pág. 121 e Tabela 19, pág. 122) criadas

para a visão e para o campo terão os registros apresentados nas Tabelas 18 e 19:

Produto	Ultimadata
Pão 50grs	21/10/1999
Leite	21/10/1999
Iogurte	21/10/1999

Tabela 16 tauxvmÚltimas_Vendas_Iguatemi-JPessoa

Produto	ÚltimaData	Qtde_Vendida
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	10
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Pão 50grs	21/10/1999	5
Leite	21/10/1999	1
Leite	21/10/1999	1
Leite	21/10/1999	1
Leite	21/10/1999	1

Leite	21/10/1999	1
Leite	21/10/1999	1
Leite	21/10/1999	1
Leite	21/10/1999	1
Leite	21/10/1999	2
Leite	21/10/1999	2
Leite	21/10/1999	2
Iogurte	21/10/1999	1
Iogurte	21/10/1999	1
Iogurte	21/10/1999	1
Iogurte	21/10/1999	1
Iogurte	21/10/1999	1
Iogurte	21/10/1999	2
Iogurte	21/10/1999	2
Iogurte	21/10/1999	2

Tabela 17 taux_sum_qtde_vendida_ vmÚltimas_Vendas_Iguatemi-JPessoa

Note que não há necessidade na etapa da função *Propagar* de processar a função agregada MAX, pois é certo que a tabela auxiliar possui o maior valor para a chave. O algoritmo cria a visão delta-sumário e irá processar apenas as demais funções agregadas, no caso SUM aplicada ao atributo `qtde_vendida`, de acordo com o processo de criação de tabela auxiliar utilizado. Depois disso, ele faz junções entre a tabela auxiliar da visão e a tabela auxiliar do campo `qtde_vendida` para processar corretamente o valor da função SUM. Daí, temos a definição de delta-sumário :

```
CREATE VIEW
DELTAvmÚLTIMAS_VENDAS_IGUATEMI-JPESSOA
(PRODUTO, ÚLTIMADATA, VENDAS) AS
SELECT DESCRIÇÃO_DO_PRODUTO AS PRODUTO,
ÚLTIMADATA, SUM(QTDE_VENDIDA) AS VENDAS
```

```
FROM TAUX_vmÚLTIMAS_VENDAS_IGUATEMI-
JPESSOA V,
TAUX_SUM_QTDE_VENDIDA_vmÚLTIMAS_VENDAS_
IGUATEMI-JPESSOA AS S
WHERE V.PRODUTO = S.PRODUTO AND
V.ÚLTIMADATA = S.ÚLTIMADATA
GROUP BY DESCRIÇÃO_DO_PRODUTO
```

Código 17 Definição de delta-sumário para a visão vmÚltimas_Vendas_Iguatemi-JPessoa

Note que aparecem as cláusulas WHERE para fazer a união entre as tabelas auxiliares e, *GROUP BY* para o algoritmo computar o somatório do agrupamento por produto.

Assim, temos a Tabela 18:

Produto	ÚltimaData	Vendas
Pão 50grs	21/10/1999	300
Leite	21/10/1999	14
Iogurte	21/10/1999	11

Tabela 18 deltavmÚltimas_Vendas _Iguatemi-JPessoa

A partir deste momento, o algoritmo entra na etapa Aplicar. O algoritmo irá percorrer os registros da visão delta-sumário, considerando-se a existência de um índice para a visão com o atributo `Descrição_do_Produto`. Caso o registro seja encontrado, o algoritmo testa se o valor de data de delta-sumário é maior que o valor de data da visão, se sim, o valor de data e ven-

das são substituídos pelos respectivos valores de delta-sumário. Caso contrário, há a possibilidade desse valor ser igual, e nesse caso, o valor de vendas é atualizado. No nosso caso, não haverá a possibilidade de valores de datas iguais, pois todos os valores de delta-sumário são maiores que os da visão. Isso por que o DW e suas visões foram atualizados no dia 20/10/1999 e agora estamos demonstrando sua atualização no dia 21/10/1999, portanto a `visão vmÚltimas_Vendas_Iguatemi-JPessoa` receberá apenas dados de uma nova data e no caso maior que a existente. Dessa forma, temos a Tabela 19:

Produto	ÚltimaData	Vendas
Pão 50grs	21/10/1999	300
Leite	21/10/1999	14
Iogurte	21/10/1999	11

Tabela 19 vmÚltimas_Vendas_Iguatemi-JPessoa atualizada

5.2 Conclusões

Apresentamos o nosso algoritmo cujo objetivo principal é manter o DW e suas VMs de forma eficiente e procurando reduzir ao máximo o tempo de manutenção e consequentemente o tempo em que o DW e suas VMs ficam indisponíveis para as consultas dos usuários. No próximo capítulo mostraremos al-

guns detalhes de implementação de um protótipo para o algoritmo *Atualiza_DW*.

Capítulo

6 Conclusões e Perspectivas

Nesta dissertação, apresentamos um algoritmo para atualização eficiente de um DW e suas visões. Os conceitos de atualização incremental e filtro de dados relevantes são empregados em nosso algoritmo. Além da reutilização de processamento, que evita realização de um processamento que pode ser obtido a partir do que já foi processado. O volume de dados entre uma atualização e outra é inferior à quantidade de dados existentes no DW. Considerando as definições das visões, apenas alguns desses dados afetam realmente o estado das visões.

Mais, se uma visão pode ser obtida a partir de outra, algum trabalho realizado para se chegar a visão atualizada será utilizado para atualizar aquela que dela pode ser obtida. Por exemplo, na estrutura do DW de vendas (ver Figura 3, pág. 35), utilizado como exemplo nessa dissertação, foi materializada uma consulta quinzenal de vendas de um determinado produto, em decorrên-

cia do levantamento das necessidades da empresa. Depois de um certo período observou-se que seriam necessários alguns dados de vendas mensais daquele produto.

Geralmente, as visões são definidas a partir da tabela de fatos, exceto quando há explicitamente em sua definição uma outra visão. Portanto, não será necessário atualizar a segunda visão (dados de vendas mensais) a partir da tabela de fatos, e sim, a partir da primeira, apenas realizando uma consolidação dos dados mensais, através da uma computação das duas quinzenas do mês.

Quando falamos em atualização eficiente, devemos entender o seguinte: por um lado, como as tabelas de um DW relacional normalmente são muito grandes, se cada atualização for considerar todos os dados fontes, o tempo de processamento pode ser inaceitável, visto que, durante esse período, o DW fica indisponível para as consultas dos usuários. Para reduzir esse tempo, o algoritmo utiliza apenas os novos dados inseridos nas fontes de dados, desde a última atualização. Também realiza a filtragem nesses dados para trabalhar apenas com aqueles que

afetam o estado das visões, conseguindo com isto reduzir o tempo global de atualização do DW e suas visões.

Por outro lado, se cada visão tiver que ser atualizada individualmente, a partir da tabela de fatos, não chegaríamos a um resultado satisfatório. Assim, as visões também são atualizadas a partir dos dados fontes. Ademais, se uma determinada visão puder ser obtida a partir de uma outra, mais rápida será sua atualização. Observe que, se a tabela de fatos e as visões são atualizadas a partir dos dados fontes, então esse processamento poderia ser realizado em paralelo, podendo chegar ao final com todas as tabelas atualizadas, dependendo, é claro, do tamanho das mesmas.

O algoritmo *Atualiza_DW* considera os dois pontos acima citados. Ele recebe como entrada uma tabela chamada `tMovimento` que contém os novos dados inseridos nas fontes desde a última atualização do DW.

Como saída, ele apresenta o DW atualizado, ou seja, a tabela de fatos e as visões representando o estado atual do contexto do DW. A partir daí, os usuários podem submetê-lo a con-

sultas e já verão refletidos em seus resultados as mudanças ocorridas nas fontes de dados que "alimentam" o DW.

6.1 Trabalhos Futuros

Como um dos objetivos de nosso trabalho foi desenvolver um algoritmo eficiente para atualização de um DW e suas visões, uma proposta para futuros trabalhos é realizar um estudo para o tratamento de outras funções agregadas, além de SUM e COUNT, além de uma análise mais rigorosa para o tratamento dados as funções MIN e MAX.

Em termos de perspectivas de nosso trabalho, vislumbramos duas linhas de investigação. A primeira delas diz respeito à necessidade de realizar um grande número de testes com o algoritmo, baseadas em casos próximos da realidade, para se determinar estatisticamente o quão satisfatória a solução apresentada pelo algoritmo é. A partir daí, podemos pesquisar novas estratégias de otimização do algoritmo. A segunda diz respeito à execução em paralelo que o algoritmo permite. Uma alternativa é submeter o algoritmo a uma máquina com dois ou mais proces-

sadores para acompanhar o desempenho do mesmo, já que não foi possível testá-lo em uma máquina com essa característica.

6.2 Considerações Finais

Realizamos um trabalho sobre atualização de DW e suas visões. Outros trabalhos nesta área têm usado várias estratégias: um método realiza o processamento em batch, utilizando os novos dados fontes, porém não realiza nenhum tipo de filtro nesses dados para atualizar as visões, o que faz o algoritmo trabalhar com uma grande quantidade de dados. Outro método realiza um filtro nos dados das fontes para atualizar uma visão, porém essa atualização é feita imediatamente a ocorrência de uma mudança nas fontes de dados, o que pode acarretar atraso e inconsistências nos resultados das consultas dos usuários.

Acreditamos que nosso algoritmo será de grande relevância para futuros projetos de desenvolvimento de ferramentas de administração de dados de DWs. Certamente nosso algoritmo integrando uma ferramenta que trabalhe em conjunto com um SGBD, poderá ser muito útil no desenvolvimento de DWs, proporcionando uma maior satisfação tanto do usuário final que

esteja interagindo fazendo consultas OLAP, como do administrador do sistema, que não irá precisar dedicar grande parte de seu tempo para atualizar o DW que gerencia.

Anexos

A

Pseudocódigo do Algoritmo *Atualiza_DW*

Algoritmo *Preparar*
Atualiza a tabela de fatos e cria as tabelas auxiliares a partir da tabela com os novos dados fontes, chamada tMovimento. Além das tabelas de associação e de dependências, se for o caso.
Entrada : conjunto de registros *r* de tMovimento
Definição da visão *v*, C=Condição de Seleção, P=Lista de atributos Projetados e J=União entre tabelas
Saída : tabela de fatos atualizada, tabelas auxiliares de visões taux_vm e campos tab_aux f_v e tabelas de associação e dependências.

```
Atua_Fatos()
Cria_Taa();
Cria_Tdv();
Para cada visão v do DW
        Se P possui função MIN ou MAX
                caso_especial := True;
                Se P possui outras funções
                        outras_funções := True;
                        cria taux_v, sem campos das demais funções presentes em P
                        cria taux_f_v para cada campo das demais funções
                Senão
                        outras_funções := False;
                        cria taux_v com os atributos presentes em P;
                FimSe
        Senão
                caso_especial := False;
                cria taux_v com todos os atributos presentes em P
```

```
                FimSe
                //preencher tabela auxiliar
                Se caso_especial
                        Se C <> {}
                                substitui valores de r correspondentes em C
                                Se C e a função MIN/MAX forem satis-
                                feitas
                                        insere r em taux_v, substituindo
                                        valor existente para MIN/MAX
                                        Se outras_funções
                                                insere registro corres-
                                                pondente em taux_fv
                                        FimSe
                                Senão
                                        ignora r
                                FimSe
                        Senão
                                Se função MIN/MAX for satisfeita
                                        insere r em taux_v, substituindo
                                        valor existente para MIN/MAX
                                        Se outras_funções
                                                insere registro corres-
                                                pondente em taux_fv
                                        FimSe
                                Senão
                                        ignora r
                                FimSe
                        FimSe
                Senão
                        Se C <> {}
                                substitui valores de r correspondentes em C
                                Se C for satisfeita
                                        insere r em taux_v
                                Senão
                                        processa demais funções
                                FimSe
                        Senão
                                Se r possui valores de alguma tabela em J
                                        insere r em taux_v
                                Senão
                                        ignora r
                                FimSe
                        FimSe
                FimSe
FimLoop
```

Algoritmo *Cria_Taa*

A partir das definições das visões, cria tabela de associação.
Entrada : Definições das visões *d_v*
Saída : Taa_Tabela_Associação_Auxiliar

```
Se Taa_Tabela_Associação_Auxiliar não existe
        Cria a estrutura de Taa_Tabela_Associação_Auxiliar;
Fim Se
//Povoando Taa_Tabela_Associação_Auxiliar
Para cada visão v do DW
        Se v é obitda apenas de fatos
                nome_visão := v;
                nome_tab_aux := taux_v;
        Senão
                procura visão v´ a partir da qual v pode ser obtida
                nome_visao := v;
                nome_tab_aux := taux_v´;
```

```
        Fim Se
FimLoop
```

Algoritmo *Cria_Tdv*

A partir das definições das visões, cria tabela de dependência.
Entrada : Taa_Tabela_Associação_Auxiliar
Saída : Tdv_Tabela_Dependência_Visão

```
Se Tdv_Tabela_Dependência_Visão não existe
        Cria a estrutura de Tdv_Tabela_Dependência_Visão;
Fim Se
//Povoando Tdv_Tabela_Dependência_Visão
Para cada visão v do DW
        Se taux é uma tabela auxilliar criada para v
                nome_visao := v;
                nome_fatos_ou_visão := fatos;
        Senão
                nome_visao := v
                nome_fatos_ou_visão := Taa.taux_v;
        Fim Se
FimLoop
```

Algoritmo *Atua_Fatos*

Atualiza a tabela de fatos do DW.
Entrada : Tabela tMovimento
Saída : Tabela de Fatos (atualizada)

```
Se fatos possui agrupamento com relação a tMovimento
        Faz agrupamento de acordo com chave de fatos
        Cria novo registro em fatos
Senão
        Para registro r da tabela tMovimento
                Cria novo registro em fatos
Fim Se
```

Algoritmo *Propagar*

Cria as visões delta-sumário.
Entrada : tabela auxiliar de visão taux_v/tabela(s) auxiliar(es) de campos taux_f_v , tabela de associação e de dependências.
Saída : visão delta-sumário Δ

```
Se caso_especial
        aparece MIN/MAX em P da visão delta-sumário
        aparece(m) a(s) tabela(s) auxiliar(es) na cláusula FROM
        aparece a cláusula WHERE para fazer a união entre tabelas auxiliares
        cria visão delta-sumário Δ a partir de v, respeitando asserções acima
Senão
        não aparece MIN/MAX em P da visão delta-sumário
        não aparece(m) a(s) tabela(s) auxiliar(es) na cláusula FROM
        cria visão delta-sumário Δ a partir de v, respeitando asserções acima
FimSe
```

Algoritmo *Aplicar*

Percorre a tabela delta-sumário para encontrar o registro na visão correspondente.
Entrada : tabela delta-sumário Δ de visão v
visão materializada atual v_{old}

Saída : visão materializada v_{new} (atualizada)

Para cada registro r de Δ
Se encontrar valor de chave correspondente em v_{old}
    Se caso_especial
        Se valor de função MIN/MAX é menor/maior que valor existente em v_{old}
            substitui valor correspondente
    FimSe
    Processa demais funções, atualizando valores
Senão
    Insere r em v_{new}
FimSe

B

Tabelas tMovimento e tauxvmVendas_por_Produto_Out_1999_Iguatemi

Registro	Loja	Produto	Data	Qtde_Vendida	Valor_Vendido (R$)	Valor_Custo (R$)
...	...	...	...	...	...	...
10000000	Iguatemi-Salvador	Pão	19/10/1999	5	0,50	0,15
10000001	Iguatemi-Salvador	Pão	19/10/1999	10	1,00	0,30
10000002	Iguatemi-Salvador	Pão	19/10/1999	10	1,00	0,30
10000003	Iguatemi-Salvador	Pão	19/10/1999	10	1,00	0,30
10000004	Iguatemi-Salvador	Leite	19/10/1999	2	1,80	1,40
10000005	Iguatemi-Salvador	Leite	19/10/1999	1	0,90	0,70
10000006	Iguatemi-Salvador	Iogurte	19/10/1999	1	3,00	2,20
10000007	Iguatemi-Salvador	Iogurte	19/10/1999	1	3,00	2,20
10000008	Iguatemi-Salvador	Iogurte	19/10/1999	1	3,00	2,20
10000009	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000010	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000011	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000012	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000013	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000014	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000015	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000016	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000017	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000018	Iguatemi-JPessoa	Pão	19/10/1999	10	1,00	0,30
10000019	Iguatemi-JPessoa	Leite	19/10/1999	1	0,90	0,70
10000020	Iguatemi-JPessoa	Leite	19/10/1999	1	0,90	0,70
10000021	Iguatemi-JPessoa	Leite	19/10/1999	1	0,90	0,70
10000022	Iguatemi-JPessoa	Leite	19/10/1999	1	0,90	0,70
10000023	Iguatemi-JPessoa	Leite	19/10/1999	1	0,90	0,70
10000024	Iguatemi-JPessoa	Iogurte	19/10/1999	1	3,00	2,20

10000025	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000026	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000027	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000028	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000029	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000030	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000031	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000032	Iguatemi-Salvador	Pão	20/10/1999	10	1,00	0,30
10000033	Iguatemi-Salvador	Pão	20/10/1999	5	0,50	0,15
10000034	Iguatemi-Salvador	Pão	20/10/1999	5	0,50	0,15
10000035	Iguatemi-Salvador	Pão	20/10/1999	5	0,50	0,15
10000036	Iguatemi-Salvador	Pão	20/10/1999	5	0,50	0,15
10000037	Iguatemi-Salvador	Pão	20/10/1999	5	0,50	0,15
10000038	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000039	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000040	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000041	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000042	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000043	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000044	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000045	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000046	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000047	Iguatemi-Salvador	Leite	20/10/1999	1	0,90	0,70
10000048	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000049	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000050	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000051	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000052	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000053	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000054	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000055	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000056	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000057	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000058	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000059	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000060	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000061	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000062	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000063	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000064	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000065	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000066	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000067	Iguatemi-JPessoa	Pão	20/10/1999	10	1,00	0,30
10000068	Iguatemi-JPessoa	Leite	20/10/1999	2	1,80	1,40
10000069	Iguatemi-JPessoa	Leite	20/10/1999	2	1,80	1,40
10000070	Iguatemi-JPessoa	Leite	20/10/1999	1	0,90	0,70
10000071	Iguatemi-JPessoa	Leite	20/10/1999	1	0,90	0,70
10000072	Iguatemi-JPessoa	Leite	20/10/1999	1	0,90	0,70
10000073	Iguatemi-JPessoa	Leite	20/10/1999	1	0,90	0,70
10000074	Iguatemi-JPessoa	Leite	20/10/1999	1	0,90	0,70
10000075	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000076	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000077	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000078	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000079	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000080	Iguatemi-JPessoa	Iogurte	20/10/1999	1	3,00	2,20
10000081	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000082	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000083	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000084	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000085	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000086	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000087	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000088	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000089	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000090	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000091	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000092	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000093	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000094	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30
10000095	Iguatemi-Salvador	Pão	21/10/1999	10	1,00	0,30

10000096	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000097	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000098	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000099	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000100	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000101	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000102	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000103	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000104	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000105	Iguatemi-Salvador	Pão	21/10/1999	5	0,50	0,15
10000106	Iguatemi-Salvador	Leite	21/10/1999	3	2,70	2,10
10000107	Iguatemi-Salvador	Leite	21/10/1999	3	2,70	2,10
10000108	Iguatemi-Salvador	Leite	21/10/1999	3	2,70	2,10
10000109	Iguatemi-Salvador	Leite	21/10/1999	3	2,70	2,10
10000110	Iguatemi-Salvador	Leite	21/10/1999	3	2,70	2,10
10000111	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000112	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000113	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000114	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000115	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000116	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000117	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000118	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000119	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000120	Iguatemi-Salvador	Leite	21/10/1999	2	1,80	1,40
10000121	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000122	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000123	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000124	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000125	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000126	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000127	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000128	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000129	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000130	Iguatemi-Salvador	Leite	21/10/1999	1	0,90	0,70
10000131	Iguatemi-Salvador	Iogurte	21/10/1999	1	3,00	2,20
10000132	Iguatemi-Salvador	Iogurte	21/10/1999	1	3,00	2,20
10000133	Iguatemi-Salvador	Iogurte	21/10/1999	1	3,00	2,20
10000134	Iguatemi-Salvador	Iogurte	21/10/1999	1	3,00	2,20
10000135	Iguatemi-Salvador	Iogurte	21/10/1999	1	3,00	2,20
10000136	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000137	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000138	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000139	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000140	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000141	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000142	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000143	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000144	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000145	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000146	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000147	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000148	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000149	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000150	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000151	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000152	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000153	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000154	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000155	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000156	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000157	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000158	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000159	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000160	Iguatemi-JPessoa	Pão	21/10/1999	10	1,00	0,30
10000161	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000162	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000163	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000164	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000165	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000166	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15

10000167	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000168	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000169	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000170	Iguatemi-JPessoa	Pão	21/10/1999	5	0,50	0,15
10000171	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000172	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000173	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000174	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000175	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000176	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000177	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000178	Iguatemi-JPessoa	Leite	21/10/1999	1	0,90	0,70
10000179	Iguatemi-JPessoa	Leite	21/10/1999	2	1,80	1,40
10000180	Iguatemi-JPessoa	Leite	21/10/1999	2	1,80	1,40
10000181	Iguatemi-JPessoa	Leite	21/10/1999	2	1,80	1,40
10000182	Iguatemi-JPessoa	Iogurte	21/10/1999	1	3,00	2,20
10000183	Iguatemi-JPessoa	Iogurte	21/10/1999	1	3,00	2,20
10000184	Iguatemi-JPessoa	Iogurte	21/10/1999	1	3,00	2,20
10000185	Iguatemi-JPessoa	Iogurte	21/10/1999	1	3,00	2,20
10000186	Iguatemi-JPessoa	Iogurte	21/10/1999	1	3,00	2,20
10000187	Iguatemi-JPessoa	Iogurte	21/10/1999	2	6,00	4,40
10000188	Iguatemi-JPessoa	Iogurte	21/10/1999	2	6,00	4,40
10000189	Iguatemi-JPessoa	Iogurte	21/10/1999	2	6,00	4,40
10000190	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000191	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000192	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000193	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000194	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000195	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000196	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000197	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000198	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000199	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000200	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000201	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000202	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000203	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000204	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000205	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000206	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000207	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000208	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000209	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000210	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000211	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000212	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000213	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000214	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000215	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000216	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000217	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000218	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000219	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000220	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000221	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000222	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000223	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000224	Iguatemi-CGrande	Pão	21/10/1999	10	0,50	0,30
10000225	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000226	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000227	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000228	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000229	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000230	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000231	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000232	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000233	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000234	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000235	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000236	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000237	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45

10000238	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000239	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000240	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000241	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000242	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000243	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000244	Iguatemi-CGrande	Pão	21/10/1999	15	0,75	0,45
10000245	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000246	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000247	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000248	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000249	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000250	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000251	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000252	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000253	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000254	Iguatemi-CGrande	Pão	21/10/1999	5	0,25	0,15
10000255	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000256	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000257	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000258	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000259	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000260	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000261	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000262	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000263	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000264	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000265	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000266	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000267	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000268	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000269	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000270	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000271	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000272	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000273	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000274	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000275	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000276	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000277	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000278	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000279	Iguatemi-CGrande	Leite	21/10/1999	1	0,90	0,70
10000280	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000281	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000282	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000283	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000284	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000285	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000286	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000287	Iguatemi-CGrande	Iogurte	21/10/1999	1	3,00	2,20
10000288	Iguatemi-CGrande	Manteiga	21/10/1999	2	2,70	1,80
10000289	Iguatemi-CGrande	Manteiga	21/10/1999	2	2,70	1,80
10000290	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000291	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000292	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000293	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000294	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000295	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000296	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90
10000297	Iguatemi-CGrande	Manteiga	21/10/1999	1	1,35	0,90

Tabela 20 tMovimento

Descrição_do_Produto	qtde_vendida
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10

Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Leite	3
Leite	3
Leite	3
Leite	3
Leite	3
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	2
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10

Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	2
Leite	2
Leite	2
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	2
Iogurte	2
Iogurte	2
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10

Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	10
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	15
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Pão 50grs	5
Leite	1

Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Leite	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Iogurte	1
Manteiga	2
Manteiga	2
Manteiga	1
Manteiga	1
Manteiga	1
Manteiga	1
Manteiga	1
Manteiga	1
Manteiga	1
Manteiga	1

Tabela 21 tauxvmVendas_por_Produto_Out_1999_Iguatemi

Bibliografia

1 ALBERT, A. Y. T. Advanced Topics in database Systems. **Course Project : Study of Data Warehousing**. Departament of Computer Science, The Chinese University of Hong Kong.

2 BARBIERI, C. Uma Abordagem Sobre a Modelagem Dimensional de Dados. **Developers' Magazine,** p. 36-40, 1997.

3 BLAKELEY, LARSON, J. A., PER-AKE, TOMP, WM, F. Efficiently Updating Materialized Views. **Data Structuring Group** Departament of Computer Science, University of Waterloo, Waterloo.

4 CHAIDHURI, S., DAYAL, U. An Overview of Data Warehousing and OLAP Technology. http://www.microsoft.com.

5 EDELSTEIN, H., BARQUIN, R. C. An Intoduction to Data Warehousing. **Planning and Designing the Data Warehouse**. New Jersey, Chap. 3, p. 31-50.

6 FIGUEIREDO, A. M. C. M. MOLAP X ROLAP : Embate de Tecnologias para Data Warehouse. **Developers' Magazine,** p. 24-25, 1998.

7 FILHO, T. R. M. On-Line Analytical Processing Server (Servidor OLAP). **Developers' Magazine,** p. 28-29, 1998.

8 HUYN, N. Efficient View Self-Maintenance. **Proceedings of the ACM Worshop on Materialized Views : Techniques and Applications**. Montreal, Junho, 1997.

9 HUYN, N. Multiple-View Self-Maintenance in Data Warehousing Enviroments. **To appear in the Proceedings of the 23rd VLDB Conference**. Athens 1997.

10 KIMBALL, R., WILEY, J., AND SONS, INC. The Data Warehouse Toolkit. 1996.

11 KONDRATIUK, E. R. Data Warehouse : Detalhes que fazem a diferença. **Developers' Magazine,** 22p, 1998.

12 LABIO, W. J., ZHUGE, Y., WIENER, J. L., GUPTA, H., GARCIA-MOLINA, H., WIDOM, J. The WHIPS Prototype for data Warehouse Creation and Maintenance. Departament of Computer Science, Stanford University.

13 MANNI, L. C., DORSA, L. F. A. Data Warehouse : Gerenciando a Qualidade dos Dados. **Developers' Magazine,** 20p, 1998.

14 MUMICK, I., QUASS, D., MUMICK, B. Maintenance of Data Cubes and Summary Tables in a Warehouse. **Proceedings of the ACM SIGMOD Conference**, Tuscon, Maio, 1997. http://www-db.stanford.edu/pub/papers/

15 NIMER, F. Analisando o retorno sobre o investimento de Data Warehouse. **Developers' Magazine,** p. 16-17, 1998.

16 PALMA, S. Os Componentes funcionais de um Data Warehouse. **Developers' Magazine,** p. 18-19, 1998.

17 QUASS, D. Maintenance Expressions for Views with Aggregation. **Proceedings of the ACMO Workshop on Materialized Views : Techniques and Applications**, Canada, Junho, 1996.

18 QUASS, D., WINDOM, J. On-Line Warehouse View Maintenance for Batch Updates. **Proceedings of the ACM SIGMOD Conference**, Tuscon, Maio, 1997.

19 ROUSSOPOULOS, N. Materialized Views and Data Warehouses. Departament of Computer Science and Institute of Advanced Computer Studies, University of Maryland.

20 SAMPAIO, M. C. Data Warehouse – Uma solução para a Integração de Múltiplos Sistemas de Bancos de Dados. **Curso da discipliba T.E.I. (Data Warehouse), lecionada no período 97.2, Mestrado em Informática**, Universidade Federal da Paraíba, Campus II, Campina Grande, 1997.

21 SOUZA, Márcio Farias de, Materialização/Utilização Eficientes de Visões Materializadas em Data Warehouses. Dissertação de Mestrado, Universidade Federal da Paraíba, Campus II, Campina Grande, 1999.

22 SULAIMAN, A., SOUZA, J. M. Técnicas de Modelagem de Dados em Projetos de Data Warehouse. **Developers' Magazine,** p. 12-114, 1998.

23 TAURION, C. Data Warehouse : Vale a pena gastar milhões investindo em um ? **Developers' Magazine,** p. 10-11, 1998.

24 TAURION, C. O Data Warehouse será útil para a sua organização ? **Developers' Magazine,** p. 26-27, 1998.

25 WIDOM, J. Research Problems in Data Warehousing, Departament of Computer Science, Stanford University.

26 WIENER, J. L., GUPTA, H., LABIO, W. J., ZHUGE, Y., GARCIA-MOLINA, H., WIDOM, J. A System Prototype for Warehouse View Maintenance. Departament of Computer Science, Stanford University. http://www-db.stanford.edu/warehousing/warehouse.html

27 ZHUGE, Y., GARCIA-MOLINA, H., WIENER, J. L. Consistency Algorithms for Multi-Source Warehouse View Maintenance. Departament of Computer Science, Stanford University.

28 ZHUGE, Y., GARCIA-MOLINA, H., HAMMER, J., WIDOM, J. View Maintenance in a Warehousing Enviroment. **Proceedings of the ACM SIGMOD Conference**, San Jose, Maio, 1995.