UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

MESTRADO EM INFORMÁTICA

DISSERTAÇÃO DE MESTRADO

PROJETO E CONSTRUÇÃO DE UM GERENTE DE DATA WAREHOUSES MÓVEIS

PLÁCIDO MARINHO DIAS
(MESTRANDO)

MARCUS COSTA SAMPAIO
&
CLÁUDIO DE SOUZA BAPTISTA
(ORIENTADORES)

CAMPINA GRANDE - PB
FEVEREIRO DE 2003

PLÁCIDO MARINHO DIAS

PROJETO E CONSTRUÇÃO DE UM GERENTE DE DE DATA WAREHOUSES MÓVEIS

DISSERTAÇÃO apresentada à COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA do DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO do CENTRO DE CIÊNCIAS E TECNOLOGIA da UNIVERSIDADE FEDERAL DE CAMPINA GRANDE como requisito parcial para obtenção do grau de MESTRE EM INFORMÁTICA.

Área de concentração:	CIÊNCIA DA COMPUTAÇÃO
Linha de Pesquisa:	SISTEMAS DE INFORMAÇÃO E BANCO DE DADOS
Orientador:	PROF. DR. MARCUS COSTA SAMPAIO & CLÁUDIO DE SOUZA BAPTISTA

CAMPINA GRANDE - PB
FEVEREIRO 2003

DIAS, Plácido Marinho

D531P

Projeto e Construção de um gerente de Data Warehouses Móveis

Dissertação de Mestrado, Universidade Federal de Campina Grande, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Fevereiro de 2003. 154p. Il.

Orientadores: Marcus Costa Sampaio
Cláudio de Souza Baptista

Palavras-chave:

1. Data Warehouse
2. Banco de Dados
3. Computação Móvel

~~CDU - 681.3.07B~~ CDU 004.65(043)

"PROJETO E CONSTRUÇÃO DE UM GERENTE DE DATA WAREHOUSES MÓVEIS"

PLÁCIDO MARINHO DIAS

DISSERTAÇÃO APROVADA EM 25.02.2003

PROF. MARCUS COSTA SAMPAIO, Dr.
Orientador

PROF. CLÁUDIO DE SOUZA BAPTISTA , Ph.D
Examinador

PROF. ULRICH SCHIEL, Dr.
Examinador

PROF. ÂNGELO RONCALLI ALENCAR BRAYNER, Dr.
Examinador

CAMPINA GRANDE – PB

"EU SOU DE UMA TERRA QUE O POVO PADECE
MAS NÃO ESMORECE E PROCURA VENCER.
DA TERRA QUERIDA, QUE A LINDA CABOCLA
DE RISO NA BOCA ZOMBA NO SOFRÊ
NÃO NEGO MEU SANGUE, NÃO NEGO MEU NOME.
OLHO PARA A FOME, PERGUNTO: QUE HÁ?
EU SOU BRASILEIRO, FILHO DO NORDESTE,
SOU CABRA DA PESTE, SOU DO CEARÁ."

(PATATIVA DO ASSARÉ)

Dedico este trabalho a:

Meus pais Francisco e Marli. Meus exemplos.

Edilda, minha metade.

Meus filhos Raíssa, Thaís e Matheus. Minha verdadeira obra!

Minha irmã, Dileuza.

Meu irmão, Italo. Um dia haveremos de nos encontrar!

Minha família, sempre confiante. Em especial a meus avós Júlio e Zenaide.

Ao meu sogro Edílson e a minha sogra Raimunda.

Aos meus cunhados e irmãos Railson, Kelsen, Rosa e Aida.

Agradecimentos

A Deus, por tudo.

Aos professores Cláudio Baptista e Marcus Sampaio pela amizade e pelas grandes orientações recebidas.

À Gladsa, minha inspiração.

Às Aninha e Vera, pelo carinho com que sempre me trataram.

Aos amigos fieis, Claudivan, Éder, Philip e Ricardo.

Aos amigos do mestrado.

Aos amigos do LSI.

À Embrapa e UVA, por me ajudar na obtenção deste objetivo.

A todos que de alguma forma contribuíram para realização deste trabalho,

OBRIGADO!

Resumo

O surgimento e a constante popularização de dispositivos portáteis, tais como notebooks, laptops e personal digital assistants (PDAs), e os avanços da tecnologia de rede de comunicação sem fio têm motivado pesquisas em uma nova classe de aplicações chamadas de móveis ou nômades, permitindo que usuários possam ter acesso a informações de qualquer lugar e a qualquer momento. Dentre essas novas aplicações destacamos Data Warehouse Móvel (DWM).

Construir sistemas de DWM significa transpor inúmeros desafios, como: disponibilidade em redes de baixa capacidade, funcionamento em plataformas móveis de pequeno porte, ao mesmo tempo garantindo o alto desempenho de consultas OLAP a DW. Como tecnologia nascente, há diversos problemas não satisfatoriamente resolvidos, ou em aberto.

Esta dissertação propõe um software de gerenciamento de DWMs, que tem como principais objetivos: a atualização simultânea de vários DWMs sem onerar a rede de comunicação de dados, a alocação de diversas tarefas de manutenção de DWMs fora das plataformas móveis, e o imperativo da eficiência dos algoritmos de atualização.

Abstract

The emergence and constantly growing popularity of portable devices such as notebooks, laptops and personal digital assistants (PDAs), and the advances in the technology of wireless communication networking, have motivated research in a new class of applications called mobile or nomadic, which permit users to have access to information from anywhere at any time. Among these new applications we focus on Mobile Data Warehouse (MDW).

To construct MDW systems means to transpose innumerable challenges, such as: availability in low capacity networks, function on small sized mobile platforms, whilst at the same time guaranteeing high performance in OLAP queries on DW. As an emerging technology there are various problems still not satisfactorily resolved, or unconcluded.

This dissertation proposes an MDW management software, that has as its main objectives: the simultaneous updating of various MDWs without burdening the data communication network, the allocation of diverse MDW maintenance tasks outside of the mobile platforms, and the imperative of the efficiency of updating algorithms.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

As transformações ocorridas nos últimos anos, fizeram com que os gerentes e executivos das corporações passassem a ver a informação como um recurso crítico, exigindo sistemas que explorassem todas suas vantagens competitivas [HH01].

As decisões tomadas pelos executivos de empresas, tradicionalmente, são baseadas em experiências adquiridas, ou mesmo, através de Sistemas de Informações Gerencias disponíveis nas organizações há vários anos, ou de outros sistemas tais como: EIS (Executive Information System) e DSS (Decision Support System) tendo com base os bancos de dados tradicionais.

Estes sistemas possuem uma capacidade limitada no fornecimento de informações relevantes para um processo de suporte à decisão, pois possuem uma fraca habilidade para analisar características dinâmicas da empresa e prever mudanças no mercado [WXL+01].

Uma das soluções para estas limitações foi a criação de um conjunto de tecnologias para gerenciamento e análise de dados tais como: Data Warehouse, Data Mart e Data Mining, dando ao processo de tomada de decisão um caráter mais dinâmico e técnico.

A necessidade constante de informações estratégicas, a popularização do uso de plataformas móveis (laptops, notebooks, etc) e o avanço dos meios de comunicação sem fio contribuíram para o surgimento de aplicações móveis destinadas ao suporte à decisão, dentre as quais destacamos Data Warehouse Móvel.

Data Warehouse Móvel é um caso particular de um Data Warehouse tradicional em que os dados para suporte a decisão encontram-se em equipamentos móveis.

Para detalharmos Data Warehouse Móvel, primeiramente faz-se necessária a definição de um Data Warehouse tradicional.

1.1 Data Warehouse

Data Warehousing é uma das áreas de maior crescimento em Sistemas de Informações Gerenciais. Nessa abordagem, os dados para EIS e DSS são separados dos dados operacionais e armazenados em um repositório denominado de Data Warehouse [Mfa96,Wid95a].

As informações obtidas a partir de um Data Warehouse são voltadas para o apoio à decisão, controle operacional e melhoria da eficiência nos processos administrativos, pois enquanto os bancos de dados transacionais são capazes de mostrar "o que" está na base de dados, o DW ajuda o usuário a descobrir o "porquê".

Data Warehouse (DW) é um banco de dados analítico utilizado para suporte à decisão [POE98]. As principais vantagens do uso de DW são melhorar a disponibilidade na obtenção dos dados gerenciais, melhorar a qualidade das informações obtidas e integrar dados de múltiplos bancos de dados heterogêneos distribuídos e de outras fontes de informação [Wid95b].

Os requisitos de um data warehouse são [KIM96]:

- O DW prover acesso a dados corporativos ou organizacionais;
- Os dados de uma data warehouse são consistentes com as fontes de dados;
- Os dados em um DW podem ser separados e combinados usando-se as diferentes medidas do negócio;
- Um DW também é constituído de um conjunto de ferramentas para consultas, análises e apresentação de informações e não apenas de dados;
- O DW é o local onde são publicados dados confiáveis;
- A qualidade dos dados em um DW deve impulsionar a reengenharia de negócios.

Segundo Chaudhuri & Dayal [CD97], Data Warehousing pode ser definido como uma coleção de tecnologias de apoio à decisão, com o propósito de possibilitar, aos que trabalham com conhecimento, ter decisões melhores e mais rápidas.

Para melhor descrever um Data Warehousing, utilizaremos uma arquitetura apresentada na figura 1.1 [CD97]. Essa arquitetura é composta de ferramentas para extrair dados de diferentes fontes internas e externas, limpar dados de anomalias presentes e carregar os dados dentro do DW [CD97, Fio98]. Estas ferramentas são denominadas de back-end.

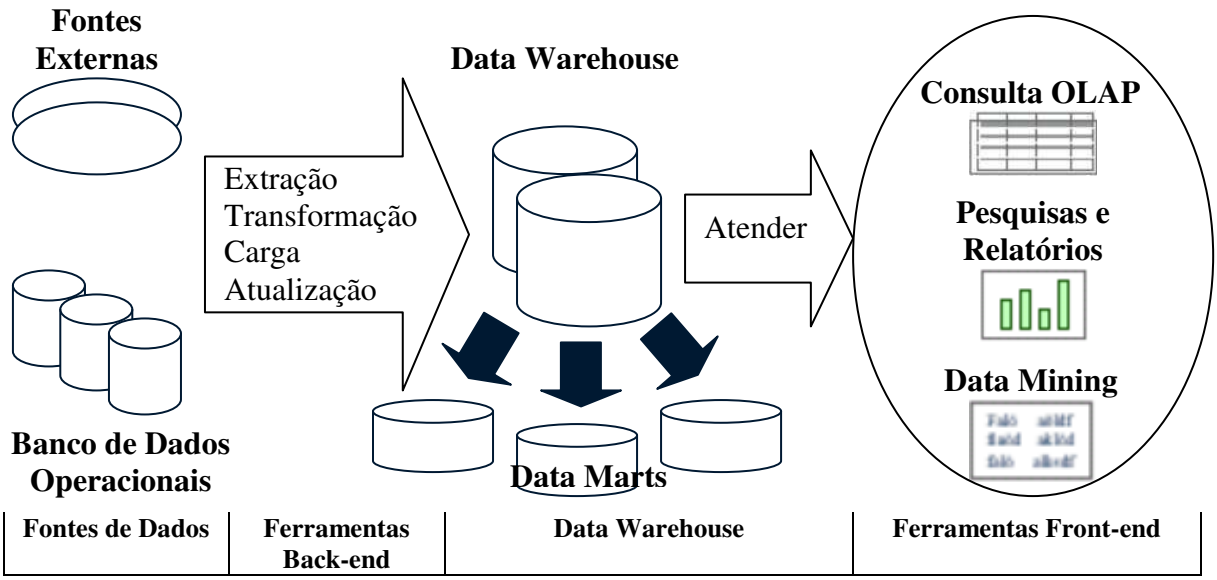


Figura 1.1: Arquitetura de um ambiente de um Data Warehouse [CD97]

As fontes de dados, normalmente, são de sistemas heterogêneos, tais como sistemas de arquivos, banco de dados relacionais e banco de dados orientado a objeto [Bre97], mas pode também ser de fontes externas tais como Internet ou banco de dados comerciais [CBS99].

Um outro componente observado na arquitetura é a presença de Data Mart (DM), que significa um DW orientado a assunto, representando um subconjunto de dados relevantes de um DW, relacionado a uma função particular de um negócio.

O termo Data Mart descreve a maneira como um departamento implementa seu próprio sistema de gerência de informação [Dev97] e é organizado em torno de um processo específico do negócio, constituindo um projeto que pode ser completado e executado mais rapidamente [KIM98].

Para fazer os dados disponíveis para os usuários do DW, a arquitetura também é composta de uma variedade de ferramentas de acesso e recuperação de dados [CD97, HHD99]. Chaudhuri & Dayal [CD97] refere-se a esse tipo de ferramenta como ferramentas front-end. Essas ferramentas são localizadas principalmente no lado do cliente do sistema, permitindo aos tomadores de decisão executar consultas ao DW [Bre97].

As principais características dos diversos tipos de ferramentas que podem ser utilizadas para extrair informações de um ambiente de data warehouse são [CAM98]:

- Pesquisas e relatórios, em geral, são ferramentas que oferecem uma interface gráfica para geração de SQL, permitindo o uso de menus e botões para a especificação de elementos de dados, condições, critérios de agrupamento, sem que seja necessário aprender uma linguagem especializada para acesso ao banco. O processamento estatístico, neste caso, é limitado a médias, totais, desvios padrão e

algumas outras funções básicas de análise, possuindo uma baixa capacidade técnica;

- Consultas OLAP (On-line Analytical Processing), acessam e analisam dados para suporte às decisões gerenciais, recuperando uma grande quantidade de dados, resumindo-os, a fim de detectar tendências e anomalias, podendo ser freqüentemente modificadas. Um dos princípios básicos destas consultas é o desempenho, ou seja, avaliar consultas complexas com um tempo de resposta adequado [Moe97, Sar97];
- Data Mining refere-se ao processo de extração não trivial de informação implícita, previamente desconhecida e potencialmente útil [PAK+02]. Tecnicamente, data mining é o processo de encontrar correlações ou padrões sobre dezenas de campos em banco de dados, arquivos externos e outros meios de armazenamento de dados [Dix97], gerando regras que guiam o processo de suporte à decisão.

1.1.1 Modelo Multidimensional

Um DW é geralmente representado por um modelo multidimensional. Neste modelo os dados são visualizados de uma forma intuitiva, como um *cubo de dados*, conforme a figura 1.2, onde cada lado representa uma dimensão e cada ponto interno ao cubo contém as medições do negócio para as diversas combinações das dimensões [KIM96].

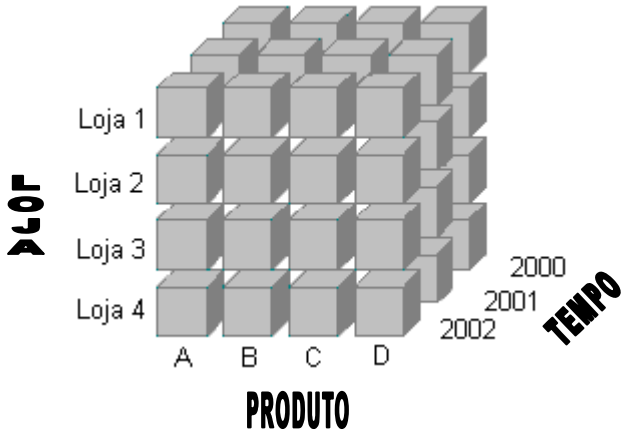


Figura 1.2: Cubo de dados

Uma das vantagens do modelo multidimensional é sua simplicidade, explicitando e facilitando a navegação nas medições do negócio, diferentemente do modelo entidade-relacionamento [KS99, Cou01], que divide os dados em várias entidades distintas sendo difícil a identificação de dados importantes para o negócio, não podendo ser usado como base para DW [KIM96].

A representação multidimensional, em um banco de dados relacional, é formada por dois tipos de tabelas [POE98, KIM98, Dev97, CAM98]:

- **Tabela de fatos**, onde são armazenadas as medições numéricas do negócio. Cada uma das medições é obtida na interseção de todas as dimensões [KIM96] e variam continuamente a cada amostragem. A tabela de fatos possui uma chave composta formada por todas as chaves das tabelas de dimensão existentes;
- **Tabelas de Dimensões**, também chamada de tabelas dimensionais, ou dimensão, contém as descrições textuais das dimensões do negócio e são usados como restrições no conjunto de repostas dos usuários e seus valores não são alterados com a mesma freqüência dos dados das tabelas de fatos. Uma das dimensões sempre necessária em DW é a dimensão tempo [KIM96].

O relacionamento entre as tabelas de fatos e as dimensões pode ser feito utilizando o esquema do tipo estrela [KIM96] ou outros esquemas, um dos quais o floco-de-neve [POE98]. O esquema estrela é o mais utilizado na modelagem multidimensional [JPB+02], onde a tabela de fatos é circundada por diversas tabelas de dimensões não normalizadas.

A figura 1.3 apresenta um exemplo de uma representação desse esquema, onde identificamos a tabela de fatos *Vendas* constituída de três chaves (Chave_Tempo, Chave_Produto e Chave_Loja) e duas medições do negócio (Quantidade e Valor); e três tabelas de dimensão (Tempo, Produto e Loja).

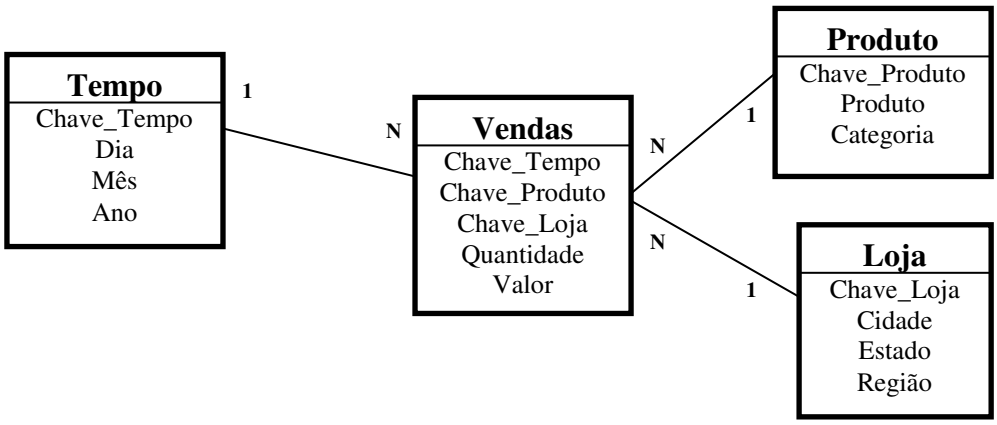


Figura 1.3: Esquema Estrela

1.1.2 Granularidade

As dimensões contêm uma ou mais hierarquias entre atributos, além de atributos que não representam nenhuma hierarquia.

Os atributos que constituem as hierarquias de uma dimensão possuem uma relação de 1 para n entre cada par contíguo destes, representando a granularidade dos itens de dados. Por exemplo, os atributos *cidade* e *estado*, da dimensão *Loja* da figura 1.3, representam uma hierarquia onde um estado possui várias cidades, portanto existe um relacionamento de 1 para n entre estes atributos.

A granularidade representa, portanto, o nível de detalhe dos dados, onde quanto maior a granularidade menor o nível de detalhe, e quanto menor a granularidade, maior o nível de detalhe [INM96]. A figura 1.4 apresenta exemplos de hierarquias das dimensões **Tempo** e **Loja**.

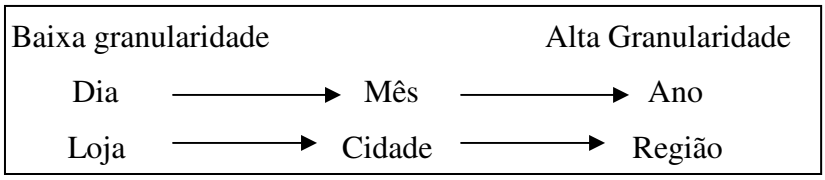


Figura 1.4: Hierarquia entre atributos

A dimensão *tempo* apresenta três atributos que formam sua hierarquia, onde *Dia* representa a menor granularidade, ou seja, um maior nível de detalhe, enquanto o *Ano* representa sua maior granularidade, ou seja, um menor nível de detalhe.

A dimensão *loja* é formada por três atributos hierárquicos, onde *Loja* possui sua menor granularidade e *Região* representa a granularidade mais alta.

1.1.3 Consultas OLAP: requisitos e operações

As consultas de dados, em sistemas OLAP, são feitas de forma interativa e os dados são apresentados numa visão multidimensional [CAM98], por isso as bases de dados são conhecidas com MDDBs (Multidimensional Database – Base de Dados Multidimensionais) [KIM98], constituindo a principal ferramenta front-end para consultas em ambiente de DW [DSB+99].

Codd e Date formularam uma lista de regras para avaliar a eficácia de uma ferramenta OLAP com relação às necessidades deste tipo de processamento [WEL+96]. A seguir, apresentamos as regras que podem caracterizar os requisitos de consultas OLAP:

1. Visão conceitual multidimensional: enfatiza a forma como o usuário "vê" dados sem impor que os dados sejam armazenados em formato multidimensional;
2. Transparência: localização da funcionalidade OLAP deve ser transparente para o usuário, assim como a localização e a forma dos dados;

3. Facilidade de Acesso: acesso a fontes de dados homogêneas e heterogêneas deve ser transparente;
4. Desempenho de consultas consistente: não deve ser dependente do número de dimensões;
5. Arquitetura cliente/servidor: produtos devem ser capazes de operar em arquiteturas cliente/servidor;
6. Dimensionalidade genérica: todas as dimensões são iguais;
7. Manipulação dinâmica de matrizes esparsas: produtos devem lidar com matrizes esparsas eficientemente;
8. Suporte multi-usuário;
9. Operações entre dimensões sem restrições;
10. Manipulação de dados intuitiva;
11. Relatórios/consultas flexíveis;
12. Níveis de agregação e dimensões ilimitados: ferramentas devem ser capazes de acomodar 15 a 20 dimensões.

As principais operações de navegação em um sistema OLAP são:

- Drill-down – Diminuir o nível de agregação, aumentando o nível de detalhe;
- Drill-up/Roll-up – Operação contrária ao drill-down, ou seja, aumenta o nível de agregação, diminuindo o nível de detalhe;
- Drill-across – é o processo de unir duas ou mais tabelas de fatos no mesmo nível de granularidade;
- Slicing and dicing – Seleção de linhas e projeção de colunas;
- Pivoting – Faz o pivoteamento dos dados (ex. rotação de linhas para colunas ou colunas para linhas)
- Outras operações.

1.1.4 Agregados

A combinação do volume de dados e complexidade de consultas ad-hoc têm motivado um grande número de pesquisas em Data Warehouse [DER98], a fim de atender aos exigentes requisitos de performance de gerentes e executivos de corporações, em relação ao tempo de resposta às consultas submetidas.

Otimização de consultas, técnicas de avaliação de consultas, estratégias de índices, particionamento, dentre outros, são usados para melhorar a performance em sistemas de suporte a decisão [CS94, GHQ95, OG95, TS97], porém o método mais eficiente para melhorar o desempenho das consultas em DW é o uso de resumos (agregados) pré-armazenados, que consiste no armazenamento dos resultados das consultas mais freqüentemente solicitadas [GBL+96, KAM93, KIM96, TS97].

Estes resumos pré-armazenados podem ser chamados de tabelas de agregados, agregados, dados pré-computados, tabelas sumarizadas, cubóide, visões materializadas, dentre outros.

Em um Data Warehouse podem existir três níveis de materialização[HAR96]:

- Nenhuma materialização, onde as visões são criadas no momento da consulta, provocando um aumento no tempo de respostas, porém economizando o espaço de armazenamento.
- Materialização completa, onde são criadas todas as visões possíveis, oferecendo um excelente tempo de resposta ao atendimento de consultas executadas. Essa abordagem provoca um aumento excessivo no espaço de armazenamento, além de aumentar significativamente o tempo para a atualização do DW;
- Materialização parcial. Neste enfoque somente algumas visões, geralmente as mais solicitadas, são materializadas, outras são computadas no momento da consulta. Este tipo de materialização tem como objetivo melhorar a performance das consultas sem exigir um grande espaço de armazenamento, diminuindo o tempo de atualização em relação à materialização completa.

Um registro de uma tabela de agregados representa o resumo de vários registros de uma tabela de fatos em seu nível básico [KIM96] ou de uma outra tabela de agregados [MQM97].

A figura 1.5 mostra uma tabela sumarizada, denominada *Vendas_Categoria*, que agrega todos os produtos da tabela de fatos *Vendas* em categorias. Nessa figura, observa-se, também, a criação de uma nova tabela de dimensão denominada *Categoria*. Essa nova tabela de dimensão possui um menor número de tuplas, em relação à tabela de dimensão *Produto* e é denominada de **dimensão encolhida** [KIM98].

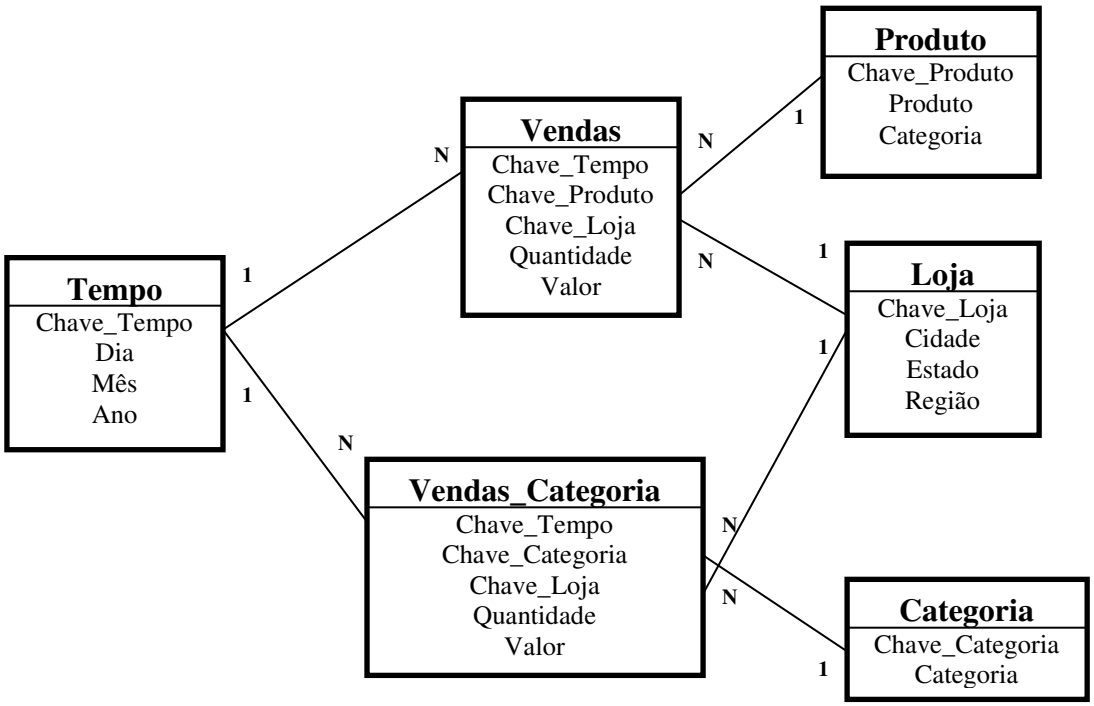


Figura 1.5: Esquema Estrela com uma tabela de agregados

1.1.5 Extração de dados das fontes

A extração de dados das fontes, para atualizar um DW, pode ser feita através de duas estratégias [BT98]: extração estática e extração incremental.

- **Extração estática**

Esta estratégia está vinculada a uma aquisição periódica de dados, não sendo fator determinante, que as técnicas pertencentes a esta categoria, detectem apenas porções de dados que foram modificadas durante o período compreendido entre duas aquisições consecutivas de dados.

Essa técnica de extração está associada à obtenção de um *snapshot* dos dados, seja de um conjunto completo de dados de interesse ou apenas um subconjunto deste, não existindo rotinas que fiquem monitorando os sistemas transacionais a procura dos registros inseridos, atualizados ou excluídos.

A extração estática é um processo automatizado, responsável por adquirir, periodicamente, as modificações ocorridas nos sistemas fontes ou o conjunto inteiro de dados de interesse do DW, não proporcionando a aquisição de dados históricos, a menos que os sistemas operacionais os mantenham.

- **Extração incremental**

Diferentemente do que ocorre com as técnicas de extração estática, a abordagem incremental equivale a uma replicação baseada em dados modificados para posterior distribuição ao DW e DM.

Nesse processo, todas os estados intermediários dos dados de produção são extraídos, visto que quaisquer mudanças, que ocorram nos dados, são capturadas através do mecanismo de *triggers* ou arquivos de *logs* disponíveis nos SGBDs, ou através das próprias aplicações responsáveis pela manutenção dos dados.

Esta abordagem representa uma vantagem sobre as técnicas de extração estática. Em contrapartida, o uso de métodos incrementais, tais como triggers e aplicações responsáveis pela extração, causam um *overhead* nos sistemas em função do monitoramento constante das fontes, excetuando-se o caso do emprego das funcionalidades do arquivo de log do SGBD.

Em ambas estratégias de extração, o DW deve estar ligado às fontes no momento da transferência de dados, porém a utilização da extração estática, com a obtenção de conjuntos completos de dados, provoca um aumento do volume de comunicação em relação à extração incremental.

1.1.6 Atualização do DW

A reconstrução completa e a manutenção incremental são as estratégias adotadas para atualizar um DW [MQM97, MSR+99, GM99].

- **Reconstrução**

Esse processo consiste na obtenção dos dados das fontes e posterior reconstrução completa de suas visões, aumentando o tempo de atualização e diminuindo o tempo de disponibilidade do DW. Esta atualização, na realidade, reconstrói totalmente as visões materializadas.

- **Manutenção incremental**

A manutenção incremental consiste na atualização de todas as visões do DW utilizando apenas as modificações extraídas nas fontes, sendo mais eficiente do que a reconstrução, principalmente se a visão tem um volume de dados muito superior ao volume das alterações ocorridas [BM90, MQM97, CKL+97,Qua97, GJM97, GM99].

A principal vantagem da manutenção incremental é diminuir o volume de comunicação e a carga de trabalho do processo de atualização, resultando em um aumento do tempo de disponibilidade do DW para os usuários.

1.2 Data Warehouse Móvel

Cada vez mais, os usuários, demandam uma constante disponibilidade de dados e informações, normalmente armazenadas em suas estações de trabalho, servidores de arquivos coorporativos e outras fontes externas, tais como WWW [HKZ02].

O surgimento e a constante popularização de dispositivos portáteis, tais como: notebooks, laptops e Personal Digital Assistant (PDA) permitiram que em qualquer lugar e a qualquer momento, os usuários possam acessar dados locais ou remotos. Além disso, os dispositivos móveis devem suportar diferentes tecnologias de rede existentes, como por exemplo, redes rápidas com ou sem fio, ou redes lentas de celular ou CDPD (Cellular Digital Packet Data) - tecnologia de transmissão de pacotes de dados pela rede de telefonia celular usando os espaços não usados pelos canais de voz [WB98].

Alguns destes dispositivos móveis têm poder suficiente para executar funções completas de servidores de banco de dados, outros somente podem executar algumas poucas funções destes [Gig01].

Os usuários que utilizam estes dispositivos são denominados de usuários móveis.

Além da proliferação dos dispositivos móveis, os avanços em rede de comunicação sem fio, têm motivado pesquisas em uma nova classe de aplicações conhecidas como móveis e nômades [Bar99], fazendo com que, provavelmente no futuro, o número de clientes móveis exceda o número de clientes fixos tradicionais [BP98].

A mobilidade e a transportabilidade, porém, impõem alguns desafios, tais como: pequena largura de banda, freqüentes desconexões, desconexões previsíveis, custo de conexão, limitada capacidade da bateria, recursos limitados, pequeno tamanho do monitor, suscetibilidade a destruição de dados por roubo e acidente, rápida mudança de localização, escalabilidade e segurança [PB94]. Estes desafios levam-nos a um requisito fundamental: os usuários não estão conectados constantemente à rede.

O uso da computação móvel implicitamente envolve duas dimensões: espaço e tempo. Uma aplicação móvel é capaz de responder consultas com relação a *onde*, *quando*, *o quê* e *como*.

Seguindo a tendência de informações serem obtidas em qualquer lugar e a qualquer momento, muitas aplicações estão migrando para estes ambientes móveis, a fim de que possam ser usadas sem interrupção, caso a rede esteja disponível ou não. Existem sistemas de banco de dados, sistemas operacionais, editores de textos e aplicações web, todas elas aplicadas aos dispositivos móveis, principalmente para atender a demanda de tomadores de

decisão. Entre estas aplicações destacamos Data Warehousing.

Data Warehouse Móvel ou DWM é um DW que reside em uma plataforma móvel, permitindo que usuários efetuem consultas OLAP mesmo durante longos períodos de desconexão das fontes de dados.

1.2.1 Exemplo de Data Warehouses Móveis

Considere uma empresa de comércio varejista com filiais localizada em várias regiões do país. Seus executivos apresentam diferentes padrões de acessos a dados em relação aos usuários tradicionais, em função da necessidade constante de informações estratégicas independente de local e de momento específicos.

Estes executivos são munidos de equipamentos móveis com DWMs para suporte à decisão. Os DWMs podem ser atualizados, com dados das filiais, mesmo quando se encontram distantes da matriz, pois os equipamentos permitem a comunicação sem fio.

Os diversos equipamentos móveis permanecem desconectados das filiais por diferentes períodos de tempo, fazendo com que os DWMs estejam em variados níveis de atualização. A freqüência de conexão dos DWMs com as filiais depende dos seguintes fatores:

- Diferentes necessidades de atualização: as diferentes características das decisões estratégicas exigem uma menor ou maior freqüência de atualização dos DWMs, pois algumas decisões podem ser tomadas com dados desatualizados, enquanto outras requerem dados constantemente consistentes com as fontes;
- Ausência/deficiência dos canais de comunicação: os executivos podem se encontrar em locais onde não exista cobertura para a transmissão sem fio, ou mesmo em locais onde a taxa de transferência de dados seja reduzida, inviabilizando a obtenção de dados das filiais e a conseqüente atualização dos DWMs;
- Custo de conexão elevado: os executivos podem está em um local onde o custo de conexão seja muito elevado, inviabilizando a comunicação com as filiais e por conseqüência a atualização dos DWMs.

Os DWMs necessitam ser atualizados após os dados serem obtidos das fontes (filiais). Durante a atualização, os executivos ficam impossibilitados de efetuar consultas nos DWMs, denotando que, quanto maior o tempo necessário para a atualização maior será o período de indisponibilidade para consultas.

O desempenho da atualização e o tempo de disponibilidade do DWM dependem dos seguintes fatores:

- Quantidade de dados necessários na atualização: o volume de dados necessários para atualizar os DWMs irá variar de acordo com o período de desconexão com as fontes de dados, ou seja, quanto maior o tempo de desconexão maior será o volume de dados transmitidos para os DWMs, aumentando o tempo de atualização e diminuindo a disponibilidade dos DWMs;
- Recursos da plataforma móvel: a atualização dos DWMs é processada na plataforma móvel, ou seja, quanto mais recursos computacionais a plataforma móvel possuir mais eficiente será o processo de atualização e maior será a disponibilidade dos DWMs para consultas;

1.2.2 Modelo de acesso a dados

Uma forma de caracterizar acesso a dados de um Data Warehouse Móvel (DWM) é categorizar, inicialmente, as aplicações de acordo com seus modelos de conectividade de dados. Baseado em [Gig01], identificamos dois modelos de acesso a dados:

- Dependente de conexão contínua; neste modelo a aplicação que está sendo executada necessita de uma conexão constante a um servidor de dados central, sendo impossível sua execução se não existe conexão. Este modelo é freqüentemente usado em computadores desktop que acessam dados centralizados através de redes locais;
- Independentes de conexão contínua; neste modelo, a aplicação pode ser executada sem haver uma conexão constante a um servidor central, pois acessam dados armazenados localmente. Este modelo, ainda permite dois modos de conexão, em relação à disponibilidade de acesso ao servidor:
    - Conexão raramente possível. O servidor de banco de dados central somente está disponível ocasionalmente, pois o dispositivo está na maior parte do tempo fora da rede local. As aplicações acessam dados armazenados localmente e são executados em dispositivos que estão

longe da rede local da corporação ou mesmo em dispositivos que não são capazes de comunicação sem fio;

- Conexão sempre disponível. As aplicações são executadas em dispositivos que permitem a comunicação sem fio e, teoricamente, são capazes de comunicar-se com um servidor de dados centralizado a qualquer momento. Existem momentos que a comunicação não é possível, quando há problemas de cobertura na comunicação dos canais sem fio, ou indesejável, se o custo da comunicação é muito alto, ou a taxa de transmissão é muito baixa. Nesse modo de conexão, as aplicações, também, acessam dados armazenados localmente.

Entre os modelos de acessos a dados apresentados, o modelo independente de conexão contínua, é o mais adequado para DWM, em função das constantes desconexões a que os dispositivos móveis são submetidos, permitindo que os usuários processem suas consultas em visões materializadas armazenadas localmente.

A desconexão é o fato dos usuários operarem sem conectividade, por um período indeterminado de tempo, usando cópias de dados criadas e atualizadas, localmente, durante uma conexão, sendo necessárias transmissões periódicas de dados alterados nas fontes para o DWM.

1.2.3 Tipos de transmissão de dados em ambientes móveis

Em ambientes de computação móvel, existem três técnicas de transmissão de dados: pure-push-based, pure-pull-based e a combinação destas duas técnicas [LYL+02].

No mecanismo pure-pull-based, também chamado *por demanda (on-demand)* [FR98], o cliente móvel explicitamente envia solicitações de seu interesse para o servidor, requerendo periodicamente as mudanças ocorridas neste.

Nesse tipo de transmissão, o servidor não necessita saber informações sobre seus clientes [LSV98], respondendo a cada solicitação individualmente [LYL+02]. Esse mecanismo de transmissão é indicado quando os dados possuem uma pequena volatilidade [TV00], tendo a escalabilidade como grande desvantagem [ST97], pois a carga de trabalho do servidor é proporcional ao número de clientes móveis que solicitam dados.

Na técnica pure-push-based, também chamado de transmissão periódica (periodic broadcast) [FR98], o servidor é responsável pela notificação de todos os clientes, sendo

indicado para disseminar informações para um grande número de clientes [PC99] ou para disseminar dados que apresentem uma alta volatilidade [TV00].

A maior vantagem desta técnica é que os clientes móveis obtêm as informações diretamente nos canais de transmissão, não sobrecarregando o servidor, apresentando uma maior escalabilidade [ST97] e sua maior desvantagem é que um cliente móvel pode está desconectado no momento da notificação, dificultando a atualização de dados nos dispositivos móveis [CFG00].

A terceira técnica combina as duas técnicas anteriores, usando o pure-push-based para dados de interesses comuns entre todos os clientes; e pure-pull-based quando os dados são destinados a clientes específicos [LYL+02, LLS02, OHP00].

Em DWM, a técnica mais adequada para transferência de informação é a pure-pull-based, em função do pequeno número de clientes existentes e do pequeno volume de dados alterados.

1.3 Desafios de um gerente de DWMs

Embora diversas pesquisas realizadas para ambientes fixos, como banco de dados distribuídos, possam ser usadas em banco de dados móveis, as desconexões, naqueles sistemas, são consideradas falhas, entretanto, em banco de dados móveis a desconexão é um modo de operação própria deste sistema de banco de dados [SAE+99], havendo necessidade de desenvolver pesquisas específicas para banco de dados móveis, notadamente para DWM.

Um Data Warehouse, de um modo geral, necessita, periodicamente, extrair dados das fontes, a fim de deixá-lo consistente com estas.

Em um DW tradicional, as extrações de dados podem utilizar as técnicas de extração estática ou incremental, obtendo conjuntos completos de dados ou apenas as alterações ocorridas nas fontes depois da última atualização.

Em DWM, entretanto, a extração estática, com obtenção completa de dados, representa uma estratégia proibitiva em função das restrições impostas pelos canais de comunicação, sendo indicado apenas a extração incremental pelo menor volume de dados obtidos das fontes.

O volume de dados quando o período de desconexão é longo, geralmente, é superior ao volume de dados quando o período de desconexão é curto.

A variação do volume de dados que podem ser transferidas das fontes para o DWM pode acarretar em um aumento no tempo e no custo de transferência de dados, bem como, no tempo de atualização do DWM, inviabilizando sua manutenção.

A existência de vários DWMs somado aos diferentes períodos de desconexão com as fontes de dados implicarão em DWMs com variados estados de atualização.

As atualizações das visões materializadas realizadas nos dispositivos móveis, também constituem um problema para esta abordagem, haja vista que os dispositivos móveis, geralmente, não possuem a mesma capacidade de processamento de um computador fixo, aumentando o tempo de indisponibilidade do DW para as consultas.

A atualização de um DWM, também impõe um outro desafio quando as visões materializadas dependem diretamente das fontes, havendo a necessidade do DWM consultá-las para calcular os novos valores para atualizar as referidas visões.

Em resumo, os desafios mais importantes para a gerência de DWMs são a diminuição do volume de comunicação com as fontes de dados e do tempo de atualização das visões materializadas nos dispositivos móveis; e a manutenção de vários DWMs em diferentes estados de atualização.

1.4 Objetivos da dissertação

O objetivo principal deste trabalho é projetar e desenvolver um software, denominado MDWManager — *Mobile Data Warehouse Manager*, responsável pela gerência de diversos data warehouses instalados em plataformas móveis — DWMs, levando em consideração as restrições impostas pela mobilidade das plataformas.

Entre os objetivos específicos traçados para o MDWManager, destacamos:

- Construção de um servidor de dados fixo − proxy −, que deve ser responsável pela preparação dos dados extraídos incrementalmente das fontes de dados[1] dos DWMs;
- O proxy deve oferecer todos os serviços de manutenção que podem ser feitos externamente aos DWMs, visando reduzir o volume da comunicação entre o proxy e os DWMs, e o custo do processamento, propriamente dito, das atualizações dos DWMs;

[1] A integração das fontes de dados, que consiste na transformação dos dados das fontes no formato do Data Warehouse, não está no escopo da dissertação.

- O proxy deve admitir a autonomia dos DWMs, isto é, ser capaz de identificar os dados necessários para a atualização de diversos DWMs, de acordo com seus níveis particulares de desatualização;
- Os DWMs devem se comunicar exclusivamente com o proxy;
- O MDWManager (proxy + DWMs) deve oferecer algoritmos eficientes de atualização incremental dos DWMs;
- O algoritmo de atualização deve prever a existência de funções de agregação diferentes das clássicas soma, média, máximo, mínimo e contagem. As novas funções de agregação são variância, desvio-padrão, análise de regressão, dentre outras;
- Os serviços oferecidos pelo MDWManager deverão dar aos DWMs a flexibilidade necessária para que seus usuários possam efetuar consultas OLAP durante os períodos de desconexão do proxy;
- As atualizações dos DWMs deverão ser sob demanda dos mesmos;
- O MDWManager deverá passar por avaliações experimentais.

1.5 Relevância

O aumento da demanda em aplicações móveis, somado à necessidade constante de utilização de Sistemas de Suporte a Decisão (DSS), fez surgir o Data Warehouse Móvel, como um novo ambiente de análise de dados gerenciais fora de um ambiente fixo tradicional. Entretanto, poucos trabalhos em DWs foram desenvolvidos com o objetivo de atender as limitações impostas pela mobilidade. A maioria dos trabalhos falha nos seguintes aspectos:

- Não define uma arquitetura para ambientes móveis;
- Não aborda a gerência de vários DWs;
- Não diminui a carga de trabalho do DW, pois todo o processamento de atualização é executado na plataforma onde se encontra o DW, através de algoritmos ineficientes e fracamente acoplados ao SGBD, não utilizando os mecanismos de otimização disponíveis;
- Os DWs necessitam acessar constantemente as fontes de dados para atualizar as visões materializadas dependentes diretamente delas, aumentando o volume

de comunicação na rede. Por exemplo, na atualização de visões materializadas com funções de mínimo e máximo.

O gerente de DWMs, apresentado neste trabalho, supre as limitações das diversas abordagens, oferecendo as seguintes contribuições:

- O gerente de DWMs utiliza e adapta aspectos positivos de diversos trabalhos sobre DW, a fim de atender as limitações impostas pela mobilidade;

- Um servidor proxy fixo é responsável, não somente, pela extração de dados das fontes, mas pela manutenção de tabelas internas necessárias à atualização de visões materializadas nos dispositivos móveis. Essas tabelas internas visam eliminar o acesso direto entre os DWMs e as fontes de dados; restringir o acesso entre o servidor proxy e as fontes de dados; diminuir o tempo de acesso e o volume de dados transferidos entre as plataformas móveis e o servidor proxy; e aumentar o desempenho da atualização dos DWMs;

- O gerente de DWMs oferece algoritmos incrementais fortemente acoplados a SGBDs, tornando a preparação dos dados no servidor proxy e atualização dos DWMs mais eficientes em relação a algoritmos fracamente acoplados, atualizando visões materializadas com as funções de agregação clássicas SUM, COUNT, AVG, MIN e MAX, bem como, novas funções de agregação;

- A comunicação entre os DWMs e o servidor proxy é feita através de uma estratégia de comunicação que obtém dados das tabelas do servidor proxy de forma eficiente, reduzindo o volume de dados na rede e a carga de trabalho do servidor proxy;

- A manutenção das tabelas do proxy é feita através de uma hierarquia (grafo), baseada no menor custo de propagação, ou seja, a atualização de uma tabela corrente utiliza a menor tabela ancestral;

- Os caches localizados no servidor proxy isolam completamente os DWMs do contato direto com as fontes de dados.

1.6 Estrutura da Dissertação

Este trabalho, além desta introdução, está organizado em mais quatro capítulos, descritos, resumidamente, a seguir.

O capítulo 2 - ***Trabalhos Relacionados***, apresenta trabalhos relevantes sobre atualização incremental de data warehouse e arquiteturas de data warehouse móveis, onde são detalhados os aspectos positivos e negativos de cada abordagem.

O capítulo 3 - ***MDWManager - Um gerente de Data Warehouses Móveis***, descreve, detalhadamente, o gerente de DWMs com relação ao modelo de um DWM, requisitos funcionais e não funcionais, arquitetura, algoritmos de preparação e atualização, estratégia de comunicação entre os DWMs e o servidor proxy, além de tecer considerações sobre os requisitos levantados.

O capítulo 4 - ***Avaliação Experimental***, apresenta diversas investigações, onde inicialmente é avaliada a performance de diversos algoritmos de preparação de dados no servidor proxy. Posteriormente, um estudo detalhado é apresentado sobre atualização, na plataforma móvel, de visões materializadas usando diferentes algoritmos e, por fim, é avaliada a carga de trabalho do servidor proxy mediante diferentes níveis de solicitação de atualização de DWMs.

Finalmente, o capítulo 5 - ***Conclusão***, apresenta as conclusões e futuras atividades de pesquisas que poderão ser desenvolvidas.

Capítulo 2

Trabalhos Relacionados

Data Warehouse Móvel não tem sido satisfatoriamente abordado na literatura apesar de sua relevância, porém algumas propostas, apresentadas neste capítulo, sobre atualização incremental de DWs tradicionais e arquiteturas podem ser utilizadas ou adaptadas para o desenvolvimento de um gerente de DWMs.

Na proposta de Mumick *et al* [MQM97] são abordados estudos sobre manutenção incremental de um DW tradicional com a definição de um novo paradigma denominado *tabela delta-sumário* e a divisão da atualização em dois processos distintos, aumentando o tempo de disponibilidade do DW.

Chan *et al* [CLS00] propõe uma arquitetura para a gerência de vários DWs instalados em servidores web e apresenta modificações na forma de atualização das visões materializadas, aumentando o tempo de disponibilidade do DW em relação à [MQM97].

Labio *et al* [LYC+99] apresenta novos algoritmos de atualização incremental de DWs tradicionais, bem como, estudos detalhados sobre a performance dos mesmos.

Na proposta de Stanoi *et al* [SAE+99] três diferentes arquiteturas são oferecidas para data warehouse móveis.

Na replicação do Oracle 9i [Ora02] é permitida a atualização incremental de visões materializadas instaladas, inclusive, em plataformas móveis.

Esse capítulo apresenta detalhadamente as propostas supracitadas, ressaltando os aspectos positivos e negativos para adoção em ambientes móveis.

2.1 Proposta de Mumick *et al* [MQM97]

O trabalho mostra que usando eficientes técnicas de manutenção incremental é possível aumentar o número de tabelas sumarizadas disponíveis no DW ou, alternativamente, diminuir o seu tempo de indisponibilidade para consultas dos usuários. Em resumo, o trabalho inclui as seguintes contribuições:

- Um novo método, chamado de tabelas delta-sumário, para manter visões agregadas. As tabelas delta-sumário representam um novo paradigma para a manutenção incremental;
- Uma estratégia para minimizar o tempo de atualização necessário para a manutenção de um data warehouse, através da divisão do trabalho de atualização em duas funções: *Propagate* e *Refresh*. A *função Propagate* não indisponibiliza o DW, enquanto somente a *função Refresh* deixa o DW indisponível;
- O trabalho mostra como múltiplas tabelas sumarizadas podem manter outras tabelas sumarizadas, obtendo vantagens significativas na otimização do processo de manutenção incremental.

2.1.1 Noções importantes

Este trabalho faz um estudo detalhado sobre funções *self-maintainable,* como funções que podem ser utilizadas em um processo de atualização incremental, bem como, suas restrições, além de fazer considerações importantes sobre funções de agregação de mínimo e máximo e a utilização de hierarquias de visões com um objetivo de acelerar a atualização do DW.

- **Funções Self-maintainable**

Antes de conceituarmos funções self-maintainable é importante tecer considerações sobre os tipos de funções de agregação. As funções de agregação podem ser divididas em três classes [GBL+96]: distributivas, algébricas e holísticas.

Funções distributivas são funções que podem ser calculadas em conjuntos disjuntos, agregando cada resultado individual no resultado final. As funções COUNT, SUM, MIN e

MAX são distributivas. Por exemplo, COUNT pode ser calculada somando as contagens parciais.

Funções algébricas podem ser expressas com uma ou mais funções distributivas. Média (AVG) é uma função algébrica que pode ser expressa com SUM/COUNT. Isto denota que se uma visão possuir uma função de agregação AVG, as funções SUM e COUNT devem substituí-la.

Funções holísticas não podem ser calculadas dividindo em partes. Mediana é um exemplo de função de agregação holística.

Uma função é *self-maintainable*, se um novo valor pode ser calculado a partir do valor antigo e das alterações ocorridas, devendo ser distributivas.

Uma função pode ser *self-maintainable* com relação à inclusão, porém, não ser *self-maintainable* em relação à exclusão. Isso significa que algumas funções de agregação *self-maintainable*, presentes em visões, suportam inserções porém não suportam exclusões, havendo a necessidade de alguns modificações.

A função COUNT pode ajudar a fazer algumas funções *self-maintainable* com relação à exclusão. Por exemplo, a inclusão da função COUNT(*) é necessário para fazer a função SUM *self-maintainable* com respeito à exclusão. Por exemplo, quando o COUNT(*) chegar a um valor zero (0), para um agrupamento, a função SUM(expr) deve ser nula e não zero.

- **Funções Mínimos e Máximos**

As funções mínimo e máximo não são e não podem se tornar *self-maintainable* com relação à exclusão. Por exemplo, quando uma tupla com um valor mínimo ou máximo é excluída, um novo valor mínimo ou máximo deve ser recalculado usando a tabela base, para um dado agrupamento. A presença da função COUNT pode ajudar, pois quando o COUNT(*) chegar ao valor zero (0) nenhuma tupla deverá existir para aquele agrupamento, porém se o valor de COUNT(*) for maior do que 0, após a exclusão de valores mínimos ou máximos, um novo valor deve ser recalculado usando a tabela base.

- **Cubo de dados**

O trabalho introduz a idéia de cubo de dados como uma forma de pensar em múltiplas visões, todas derivadas da tabela de fatos, usando diferentes subconjuntos de atributos de

agrupamentos (group-by). Um cubo de dados com k dimensões pode gerar 2^k visões de cubos, representando 2^k subconjuntos de atributos group-by [MQM97].

A figura 2.1 mostra um cubo de dados com as dimensões loja, produto e dia.

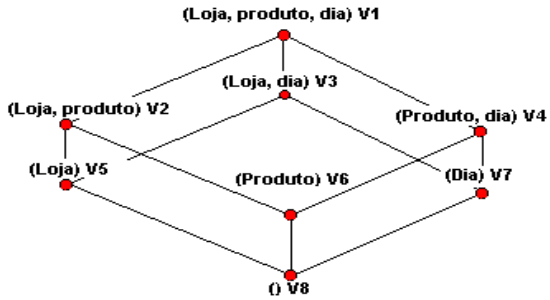


Figura 2.1: Lattice de um cubo de dados [MQM97]

As arestas do lattice dirigem-se do nó superior para o nó inferior. $V_2 \rightarrow V_5$, da figura 2.1, implica que V_5 pode ser respondida usando V_2, ao invés de acessar a tabela de fatos, denotando uma relação de dependência. Por exemplo, a aresta V_2 = (loja, produto) $\rightarrow$ V_5 = (loja) indica que para criarmos uma consulta agregando *loja*, não é necessário usar a tabela de fatos, mas a tabela V_2.

2.1.2 Exemplo de um Data Warehouse

Consideramos um data warehouse, onde a tabela de fatos, denominada PV, contém dados de todas as vendas registradas nos bancos de dados de diversas lojas. A tabela PV contém a seguinte estrutura:

PV (loja, produto, dia, data, qtd, preço).

As tabelas de dimensões loja e produto terão os seguintes formatos:

loja(loja, cidade, região) e

produto (produto, nome, categoria, custo)

O DW também é formado por quatro tabelas sumarizadas criadas a partir da tabela de fatos PV. As tabelas sumarizadas são criadas como a seguir:

- ✓ CREATE VIEW LPD_vendas(loja, produto, dia, TotalCount, Menor_qtd, TotalQuantitade) as
  SELECT loja, produto, dia, COUNT(*) AS TotalCount,
  MIN(qtd) AS Menor_qtd, SUM(qtd) AS TotalQuantidade
  FROM pv
  GROUP BY loja, produto, dia

- ✓ CREATE VIEW CD_vendas (cidade, dia, TotalCount, TotalQuantidade) as
  SELECT cidade, dia, COUNT(*) AS TotalCount,
  SUM(qtd) as TotalQuantidade
  FROM pv, loja
  WHERE pv.loja = loja.loja
  GROUP BY cidade, dia

- ✓ CREATE VIEW LC_vendas (loja, categoria, TotalCount, Menor_qtd,
  TotalQuantidade) as
  SELECT loja, categoria, COUNT(*) AS TotalCount, MIN(qtd) as
  Menor_qtd, SUM(qtd) AS TotalQuantidade
  FROM pv, produto
  WHERE pv.produto = produto.produto
  GROUP BY loja, categoria

- ✓ CREATE VIEW R_vendas (região, TotalCount, TotalQuantidade) as
  SELECT região, COUNT(*) AS TotalCount,
  SUM(qtd) AS TotalQuantidade
  FROM pv, loja
  WHERE pv.loja = loja.loja
  GROUP BY região

2.1.3 Criação otimizada de tabelas sumarizadas

Como mostrado anteriormente é possível a criação de uma visão V_5 a partir de V_2, sem a necessidade de usar a tabela de fatos. Entretanto para criarmos uma tabela V_5, pode ser necessária a junção da tabela V_2 a uma ou mais tabelas de dimensão.

A figura 2.2 apresenta a criação otimizada de tabelas sumarizadas a partir de uma outra tabela sumarizada, usando nosso exemplo de DW, onde as arestas rotuladas representam as dimensões necessárias para criar as visões descendentes.

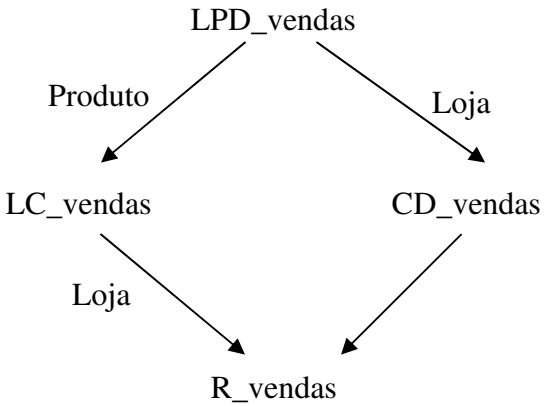


Figura 2.2: Lattice otimizado de tabelas sumarizadas [MQM97]

Abaixo apresentamos uma instrução que cria uma tabela sumarizada CD_vendas utilizando a tabela sumarizada LPD_vendas, sem a utilização da tabela de fatos PV:

- ✓ CREATE VIEW CD_vendas (cidade, dia, TotalCount, TotalQuantidade) as
  SELECT cidade, dia, SUM(TotalCount) AS TotalCount,
  SUM(TotalQuantidade) AS TotalQuantidade
  FROM LPD_vendas, loja
  WHERE LPD_vendas.loja = loja.loja
  GROUP BY cidade, dia

2.1.4 Algoritmo de manutenção de tabelas sumarizadas

Um importante aspecto desse trabalho é que o processo de atualização de um DW é dividido em duas funções: *propagate* e *refresh*.

A função propagate envolve a criação de tabelas delta-sumário representando um resumo das mudanças ocorridas na tabela base, que deverão ser aplicadas nas tabelas sumarizadas. Essa função não indisponibiliza o DW.

A função refresh é responsável pela atualização das tabelas sumarizadas, onde serão usadas as tabelas delta-sumário criadas pela função propagate. Durante a atualização, o DW fica indisponível para consulta.

- ➢ **Função Propagate**

A função propagate é dividida em duas fases: a preparação das mudanças e a criação das tabelas delta-sumário.

Preparando mudanças

Inicialmente é necessária a criação de três visões: prepara_mudança, prepara_inserção e prepara_exclusão. A visão prepara_mudança representa a união da visão prepara_inserção e prepara_exclusão.

As visões prepara_inserção e prepara_exclusão são projeções das inserções e exclusões na tabela de fatos.

A Tabela 2.1 apresenta as conversões necessárias para a criação das visões prepara_inserção e prepara_exclusão, a partir das funções existentes nas tabelas sumarizadas.

	Prepara_inserção	prepara_exclusão
COUNT(*)	1	-1
COUNT(expr)	caso expr é nulo então 0 senão 1	caso expr é nulo então 0 senão -1
SUM(expr)	expr	-expr
MIN(expr)	expr	expr
MAX(expr)	expr	expr

Tabela 2.1: Tabela de conversão de funções de agregação [MQM97]

A tabela mostra que se alguma tabela sumarizada do DW possuir uma função de agregação COUNT(*), as visões prepara_inserção e prepara_exclusão deverão possuir o valor 1 e -1, respectivamente; e se possuírem uma função de agregação SUM(expr), as visões prepara_inserção e prepara_exclusão deverão ter *expr* e *-expr* respectivamente.

Considerando o exemplo da tabela PV, as inserções são armazenadas na tabela PV_ins e as exclusões na tabela PV_del.

Posteriormente são criadas as visões prepara_inserção (prefixo pi_), prepara_exclusão (prefixo pe_) e prepara_mudanças (prefixo pm_). Abaixo apresentamos a criação dessas visões para manter a tabela LPD_vendas:

- ✓ CREATE VIEW pi_LPD_vendas(loja, produto, dia, TotalCount, Menor_qtd TotalQuantidade) as
  SELECT loja, produto, dia, 1 as TotalCount, qtd as Menor_qtd, qtd as TotalQuantidade
  FROM pv_ins WHERE pv_ins.produto = produtos.produto

- ✓ CREATE VIEW pe_LPD_vendas (loja,produto,dia,TotalCount, Menor_qtd
  TotalQuantidade) as
  SELECT loja, produto, dia, -1 as TotalCount, qtd as Menor_qtd,
  -qtd as TotalQuantidade
  FROM pv_del WHERE pv_del.produto = produtos.produto

- ✓ CREATE VIEW pm_LPD_vendas(loja,produto,dia,TotalCount, Menor_qtd
  TotalQuantidade) as
  SELECT * FROM (pi_LPD_vendas UNION ALL pe_LPD_vendas)

Criando tabelas delta-sumário

As tabelas delta-sumários são criadas agregando as visões prepara_mudanças com os mesmos critérios de agregações das tabelas sumarizadas, possuindo os mesmos esquemas destas.

A consulta para criar a tabela delta-sumário é semelhante à consulta para a criação da tabela sumarizada, com as seguintes diferenças:

- A cláusula FROM é substituída pela prepara_mudanças (pm_);
- A cláusula WHERE é removida;
- A função COUNT da tabela sumarizada é substituída pela função SUM.

A tabela delta-sumário sd_LPD_vendas é criada a partir da tabela pm_LPD_vendas, usando a seguinte instrução SQL:

- ✓ CREATE VIEW sd_LPD_vendas(loja,produto,dia,TotalCount, Menor_qtd
  TotalQuantidade) as
  SELECT loja, produto,dia, SUM(TotalCount) as TotalCount,
  MIN(Menor_qtd) as Menor_qtd,
  SUM(TotalQuantidade) as TotalQuantidade
  FROM pm_LPD_vendas
  GROUP BY loja, produto,dia

- **Função Refresh**

A função Refresh aplica as mudanças, representadas nas tabelas delta-sumário às tabelas sumarizadas através de cursor. Cada tupla na tabela delta-sumário causa mudança em uma simples tupla correspondente na tabela sumarizada. A tupla correspondente pode ser alterada, se uma tupla correspondente for encontrada na tabela sumarizada; ou inserida, se uma tupla correspondente não for encontrada.

Uma tupla de uma tabela sumarizada pode ser excluída, se COUNT(*) de uma tupla t de uma tabela sumarizada adicionada ao COUNT(*) de uma tupla t' de uma tabela delta-sumário for igual a zero, porém, se a adição resultar em um valor diferente de zero e a tabela sumarizada possuir funções de agregação de mínimo ou máximo, o *Refresh* verificará se um valor igual ao mínimo; ou igual ao máximo foi excluído, neste caso há necessidade de recalcular um novo valor para essas funções, utilizando a tabela base.

2.1.5 Criação otimizada de tabelas delta-sumários

Semelhante ao lattice otimizado de tabelas sumarizadas, uma estrutura de lattice pode ser usada para gerar outras tabelas delta-sumários.

A figura 2.3 apresenta a criação otimizada de tabelas delta-sumários, usando nosso exemplo de DW, onde as arestas rotuladas representam as dimensões necessárias para criar tabelas delta-sumários descendentes.

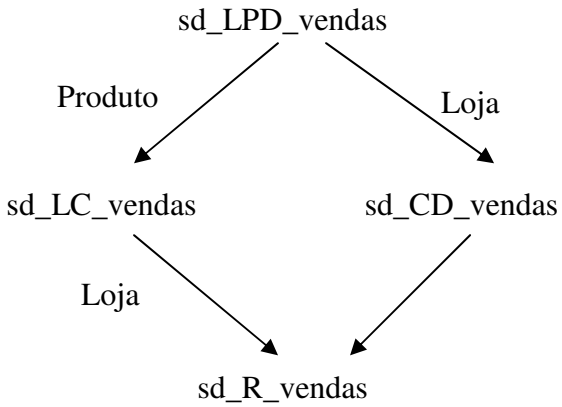


Figura 2.3: Lattice otimizado de tabelas delta-sumários

Abaixo apresentamos uma instrução que cria uma tabela delta-sumário sd_CD_vendas utilizando a tabela sd_LPD_vendas:

- ✓ CREATE VIEW sd_CD_vendas(cidade,dia,TotalCount,TotalQuantidade)as
  SELECT cidade, dia, SUM(TotalCount) AS TotalCount,
  SUM(TotalQuantidade) AS TotalQuantidade
  FROM sd_LPD_vendas, loja
  WHERE sd_LPD_vendas.loja = loja.loja
  GROUP BY cidade, dia

2.1.6 Estudo de performance

O trabalho mostra que a manutenção incremental, usando tabelas delta-sumário, melhora a performance da atualização de tabelas sumarizadas em relação à reconstrução, além disso, o estudo mostra os efeitos benéficos da divisão da atualização em duas funções (propagate e refresh).

2.1.7 Críticas

O trabalho tem como principais pontos fortes:

- Criação de um novo paradigma, denominado de tabela delta-sumário, resumindo o conjunto das alterações ocorridas na tabela base, pois agrupa as tuplas com o mesmo critério de agregação da tabela sumarizada que ela manterá. As tabelas delta-sumários podem ser usadas em ambientes móveis em função da redução da quantidade de dados que serão enviadas para as plataformas móveis, diminuindo o volume de comunicação na rede, o tempo necessário de conexão e os custos destas. Além disso, tabelas delta-sumários aceleram a atualização de tabelas sumarizadas;
- Definição de um método para a criação de tabelas sumarizadas e tabelas delta-sumário através de um lattice. A hierarquia do lattice facilita e acelera o processo de atualização de tabelas sumarizadas, aumentando o tempo de disponibilidade do DW;
- Divisão da tarefa de atualização de tabelas sumarizadas em duas funções: propagate e refresh, aumentando o tempo de disponibilidade DW.

Além disso, o trabalho apresenta um estudo detalhado sobre funções que podem ser usadas em um processo de manutenção incremental, bem como, soluções para as restrições existentes.

Quanto às limitações para o uso do trabalho em ambientes móveis destacamos:

- Não define uma arquitetura para ambientes móveis;
- Não aborda a gerência de vários DWs;
- Os DWs necessitam acessar a tabela base para calcular valores de mínimo e máximo, aumentando o volume de comunicação na rede, se as tabelas base estiverem nas fontes;
- A carga de trabalho da função Refresh é muito elevada principalmente para recalcular os valores de mínimos e máximos, apesar da divisão do trabalho com a função propagate;
- Elevada carga de trabalho do DW, pois todo o processamento (propagate e refresh) é executado na plataforma onde se encontra o DW, não levando em consideração as limitações de algumas plataformas móveis;
- Os algoritmos propostos são fracamente acoplados ao SGBD (cursor). O próprio trabalho refere-se a necessidade de aumentar o acoplamento ao SGBD;
- Propagação das alterações das fontes em tabelas delta-sumário não é eficiente, pois o algoritmo não escolhe a tabela ancestral que mais eficientemente criará uma determinada tabela delta-sumário. Por exemplo, baseado na figura 2.3, o algoritmo não define se a tabela sd_R_Vendas será criada a partir da tabela sd_LC_vendas ou a partir da tabela sd_CD_vendas.

2.2 Proposta de Chan *et al* [CLS00]

O trabalho propõe um algoritmo eficiente de atualização incremental para data warehouse em ambientes web. As características mais importantes desta abordagem são:

- Existência de caches, tendo como principal objetivo, a redução do acesso às tabelas fontes para recalcular valores mínimos e máximos excluídos de visões agregadas;
- O recálculo dos valores mínimos e máximos, quando necessário, será executada pela função propagate, diminuindo a carga de trabalho da atualização (refresh), aumentando o tempo de disponibilidade do DW.

2.2.1 Arquitetura de um Data Warehouse baseado na web

Na arquitetura do sistema, apresentada na figura 2.4, existe um *monitor* conectado às fontes de dados, responsável pela detecção das mudanças ocorridas nas fontes através de arquivos de logs. Os logs, que contém o conjunto de alterações na tabela base, serão lidos por servidores web periodicamente, cada um mantendo um conjunto de visões para seus clientes.

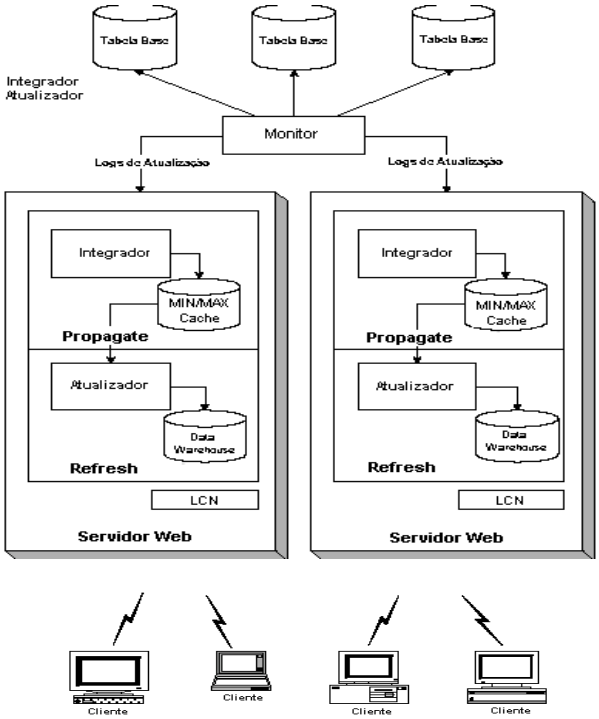


Figura 2.4 - Arquitetura [CLS00]

O *integrador*, localizado no servidor web, lê os logs do monitor, durante a execução da função propagate; e então traduz, filtra e integra informações relevantes, antes de passar para o refresh. Um outro papel importante da função propagate é a atualização dos caches de mínimo e máximo. O cache de mínimo contém os valores mais baixos e o cache de máximo contém os valores mais altos de um campo particular na tabela base.

No *atualizador*, os logs e os caches (min/max) são usados para atualizar as visões materializadas armazenadas no DW. A atualização é feita pela função refresh.

O servidor web pode deixar de acessar, por um longo tempo, o monitor e para satisfazer os longos períodos de desconexão, é mantido um número (LCN) em cada DW, que deverá ser sincronizado com o máximo LCN da tabelas de log, durante um refresh.

Quando o período de desconexão é muito longo, temos que fixar um limite do número de alterações e passando deste, a reconstrução das tabelas sumarizadas é indicada.

2.2.2 Criando log de atualização

Os logs contêm um conjunto de mudanças responsáveis pelas atualizações de uma ou mais visões materializadas, que serão recuperadas pelo servidor web.

Quando ocorre uma inserção, exclusão ou alteração em uma tabela base, um trigger é disparado a fim de inserir logs em uma determinada tabela de log. Múltiplas atualizações de logs são possíveis, onde cada log é responsável pela atualização de uma determinada visão agregada.

A estrutura da tabela de log contém quatro partes: chave da visão materializada, valor delta de campos que são agregados na visão materializada, timestamp e um campo contador que somente é necessário quando existem operadores COUNT(*) armazenados na visão agregada. O campo contador é usado para indicar o número de inserções e exclusões feitas na tabela base.

Para uma inserção, o mais novo valor é inserido no campo delta e +1 é armazenado no campo contador. Para uma exclusão, o valor negativo é armazenado no campo delta e no contador é armazenado -1. Cada operação de atualização é considerada como uma exclusão seguida por uma inserção.

As tabelas 2.2 e 2.3 mostram como os logs são construídos, para manter a visão LPD_Vendas e LC_vendas, respectivamente, apresentadas no exemplo da seção 2.1.2.

Loja	Produto	Dia	TotalCount	Menor_qtd	TotalQuantidade	Timestamp
Para inserção em uma tabela base						
Nova loja	Novo produto	Nova Dia	+ 1	+ Novo qtd	+ Novo qtd	LCN
Para exclusão em uma tabela base						
Velha loja	Velho produto	Velha Dia	- 1	+ Velho qtd	- Velho qtd	LCN

Tabela 2.2: Estrutura de log da visão LPD_VENDAS

Loja	Categoria	TotalCount	Menor_qtd	TotalQuantidade	Timestamp
Para inserção em uma tabela base					
Nova loja	Nova categoria	+ 1	+ Novo qtd	+ Novo qtd	LCN
Para exclusão em uma tabela base					
Velha loja	Velha categoria	- 1	+ Velho qtd	- Velho qtd	LCN

Tabela 2.3: Estrutura de log da visão LC_VENDAS

2.2.3 Algoritmo de atualização incremental

O algoritmo de atualização é dividido em propagate e refresh.

- **Função Propagate**

O objetivo principal da função propagate é atualizar os caches de mínimo e máximo. O cache de mínimo e máximo é carregado com um número de valores candidatos mínimos e máximos, obtidos na tabela base, quando o servidor web é inicializado; e também através de reconstrução. A estrutura do cache é:

MINCACHE(chave_visão, valor, contador) e

MAXCACHE(chave_visão, valor, contador),

em que *chave_visão* é a chave da visão materializada que ela serve, *valor* é o valor candidato mínimo ou máximo e *contador* é o número de ocorrências de valores candidatos de mínimo ou máximo.

A abordagem permite a definição do número de candidatos por chave através da fixação de um parâmetro (*v_size*).

Para manter a visão LC_vendas, apresentada na seção 2.1.2, é necessário um cache mínimo, motivado pela presença da função de agregação *mínimo* na tabela sumarizada. O cache denominado, *LC_vendas_min*, terá a seguinte estrutura:

LC_Vendas_MIN (loja, categoria, qtd, contador)

em que *loja* e *categoria* representam a chave da visão LC_Vendas, *qtd* é um valor mínimo candidato e *contador* é a quantidade de um determinado valor candidato para uma determinada chave.

A função *propagate* examina a tabela de log e determina as mudanças a serem propagadas para os caches de mínimo e máximo. Se o campo contador, da tabela log, for diferente de zero (COUNT(*)<0 ou COUNT(*)>0), uma inserção ou uma exclusão será aplicada ao cache mínimo ou máximo.

Quando uma exclusão é feita, é possível que todas as tuplas para uma dada chave tenham sido excluídas. Nesse caso é necessária a reconstrução do cache para aquele valor chave, consultando a tabela base.

Quando uma inserção é feita, se tuplas com as mesmas chaves já existirem, o valor do contador é acrescido em uma unidade, porém, em um banco de dados que permita duplicação, a tupla simplesmente é inserida.

Se nenhuma tupla com a mesma chave está em cache, ela é inserida na tabela de cache somente quando o valor, na tabela de log, é menor que o mais alto valor do cache mínimo ou maior do que o mais baixo valor do cache máximo.

- **Função Refresh**

O *refresh* aplica os dados das tabelas cache mínimo/máximo e das tabelas de log às visões materializadas armazenadas no data warehouse. Para cada tupla t' na tabela de log, uma correspondente tupla t em uma visão materializada é procurada. se t não existe, t' é uma nova tupla inserida em t.

Caso t tenha sido encontrado e se contador de t adicionado à soma dos contadores de t' for igual a zero, a tupla t é excluída. Caso contrário, as funções de agregação de t são atualizadas com os valores de t' e/ou através do cache mínimo ou máximo.

2.2.4 Críticas

O trabalho tem como principal ponto forte, a manutenção de caches com valores candidatos de mínimos e máximos, utilizados na atualização de visões materializadas com estas funções de agregação, diminuindo o volume de comunicação entre o DW e as fontes em relação à [MQM97].

Um outro elemento importante foi o aumento do tempo de disponibilidade do DW, motivado pela diminuição da carga de processamento da função atualização (refresh), em conseqüência da transferência do cálculo dos valores de mínimo e máximo para a função propagate.

Finalmente, destacamos a arquitetura apresentada, a qual permite a manutenção de vários DWs em servidores web através do LCN.

Entretanto, esta abordagem não permite sua plena aplicação em ambientes móveis em função dos seguintes aspectos:

- O cache é localizado em cada DW, aumentando o volume de comunicação na rede e a carga de trabalho do DW;
- Concentração do Processamento no DW. Embora esse algoritmo apresente uma redução da carga de trabalho da função refresh, todas as duas funções são executadas no DW, sobrecarregando-o e aumentando o tempo de atualização;
- Processamento fracamente acoplado ao SGBD, executado através de cursor, semelhante à [MQM97];

- Não utiliza hierarquias de visões. As hierarquias aceleram o trabalho de atualização de um DW;
- Elevado volume de comunicação na rede. A comunicação em ambientes móveis constitui uma das limitações, em função dos custos e da velocidade dos canais de comunicação. Nesse algoritmo, o volume de dados é muito elevado por duas razões, definidas abaixo:

  - Não utiliza tabelas delta-sumários. A transferência de dados do monitor para o DW é feita usando todas as tuplas do log, sem uma sumarização (delta-sumário). Esta transferência do log inteiro, baseia-se na necessidade de atualizar os caches localizado no DW utilizando os arquivos de logs obtidos;
  - Reconstrução do cache executado no DW utilizando dados das fontes. Para executar a reconstrução dos caches nos DWs, o algoritmo utiliza dados das fontes, aumentando a troca de mensagens entre o DW e as fontes.

2.3 Proposta de Labio *et al* [LYC+99]

Este trabalho apresenta diferentes algoritmos de manutenção incremental de visões agregadas, bem como, intensas avaliações experimentais sobre essas diferentes alternativas.

2.3.1 Arquitetura

A arquitetura, desse trabalho, apresentada na figura 2.5, foi baseada em um protótipo de um sistema de data warehousing de Standford, denominado de WHIPS (WareHouse Information Processing System) [WGL+96].

O protótipo é composto de extratores, um integrador e um mantenedor de Warehouse.

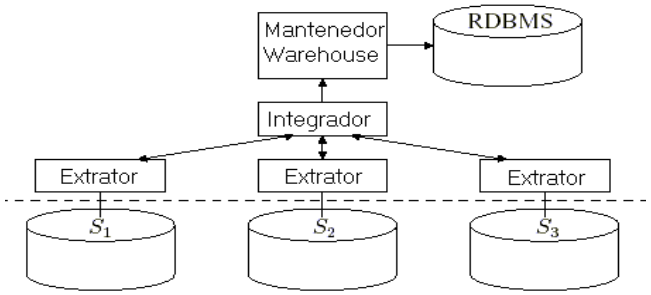


Figura 2.5: Arquitetura WHIPS [WGL+96]

Cada *Extrator* detecta as atualizações (inserções, exclusões e alterações) nas fontes de dados.

O *Integrador* recebe os deltas detectados pelos extratores e calcula um conjunto de deltas para serem enviados para o DW.

O *Mantenedor do Warehouse* recebe o conjunto dos deltas do Integrador e atualiza todas as visões do DW, através de seqüências de instruções SQL enviadas pelo mantenedor.

2.3.2 Representação de visões e instalação de deltas

Existem duas formas de representar visões em DW, uma mantendo duplicações e uma outra mantendo apenas uma tupla e armazenando o número de duplicações em um atributo denominado dupcount. As figuras 2.6 e 2.7 mostram a representação *com* e *sem* duplicações respectivamente.

Loja	Produto	Dia	Qtd
1	a	1	20
1	b	1	250
1	a	1	20

Figura 2.6: Representação com duplicações

Loja	Produto	Dia	Qtd	dupcount
1	a	1	20	2
1	b	1	250	1

Figura 2.7: Representação sem duplicações

A representação sem duplicações acelera a seleção, junção e agregação em função da menor quantidade de tuplas envolvidas.

- **Instalação de exclusões**

Usando a visão PV da seção 2.1.2, se não há duplicações, então as exclusões de PV, denotadas por ∇PV, podem ser instaladas em PV usando uma simples instrução DELETE, mostrada abaixo:

- ✓ DELETE FROM PV WHERE (loja, produto, dia)
  IN (SELECT loja, produto, dia FROM ∇PV)

A instrução acima assume que a chave de PV é loja, produto e dia, porém, quando PV tem duplicações e portanto sem chave, a referida instrução pode excluir mais tuplas do que o necessário. Em geral, na instalação de ∇PV, com duplicações, é necessário o uso de cursor.

Na representação sem duplicações, cada tupla excluída t em ∇PV resulta em uma atualização ou uma exclusão em PV.

Se t.dupcount é menor do que valor do dupcount da tupla em PV que combina com t, nós decrementamos t.dupcount do valor de dupcount de PV, caso contrário a tupla de ∇PV será excluída de PV.

Esse procedimento pode ser implementado com o uso de cursor ou através de instruções SQL (UPDATE e DELETE), com sub-consultas complexas. A instrução DELETE é ilustrada abaixo:

- ✓ DELETE FROM PV WHERE EXISTS (SELECT * FROM ∇PV
  WHERE ∇PV.loja = PV.loja and ∇PV.Produto = PV.Produto
  and ∇PV.dia = PV.dia and ∇PV.dupcount >= PV.dupcount)

- ➢ **Instalação de inserções**

Na representação com duplicações, a instalação de inserções em PV, denotada por ΔPV, pode ser feita usando a instrução INSERT do SQL. Na representação sem duplicações, a instrução INSERT somente pode ser usada se não houver duplicações. Em geral, entretanto, cada tupla t, em ΔPV, resulta em uma atualização ou em uma inserção em PV.

Se existe uma tupla em PV que combina t, a tupla dupcount de PV é incrementada ao t.dupcount. Caso contrário, t será inserido em PV. Novamente, esse procedimento pode ser implementado através de cursor ou através de instruções INSERT e UPDATE com sub-consultas complexas.

2.3.3 Mantendo visões agregadas

Inicialmente, o Mantenedor de Warehouse calcula as variações (deltas) usando consultas específicas, denominadas *expressões de manutenção,* e então instala estes deltas dentro de visões derivadas.

Por exemplo, vamos supor que a visão PV contém as tuplas mostradas na figura 2.8. Uma visão *Produtos*, apresentada na figura 2.9, é definida sobre PV agrupando suas tuplas por produto. A quantidade de cada produto é calculada a partir de PV somando a quantidade de cada produto particular.

A visão *Produtos* também armazena no atributo *tupcount,* o número de tuplas PV que são usados para derivar cada tupla de *Produtos*. A definição SQL de Produtos é apresentada a seguir:

- ✓ CREATE VIEW *Produtos* AS
  SELECT produto, SUM(qtd) AS total,
  COUNT(*) AS tupcount FROM PV GROUP BY produto

Loja	Produto	Dia	Qtd
1	a	1	20
1	b	1	250
2	a	1	20
3	c	1	500

Figura 2.8: Visão PV

Produto	Total	tupcount
a	40	2
b	250	1
c	500	1

Figura 2.9: Visão Produtos

As figuras 2.10 e 2.11 representam tuplas inseridas e excluídas em PV, respectivamente.

Loja	Produto	Dia	Qtd
1	a	1	20
4	c	1	500
4	d	1	30

Figura 2.10: Visão ΔPV

Loja	Produto	Dia	Qtd
1	a	1	20
1	b	1	250

Figura 2.11: Visão ∇PV

A manutenção de visão agregada pode ser feita usando 04 (quatro) algoritmos diferentes: recálculo completo, delta-sumário com instalação baseada em cursor, delta-sumário com instalação em lote e delta-sumário com reescrita.

- **Recálculo completo - Reconstrução (FullRecomp)**

Recálculo completo é conceitualmente simples de ser implementado. Tomando por base a atualização da visão *Produtos*, primeiramente é instalado *∇PV* e *ΔPV* em *PV*, utilizando a instalação de exclusões e inserções, apresentadas na seção 2.3.2. Então, uma nova visão *Produtos* é recriada inteiramente a partir do novo *PV* criado.

- **Delta-Sumário com Instalação baseada em cursor (SDcursor)**

O algoritmo original delta-sumário (SDcursor) para manutenção incremental de visões agregadas tem a fase de cálculo (propagate) e uma fase chamada de instalação (refresh) baseado em cursor [MQM97].

Na fase de cálculo, os conjuntos de mudanças de ΔPV e ∇PV são capturados por uma tabela delta-sumário denominada de sd_Produtos, criada de acordo com a instrução abaixo:

- ✓ SELECT produto, SUM(total) AS total, SUM(tupcount) AS tupcount
FROM ((SELECT produto, SUM(qtd) as total, COUNT(*) AS tupcount
FROM ΔPV GROUP BY produto) UNION ALL
(SELECT produtoID, - SUM(qtd) as total, - COUNT(*) AS tupcount
FROM ∇PV GROUP BY produto))
GROUP BY produto

Na fase de instalação, SDcursor instala as alterações, registradas em sd_Produtos, na visão Produtos, usando cursor, definido pela função refresh de [MQM97].

- **Delta-sumário com instalação em lote (SDbatch)**

O algoritmo delta-sumário com instalação em lote (SDbatch) é uma variação de [MQM97], proposta nesse trabalho. A idéia básica é aumentar o processamento na fase de cálculo, aumentando a velocidade da etapa de instalação.

Fase de Cálculo

Primeiramente é criada a visão sd_Produtos, como mostrado no algoritmo SDcursor. Em seguida é criada a visão ∇Produtos, com as exclusões, conforme a seguir:

- ✓ SELECT * FROM Produtos WHERE produto
IN (SELECT produto FROM sd_Produtos)

Finalmente, são calculadas as inserções (ΔProdutos), conforme abaixo:

- ✓ SELECT produto, SUM(total) AS total, SUM(tupcount) AS tupcount
FROM ((SELECT * FROM ∇Produtos) UNION ALL
(SELECT * FROM sd_Produtos))
GROUP BY produto HAVING SUM(tupcount) > 0

Fase de Instalação

Na fase de instalação do SDbatch, inicialmente é aplicado ∇Produtos na visão *Produtos* usando a instrução DELETE, como a seguir:

- ✓ DELETE FROM Produtos WHERE (produto)
  IN (SELECT produto FROM ∇Produtos)

Posteriormente, aplicamos ΔProdutos em *Produtos*, conforme a seguir:

- ✓ INSERT INTO Produtos (SELECT * FROM ΔProdutos)

- ➢ **Delta-sumário com reescrita (SDoverwrite)**

Essa abordagem substitui completamente o conteúdo de *Produtos* por um novo conteúdo, usando a tabela delta-sumário *sd_Produtos* e a visão antiga de *Produtos*. A instrução SQL é semelhante ao cálculo de ΔProdutos da abordagem SDbatch, conforme a seguir:

- ✓ SELECT produto, SUM(total) AS total, SUM(tupcount) AS tupcount
  FROM ((SELECT * FROM Produtos) UNION ALL
  (SELECT * FROM sd_Produtos))
  GROUP BY produto HAVING SUM(tupcount) > 0

Esta consulta cria uma tabela que, posteriormente, será a nova tabela *Produtos*. Um dos problemas desta abordagem é a necessidade de uma maior espaço de armazenamento.

2.3.4 Experimentos

- ➢ **Instalação de Deltas e Representação de visões**

Inicialmente, foram avaliadas as performances das instalações de inserções e exclusões quanto ao tipo de representação (*com* ou *sem* duplicações) e instalação (Cursor e SQL).

Representação *com* x *sem* duplicações

Nesse estudo foi observado que quando não há duplicações, a representação de visão que permitem duplicações apresenta uma melhor performance. Por outro lado, o aumento do número de duplicações (três repetições) fez com que as instalações de exclusões, na abordagem com duplicações, tornassem duas vezes mais lentas do que a instalação usando a abordagem sem duplicações, porém as instalações das inserções continuaram mais rápidas na abordagem com duplicações.

Uma outra observação é que na abordagem sem duplicações, as performances permanecem constantes independentes do número delas. Enquanto na abordagem com duplicações os tempos das instalações aumentam proporcionalmente ao aumento do número destas.

Instalação de Deltas: Cursor x SQL

Os experimentos demonstraram que para a instalação de exclusões, o cursor apresenta um melhor desempenho, enquanto na instalação de inserções o uso de SQL foi mais bem avaliado.

- **Manutenção de visões agregadas**

Nesse estudo, foram avaliados a performance dos algoritmos FullRecomp, SDcursor, SDbatch e SDoverwrite para manter visões materializadas com percentual fixo de compressão e visões com tamanho fixo.

Manutenção de visões agregadas com percentual fixo de compressão

Nesse experimento, foi usado uma visão com 25% de compressão, ou seja, uma visão V agrupa V dentro de 0,25 x |L| tuplas, onde |L| é o número de tuplas da tabela base, variando o percentual de atualização entre 1% e 10%.

Nesses experimentos, foi observado que os algoritmos SDbatch, SDcursor e SDoverwrite são mais rápidos do que o algoritmo FullRecomp, especialmente para visões muito grande.

Outra observação, é que o SDcursor é preferido para grandes visões com pequenos deltas, enquanto SDoverwrite é indicado para pequenas visões com uma grande quantidade de deltas. SDbatch é indicado para situações entre as duas citadas.

Finalmente, os estudos demonstraram que o algoritmo SDbatch apresenta melhor performance na fase de instalação, porém a vantagem diminui, em relação aos outros, quando a tabela base aumenta.

Manutenção de visões agregadas com tamanho fixo

Esse experimento mostra que SDbatch, SDcursor e SDoverwrite possuem desempenhos semelhantes, porém o SDbatch apresenta melhor performance na fase de instalação.

2.3.5 Críticas

O trabalho apresenta diferentes alternativas de manutenção incremental, bem como oferece critérios para decisão sobre a melhor alternativa a adotar, dependendo da configuração do DW.

Essas decisões baseiam-se em exaustivos experimentos executados sobre diferentes cenários, sendo esse, o grande diferencial do referido trabalho em relação aos outros.

Um outro benefício foi a utilização de instruções SQL em alguns algoritmos (SDbatch e SDoverwrite), em que foi demonstrada sua melhor eficiência em relação a outros (SDcursor e FullRecomp), mostrando que um maior acoplamento ao SGBD permite melhores desempenhos.

Finalmente, o trabalho utiliza, em alguns de seus algoritmos (SDbatch e SDoverwrite), o paradigma de tabelas delta-sumário. O uso desse paradigma diminui o volume de comunicação quando utilizado em ambientes móveis.

Entretanto, o trabalho apresenta algumas restrições para sua utilização em ambientes móveis:

- Não define uma arquitetura para ambientes móveis;
- Não aborda a gerência de vários DWs;
- Propagação das alterações em tabelas delta-sumário não é eficiente, pois o algoritmo não escolhe a tabela ancestral que mais eficientemente criará uma outra tabela delta-sumário;
- Os algoritmos não abordam a manutenção de visões materializadas com função de agregação de mínimo e máximo, bem como outras funções de agregação não clássicas;
- Concentração do Processamento no DW, pois todo o processamento é executado no local onde se encontra o DW.

2.4 Proposta de Stanoi *et al* [SAE+99]

Este trabalho estende a arquitetura de um data warehouse tradicional, permitindo a existência de data warehouse móvel, fontes de dados móveis ou ambas, propondo as seguintes opções de arquitetura:

- Visões materializadas instaladas em um servidor proxy fixo e nos dispositivos móveis;
- Visões materializadas instaladas somente em dispositivos móveis e as alterações das fontes de dados mantidas e extraídas por um servidor proxy fixo;
- Visões materializadas e alterações das fontes localizadas somente nos dispositivos móveis.

Além da arquitetura, o trabalho dá uma grande ênfase na hierarquia de visões como sendo um dos elementos que aceleram o processo de atualização de DWM.

2.4.1 Arquiteturas

O trabalho permite três tipos de arquiteturas: visões materializadas em um servidor proxy e nos dispositivos; visões materializadas somente nos dispositivos móveis; e visões materializadas e alterações das fontes nos dispositivos móveis.

➢ **Visões Materializadas em um servidor proxy fixo e nos dispositivos móveis.**

Nessa arquitetura, mostrada na figura 2.12, as visões materializadas hierárquicas são armazenadas em um servidor proxy fixo.

Os dados são replicados para os dispositivos móveis, a cada conexão com o proxy, atualizando as visões materializadas neles residentes.

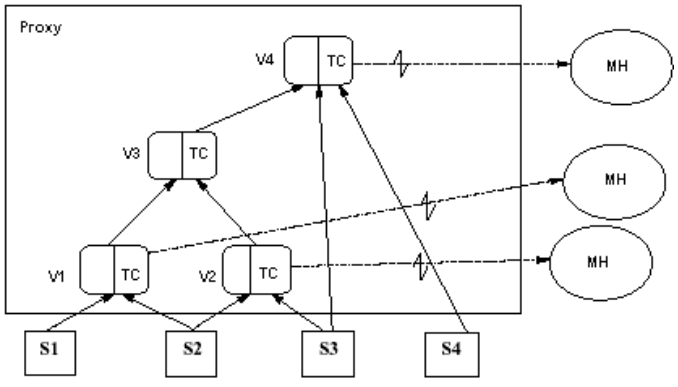


Figura 2.12: Plataformas móveis acessando visões [SAE+99]

O servidor proxy pré-computa e grava as atualizações das fontes em tabelas denominadas de TCs (Table of Changes). Estas tabelas representam o conjunto de mudanças ocorridas nas fontes de dados.

As alterações das fontes, desde a última conexão, são facilmente obtidas com a inclusão do timestamp, em cada tupla, nas visões existentes no proxy.

Esse modelo apresenta algumas vantagens, em relação aos sistemas onde os dispositivos móveis conectam as fontes de dados diretamente:

- Reduz a replicação, pois dispositivos móveis podem usar as mesmas visões pré-definidas;
- O proxy necessita interagir com as fontes de dados somente quando as visões dependem diretamente das fontes;
- Atualizações são pré-computadas e acessíveis pelos dispositivos móveis durante a conexão.

➢ **Visões materializadas somente nos dispositivos móveis**

A arquitetura, mostrada na figura 2.13, elimina a necessidade de armazenar visões explicitamente no servidor proxy.

O principal objetivo do proxy é a pré-computação de alterações ocorridas nas fontes para serem usadas pelos dispositivos móveis, diminuindo o volume de comunicação entre a plataforma móvel e o proxy.

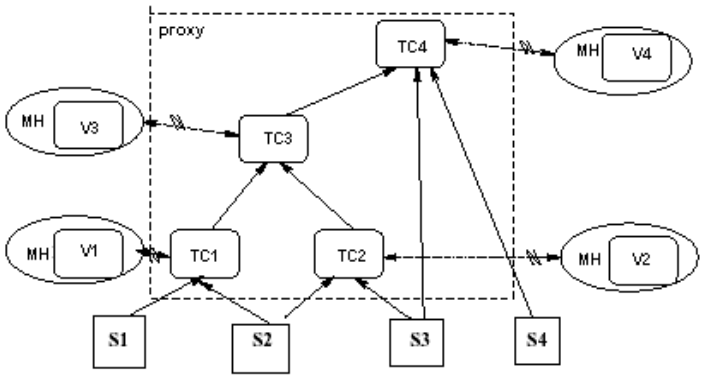


Figura 2.13: Plataformas móveis que armazenam visões materializadas [SAE+99]

Nessa arquitetura, não é necessário o armazenamento de visões no proxy, diminuindo os requisitos de armazenamento no servidor. A hierarquia, nesta arquitetura, não é formada por visões, mas por tabelas de mudanças (TC).

A manutenção incremental das tabelas de mudanças (TC), nessa configuração, é relativamente complexa, pois pode ser necessário acessar os estados de visões materializadas que residem em dispositivos desconectados.

Por exemplo, para calcular as mudanças em TC_3, da figura 2.13, devido a atualizações da tabela fonte S_2, deve ser usada a seguinte consulta:

$$\Delta TC_3 = \Delta TC_1 \bowtie (TC_2 + \Delta TC_2 + V_2) \bowtie (TC_1 + V_1) \bowtie \Delta TC_2$$

Essa consulta denota que para calcular as novas alterações em TC_3, não são necessários somente os dados armazenados nas tabelas TC_1 e TC_2, mas também os dados das visões V_1 e V_2. Uma solução para diminuir acessos às fontes é manter relações base no proxy.

- **Visões materializadas e alterações das fontes nos dispositivos móveis**

Nessa arquitetura, mostrada na figura 2.14, elimina-se a necessidade de um servidor proxy fixo, pois o data warehouse e as alterações das fontes estão localizados nas plataformas móveis. Os dispositivos móveis são responsáveis pela atualização de suas visões materializadas, durante uma conexão.

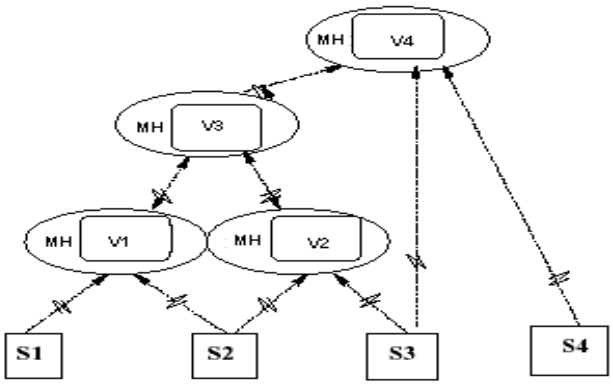


Figura 2.14: TCs e visões armazenadas na plataforma móvel [SAE+99]

Nesse caso, a plataforma móvel consulta outros dispositivos ou comunica-se diretamente com as fontes para manter suas visões. A principal dificuldade dessa arquitetura é obter informações de clientes desconectados.

Uma alternativa é replicar cada relação base para os dispositivos móveis e as alterações das fontes enviadas através de arquivos de logs obtidos durante uma conexão, porém essa solução exige uma grande capacidade de armazenamento no dispositivo móvel.

2.4.2 Críticas

O trabalho tem como principal característica, as diferentes sugestões de arquiteturas apresentadas para tratar desconexões em DWMs, principalmente com a adoção de um servidor proxy fixo responsável pela detecção de alterações ocorridas nas fontes e pela preparação de dados alterados a serem enviados para os dispositivos móveis. Além disso, o servidor proxy fixo permite a gerência de vários DWMs.

Entretanto, o trabalho apresenta algumas restrições:

- Não fornece algoritmos para atualização de visões materializadas com funções de agregação de mínimo e máximo, bem como outras funções de agregação não clássicas;

- Não utiliza tabelas delta-sumário, aumentando o volume de comunicação entre as plataformas móveis e o servidor proxy;
- Não apresenta uma estratégia de comunicação entre as plataformas móveis e o proxy;
- Não elimina o acesso direto entre os DWMs e as fontes de dados.

2.5 Proposta do Oracle9i [Ora02]

A replicação no Oracle 9i suporta uma variedade de aplicações com requisitos diferentes, tais como aplicações com visões dispostas remotamente, sendo necessário uma sincronização periódica entre o banco de dados central e um grande número de bancos de dados remotos; aplicações com múltiplos servidores, sincronizados continuamente; e sistemas que combinam aspectos de ambos tipos de aplicações.

A replicação no Oracle 9i replica dados em diferentes versões do Oracle e em diferentes sistemas operacionais que executam o Oracle, podendo criar visões materializadas a partir de uma ou mais visões, conforme mostra a figura 2.15; além de permitir a atualização incremental, denominada *rápida atualização* (fast refresh).

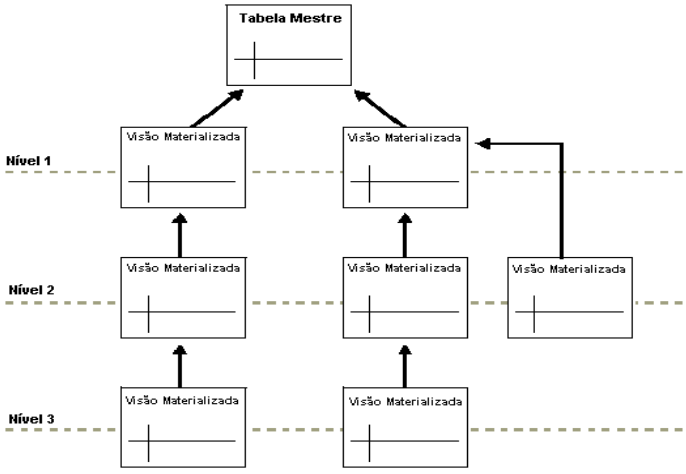


Figura 2.15: Visões materializadas em múltiplos níveis [Ora02]

Em um ambiente de replicação, qualquer atualização feita em um determinado objeto, faz com que estas atualizações sejam aplicadas às copias existentes em todos os locais. Os objetos replicados são: tabelas, índices, visões, pacotes, Procedures, Funções, Tipos definidos por usuário, sinônimos, dentre outros.

2.5.1 Tipos de ambientes de replicação

A replicação suporta os seguintes tipos de ambientes de replicação: replicação com múltiplos bancos de dados mestres, replicação de visões materializadas e configuração híbrida.

- **Replicação com múltiplos bancos de dados mestres**

Na replicação com múltiplos bancos de dados mestre, mostrada na figura 2.16, as atualizações feitas em um banco de dados mestre são propagadas para todos os outros bancos de dados mestres.

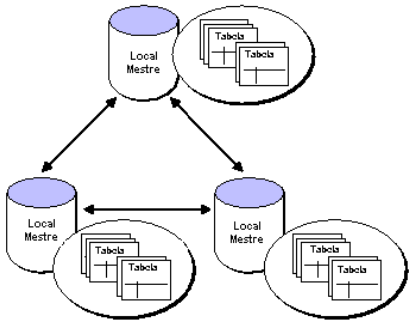


Figura 2.16: Replicação com múltiplos bancos de dados mestres [Ora02]

- **Replicação de visões materializadas**

Uma visão materializada em um banco de dados pode conter uma cópia completa ou parcial de dados de um banco de dados mestre, em um determinado momento no tempo. A figura 2.17 mostra a atualização de uma visão materializada a partir de uma tabela situada em um banco de dados mestre.

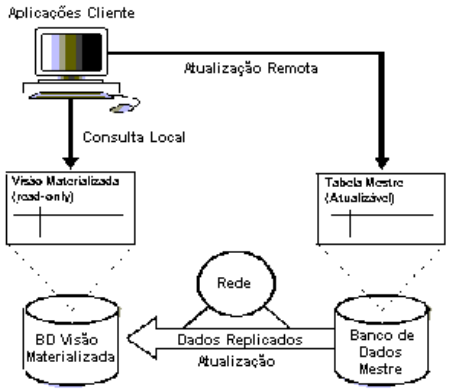


Figura 2.17: Replicação de uma visão materializada [Ora02]

- **Configurações híbridas**

Replicação de múltiplos bancos de dados mestres e de visões materializadas podem ser combinadas em uma configuração híbrida para atender diferentes requisitos de aplicações.

2.5.2 Tipos de visões materializadas suportadas pelo Oracle

O Oracle permite três tipos de acessos às visões materializadas: visão materializada somente para leitura, atualizável ou com capacidade de escrita.

- **Visão Materializada somente para leitura**

Em uma configuração básica, visões materializadas podem prover acesso somente para leitura de dados de tabelas originalmente dispostas em bancos de dados locais ou remotos.

Aplicações podem acessar dados de visões materializadas somente para leitura para evitar acesso à rede.

As visões materializadas somente para leitura eliminam a possibilidade de conflitos porque não podem ser atualizadas e suportam visões complexas.

- **Visão Materializada atualizável**

Uma visão materializada atualizável permite que usuários insiram, atualizem e excluam tuplas. Uma visão materializada atualizável é baseada em tabelas ou em outras visões materializadas.

A figura 2.18 mostra que aplicações podem consultar e atualizar visões materializadas no local onde se encontra a visão. Essas aplicações podem também atualizar tabelas em um banco de dados mestre. Na atualização, a visão materializada é atualizada com os novos dados do banco de dados mestre e o mestre é atualizado com os novos dados da visão materializada.

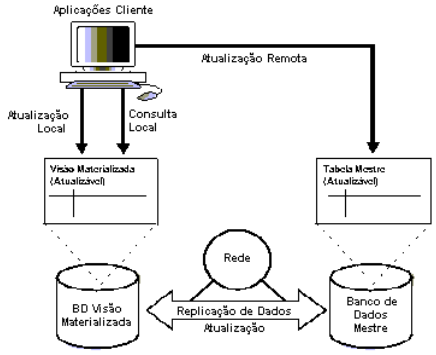


Figura 2.18: Visão materializada atualizável [Ora02]

- **Visões Materializadas com possibilidade de escrita**

O Oracle permite a criação uma visão materializada usando a cláusula FOR UPDATE. Nesse caso, usuários podem executar qualquer alteração através de instruções DML, mas estas alterações não podem ser enviadas para um mestre, portanto serão perdidas se uma visão for atualizada.

2.5.3 Opções de atualização de visões materializadas

Uma visão materializada possui duas opções de atualização: como atualizar e qual o tipo de atualização. Se não for especificado, o padrão assumido é ON DEMAND e FORCE.

Os dois tipos de atualização são: ON COMMIT e ON DEMAND. Dependendo do tipo de visão materializada alguma opção pode não está disponível. A Tabela 2.4 descreve os tipos de atualização.

Modo de Atualização	**Descrição**
`ON COMMIT`	A atualização ocorre automaticamente quando uma transação recebe um commit, ou seja, quando uma tabela mestra ou uma visão materializada mestra é modificada, todas as materializações, dependentes destas, são atualizadas no momento do commit. Este tipo de atualização é possível quando a visão é FAST REFRESH.
`ON DEMAND`	A atualização ocorre através de uma solicitação explícita do usuário, ou seja, as atualizações das materializações dependentes de tabelas ou de outras visões materializadas que foram modificadas, somente serão atualizadas quando ocorre uma solicitação explícita de um usuário.

Tabela 2.4: Tipos de atualizações [Ora02]

Quando uma visão materializada é mantida usando o método ON COMMIT, o tempo requerido para completar o commit demorará mais do que o normal. Isso acontece porque a operação de atualização de todas as visões materializadas, dela dependente, é parte do processo.

Caso ocorra uma falha na visão materializada, durante a atualização, no momento da execução do COMMIT, deverá ser solicitada a atualização explicitamente após a verificação dos motivos da falha. Enquanto a falha não for resolvida a visão materializada não será atualizada.

Além de especificar o tipo de atualização, é possível escolher como a visão materializada será atualizada. Existem quatro opções: `COMPLETE`, `FAST`, `FORCE` e `NEVER`. A Tabela 2.5 descreve as opções de atualização.

Opção de Atualização	**Descrição**
`COMPLETE`	A atualização é feita recalculando completamente a consulta que define a visão materializada. Esta atualização na realidade reconstrói totalmente a visão materializada.
`FAST`	Aplica as mudanças incrementais a fim de atualizar uma visão materializada usando informações em logs, definidas a partir do create materialized view log.
`FORCE`	Aplica a rápida atualização (FAST REFRESH) se possível. Não sendo possível a atualização será completa.
`NEVER`	Indica que a visão materializada não será atualizada com os mecanismos de atualização do Oracle.

Tabela 2.5: Opções de atualização [Ora02]

2.5.4 Atualização incremental de uma visão materializada

Para haver uma atualização incremental é necessária a existência de um log de uma visão materializada situado no local da tabela mestre ou da visão materializada mestra, gravando todas as mudanças ocorridas nestas. O log é associado a uma única tabela ou visão e cada um destes possui apenas um log.

Uma atualização incremental de uma visão materializada somente é possível se a tabela ou visão mestra possui um log, pois o banco de dados o utiliza para efetuar a atualização.

2.5.5 Críticas

A replicação no Oracle 9i possui as seguintes características que podem ser adotadas para DWM:

- Arquitetura suporta plataformas móveis;
- A abordagem permite a extração incremental de dados através de arquivos de logs;
- As visões materializadas podem ser atualizadas incrementalmente;
- A replicação suporta diversos objetos de um SGBD, tais como: tabelas, índices, visões, pacotes, Procedures, Funções, Tipos definidos por usuário, sinônimos, dentre outros;
- A atualização incremental suporta outras funções de agregação.

Entretanto a replicação utilizada pelo Oracle possui as seguintes limitações:

- A atualização incremental de visões materializadas no Oracle, não pode ser utilizada se houver diferentes Sistemas de Gerência de Banco de Dados na arquitetura, ou seja, as plataformas móveis e o servidor devem utilizar o Oracle como SGBD;
- A quantidade de tuplas do log na replicação do Oracle é superior ao número de tuplas quando é usada a abordagem delta-sumário, aumentando o volume de comunicação;
- A quantidade de colunas no log na replicação do Oracle é superior ao número de colunas das tabelas delta-sumário, em função da presença de várias colunas adicionais, como por exemplo ROWID. Estas colunas adicionais têm como objetivo permitir exclusões e alterações de tuplas em visões materializadas além das inserções;
- O tamanho da visão materializada pode ser maior em função de colunas extras necessárias a fim de permitir que tuplas em visões materializadas possam ser excluídas ou modificadas, ocupando um maior espaço de armazenamento em disco;

- Na replicação do Oracle, as funções MIN (expr) e MAX(expr) somente aceitam a manutenção incremental em caso de inserções;
- Os requisitos de armazenamento de um banco de dados são muito grandes em função da presença de visões materializadas mestres e dos arquivos de logs.

2.6 Características de um gerente de Data Warehouses Móveis

As principais características necessárias de um gerente de Data Warehouses Móveis são:

- **Gerência de DWMs simultâneos**

Uma corporação pode ter DWMs em variados estados de atualização em função dos diferentes períodos de desconexão com as fontes de dados. O gerente deve ser responsável pela manutenção incremental dos diversos DWMs, independente do tempo de desconexão com as fontes e do número de atualizações simultâneas solicitadas pelas plataformas móveis.

- **Servidor proxy fixo extraindo alterações das fontes e executando um pré-processamento**

A arquitetura de um gerente de DWMs deve ser dotada de um servidor central fixo, denominado proxy, encarregado de receber as alterações incrementais das fontes de dados e executar um pré-processamento, com o objetivo de diminuir o tráfego de rede e efetuar parte do processo de atualização dos DWMs fora da plataforma móvel.

Parte do processo de atualização dos DWMs, executado no proxy, diminui a carga de trabalho a ser executada nas plataformas móveis, aumenta a velocidade de atualização dos DWMs e o tempo de disponibilidade dos DWMs para consultas dos usuários.

- **Manutenção de visões materializadas com funções de agregações mínimo e máximo**

O gerente deve permitir a manutenção de visões materializadas com funções de agregação de mínimo e máximo, apesar de sua complexidade por não se tratar de funções *self-maintainable*, agravada pelas limitações impostas pela mobilidade.

- **Isolamento completo entre o DWM e as fontes**

Devido ao alto custo e baixa velocidade dos canais de comunicação sem fio, o gerente deve eliminar completamente o acesso direto entre os DWMs e as fontes de dados, restringindo o acesso unicamente ao servidor proxy. Esta eliminação além de reduzir o tempo e o custo para se obter dados necessários para a atualização dos DWMs, aumenta a velocidade de atualização dos DWMs e conseqüentemente o tempo de disponibilidade dos DWMs para consultas.

- **Atualização de visões materializadas através tabelas delta-sumário**

As tabelas delta-sumário [MQM97] têm como objetivo principal reduzir o tempo para atualização das visões materializadas dos DWMs, pois resumem as alterações das fontes agrupando-as com os mesmos critérios de agregação das visões materializadas.

- **Algoritmos acoplados a SGBDs**

As instruções SQL usadas nos algoritmos do gerente devem ser fortemente acopladas a SGBDs, utilizando comandos DML sobre um grupo tuplas ao invés de comandos DML sobre tuplas individuais como ocorre com a utilização de cursor. Este acoplamento aumentará a velocidade de atualização dos DWMs.

- **Processamento de atualização do DW não concentrado na plataforma móvel**

Uma das funções importantes do proxy é executar parte do processo de atualização dos DWMs fora da plataforma móvel, principalmente no que se refere ao agrupamento das alterações e ao cálculo dos novos valores de mínimos e máximos, reduzindo a carga de trabalho executada nos DWMs, levando em consideração as possíveis limitações das plataformas móveis.

- **Propagação de alterações no servidor proxy otimizada automaticamente**

Um dos processos realizados no proxy é a propagação das alterações das fontes em tabelas delta-sumários e cache-sumários através de uma hierarquia. A propagação automática permite definir, baseado no menor custo de propagação, as tabelas delta-sumários e cache-sumários que mais eficientemente atualizarão outras tabelas delta-sumários e cache-sumários. Esta propagação visa aumentar a eficiência do processo de preparação executado no proxy.

- **Estratégia de comunicação entre o DWMs e o Servidor Proxy**

A obtenção de dados do proxy necessários para atualização dos DWMs deve utilizar uma estratégia de comunicação entre os DWMs e o proxy baseado em um reduzido tráfego de rede e uma reduzida carga de trabalho do servidor proxy em função dos acessos concorrentes de diversos DWMs.

- **Novas funções de agregação**

O SQL é dotado de diversas funções de agregação estatísticas importantes para um processo de suporte à decisão, tais como variância, desvio padrão, análise de regressão, covariância dentre outros. O gerente deve permitir a manutenção incremental de visões materializadas com estas novas funções de agregação.

A Tabela 2.6 apresenta um resumo comparativo das propostas estudadas em relação às características necessárias de um gerente de DWMs.

Característica	**Propostas**				
	MQM97	**CLS00**	**LYC+99**	**SAE+99**	**Oracle**
Gerência de DWMs simultâneos	Não	Não	Não	Sim	Sim
Servidor proxy fixo extraindo alterações das fontes e executando um pré-processamento.	Não	Não	Não	Sim	Não
Manutenção de visões materializadas com funções de agregações mínimo e máximo.	Sim	Sim	Não	Não	Somente inserção
Isolamento completo entre o DWM e as fontes.	Não	Não	Não	Não	Não
Atualização de visões materializadas através tabelas delta-sumário.	Sim	Não	Sim	Não	Não
Algoritmos acoplados ao SGBD.	Não	Não	Sim	Não	Sim
Processamento de atualização não concentrado no DW.	Não	Não	Não	Sim	Não
Propagação de alterações no servidor proxy otimizada automaticamente	Não	Não	Não	Não	Não
Estratégia de comunicação entre DWM e Servidor Proxy	Não	Não	Não	Não	Não
Novas funções de agregação	Não	Não	Não	Não	Parcial

Tabela 2.6: Características das propostas

Capítulo 3

MDWManager - Um gerente de Data Warehouses Móveis

Este capítulo tem como objetivo apresentar detalhadamente o projeto e a construção de um gerente de DWMs, denominado de MDWManager, responsável pela manutenção de diversos DWs instalados em plataformas móveis, atendendo às restrições impostas pela mobilidade.

As principais características do MDWManager são:

- Gerência de vários DWMs simultâneos;
- Manutenção incremental do DWMs;
- Isolamento completo entre os DWMs e as fontes de dados;
- Diminuição do volume de comunicação na rede, em função do tempo e do custo de conexão;
- Diminuição da carga de processamento na plataforma móvel, aumentando a velocidade de atualização e o tempo de disponibilidade dos DWMs;
- Algoritmos acoplados a SGBD, beneficiando-se das otimizações existentes nos bancos de dados;
- Estratégia de comunicação entre as plataformas móveis e o servidor proxy;
- Manutenção de visões materializadas com funções de agregação de mínimo e máximo; e novas funções de agregação

3.1 Modelo de dados

Um Data Warehouse (DW) é usado para responder consultas complexas sobre uma grande quantidade de dados com muita rapidez e precisão, entretanto o tamanho do DW e a

complexidade das consultas aumentam o tempo de resposta das consultas solicitadas [HAR96].

Armazenar as consultas mais freqüentemente solicitadas em diversas visões materializadas é a técnica mais indicada para melhorar a performance em Data Warehouse [BM90, GBL+96, KAM93, Wid95a], havendo três níveis de materialização [HAR96]: nenhuma materialização, materialização completa, materialização parcial.

O modelo do MDWManager utilizará a materialização parcial ou completa, entretanto, essas materializações impõem algumas restrições, dentre elas, o tempo necessário para atualizá-las.

Uma das técnicas para acelerar esse processo é criar visões materializadas a partir de outras visões, ao invés de utilizar uma tabela base. Como exemplo, considere um DW com uma tabela de fatos (V_1) que agrupa dados da tabela *vendas* usando três dimensões: *produto*, *loja* e *dia*. A visão V_1 é criada a partir da seguinte instrução SQL:

```
✓ Create View V1 as
  Select Produto, Loja, Dia, Sum(qtd), count(*)
  From Vendas
  Group By  Produto, Loja, Dia
```

Outras visões materializadas podem ser criadas a partir desta visão, uma delas, a visão V_2, em que os valores de todos os dias são agregados, formando uma nova visão com o agrupamento por produto e loja. Na realidade a dimensão *dia* foi substituída pelo valor "all" (produto, loja, all).

A visão V_2 criada, representa uma agregação de valores (produto, loja, dia_1), ... , (produto, loja, dia_n), em que n representa o número de dias que foram agrupados.

A visão V_2 pode ser criada através da seguinte instrução SQL:

```
✓ Create View V2 as
  Select Produto, Loja,  Sum(qtd), count(*)
  From V1
  Group By  Produto, Loja
```

A diferença básica para criação da visão materializada V_2, em relação à criação da visão V_1, resume-se à mudança da cláusula FROM, onde não será usada a tabela base *vendas*

mas a visão V_1 e a cláusula GROUP-BY, em que dia é substituído por "all", devendo ser omitida.

3.1.1 Lattice de um cubo de dados

A estrutura de um lattice[1] é uma forma gráfica de representar um cubo de dados multidimensional com uma, duas ou mais dimensões, onde são apresentadas as diferentes possibilidades de agrupamento dos atributos de dimensão da tabela de fatos.

O número de possibilidades de agrupamento é determinado pelo número de atributos de dimensão da tabela de fatos, expressado por 2^k, em que k representa o número de dimensões [MQM97].

Considerando um exemplo, em que existem três atributos de dimensão: produto, loja e dia, temos 08 (oito) combinações de agrupamento, formando um lattice como o apresentado na figura 3.1. O vértice V8 representa a ausência de agrupamento.

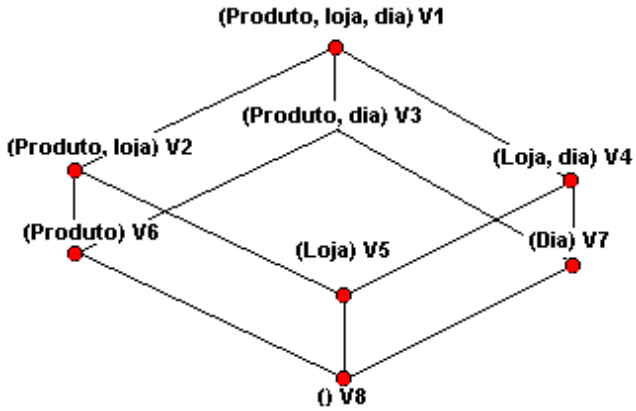


Figura 3.1: Lattice de cubo de dados [MQM97]

3.1.2 Relação de dependência

Quando uma consulta pode ser obtida a partir de uma outra, existe uma relação de dependência. Considerando duas consultas C_1 e C_2, $C_2 \leq C_1$, se C_2 pode ser respondida a

[1] Usamos o termo inglês *lattice* para denotar *reticulado*

partir de C_1, portanto C_2 é dependente de C_1. Essa relação de dependência estende-se também para a possibilidade de criação de uma visão materializada utilizando uma outra visão.

Na figura 3.1, $V_2 \leq V_1$, pois a visão V_2 pode ser criada a partir da visão V_1, ao invés de acessar uma tabela base nas fontes. Pode-se observar também que é possível criar a visão V_5 (loja) a partir da visão V_2 (produto, loja) ou a partir da visão V_4 (loja, dia), existindo, portanto as seguintes relações de dependências $V_5 \leq V_2$ e $V_5 \leq V_4$.

Além disso, o operador "≤" impõe uma ordenação de consultas ou de geração de visões. No exemplo da geração da visão V_5, a ordenação implica que as visões V_2 e V_4 devem ser criadas anteriormente à visão V_5.

Nesse modelo de dados, o custo de geração de uma visão será determinado pelo número de linhas examinadas para gerar a nova visão. Isto quer dizer que, para gerar a visão V_5 deverá ser usado o critério de menor números de linhas existentes entre V_2 e V_4.

3.1.3 Lattice parcialmente materializado

Um lattice pode ser parcialmente materializado através da remoção de alguns vértices do lattice totalmente materializado. Quando um vértice v é removido, todos os arcos de entradas e saídas destes vértices também são removidos e novos arcos são adicionados entre os vértices de saída e de entrada do vértice eliminado.

Por exemplo, removendo-se o vértice V_2 de um subconjunto de um lattice com as relações de dependência $V_3 \leq V_2 \leq V_1$, tem-se uma nova relação de dependência $V_3 \leq V_1$. Isto significa que a visão V_3 gerada a partir de V_2, será obtida a partir de V_1, indicando que, em uma consulta SQL, a cláusula FROM com V_2, será substituída por V_1.

3.1.4 Lattice de visões: um dígrafo

Lattice é um conjunto parcialmente ordenado de elementos L e relações de dependências $\leq$ denotado por $(L, \leq)$. Para os elementos v e w de um lattice $(L, \leq)$, $v \leq w$ significa que $v < w$ e $v \neq w$.

Os ancestrais e descendentes de um elemento de um lattice $(L, \leq)$, são definidos como a seguir:

$$\text{ancestral}(v) = \{w \mid v \leq w\}$$

$$\text{descendente}(v) = \{w \mid w \leq v\}$$

O ancestral imediato de um elemento v é denotado por *imediato(v)*, onde:

$$\text{imediato}(v) = \{w \mid v < w, \nexists\, x, v < x, x < w\}$$

Um diagrama de lattice é um grafo em que os elementos do lattice são nós e existe uma aresta dirigida de w para v se e somente se w é um *imediato(v)*.

Um grafo G (V, E) é um conjunto finito não vazio **V** e um subconjunto de pares não ordenados **E** de elementos distintos de **V**. Os elementos de ***V*** são designados por **vértices** e os elementos de ***E*** por **arestas** ou **arcos** [Boa96, Fur73].

Um grafo ***G*** é representado por um conjunto de suas arestas ***E(G)*** e um conjunto dos seus vértices ***V(G***).

Cada aresta ***e*** pertencente a E será denotada pelo par de vértices e = (v,w) que a forma. Os vértices v e w são extremos das arestas e são denominados adjacentes.

Como o lattice possui vértices ordenados, cada aresta (v,w) possui uma única direção de w para v, onde w é divergente e v é convergente, este grafo é denominado de dígrafo ou grafo dirigido, denotado por D(V,E).

Um **grafo dirigido *D*** (ou **dígrafo**) é um par ***(V, A)*** em que ***V*** é um conjunto (finito não vazio) e ***A*** é um subconjunto de ***V***. Os elementos de ***V*** são chamados de **vértices** de ***D*** e os elementos de ***A*** são denominados **arcos** (ou **arestas orientadas**) de ***V*** [Boa96, Fur73].

O lattice de visões materializadas é um dígrafo onde os ***V(D)*** são representados pelas visões materializadas; e as ***A(D)*** são representadas pelas dependências destas visões.

Tomando por base a figura 3.1, a figura 3.2 apresenta o dígrafo correspondente.

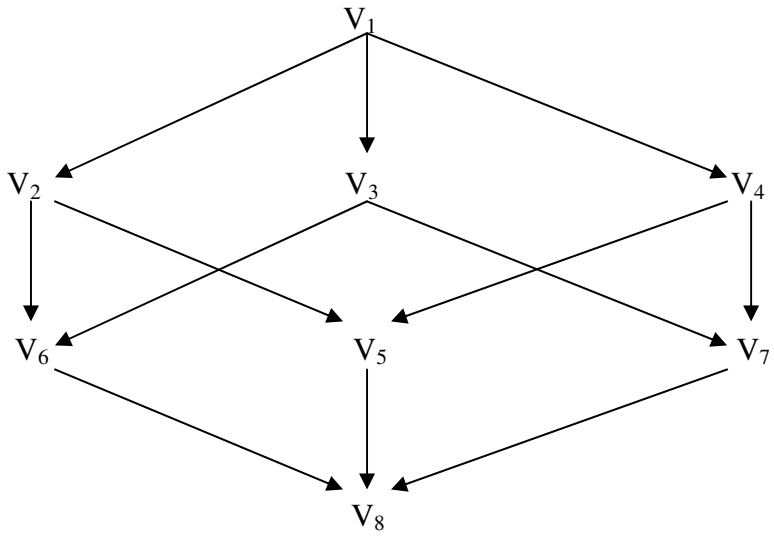


Figura 3.2: Dígrafo de um lattice de visões

3.1.5 Lattice de hierarquias de dimensão

Hierarquia de dimensão é um conjunto de dependências funcionais entre atributos de uma tabela de dimensão, em que os pares contíguos destes atributos possuem um relacionamento de 1 para n. Estas hierarquias são importantes para consultas OLAP nas operações drill-down e drill-up.

Drill-down é o processo de visualizar dados com maior nível de detalhe. Por exemplo, o usuário pode primeiramente visualizar o total de vendas por ano, depois o total de vendas por mês e finalmente por dia. O processo drill-up é o oposto do drill-down, ou seja, o usuário inicia verificando os dados de vendas por dia e finalmente por ano.

Normalmente, uma dimensão possui uma hierarquia linear, como por exemplo, a dimensão loja, da figura 3.3, em que várias lojas podem ser agrupadas em uma cidade e várias cidades podem ser agrupadas em uma região, porém uma única dimensão pode conter múltiplas hierarquias, como por exemplo, uma dimensão tempo, em que podem ser definidas duas hierarquias diferentes, uma para datas do calendário normal outra para um calendário fiscal com as seguintes relações de dependências: ano ≤ mês ≤ dia e ano_fiscal ≤ mês_fiscal ≤ dia.

A figura 3.3 apresenta exemplos de lattices de hierarquias de dimensões.

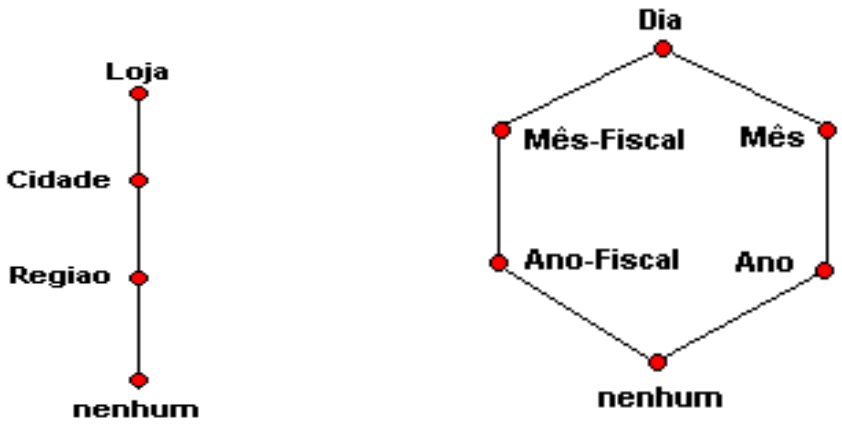


Figura 3.3: Lattice de hierarquias das dimensões loja e tempo

3.1.6 Lattice composto por múltiplas hierarquias de dimensões

As dimensões que compõem o cubo de dados possuem hierarquias que permitem diferentes agrupamentos. Estes diferentes agrupamentos formam um lattice composto por múltiplas hierarquias de dimensões, gerando um maior número de visões materializadas que podem ser criadas. Estas combinações são obtidas através do *produto direto* dos lattices das hierarquias de dimensões [HAR96, MQM97]. O *produto direto* é o resultado do produto cartesiano dos lattices das hierarquias de dimensões.

A figura 3.5 [MQM97] apresenta o resultado do produto cartesiano dos lattices de hierarquias das dimensões produto (d1), loja (d2) e dia (d3), apresentados na figura 3.4.

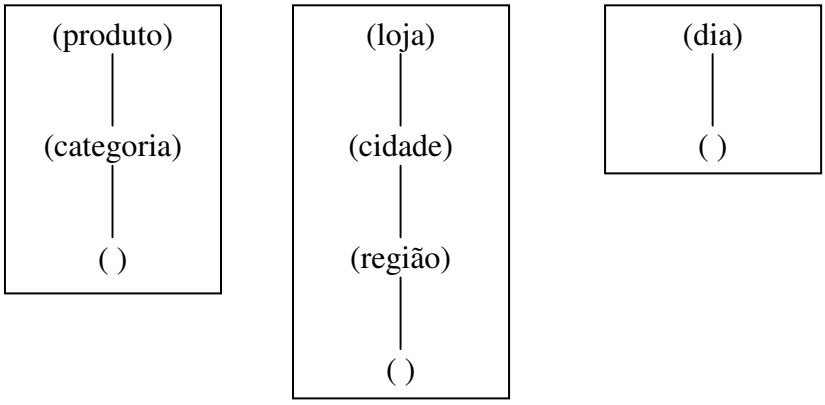


Figura 3.4: Lattice das hierarquias das dimensões produto, loja e data

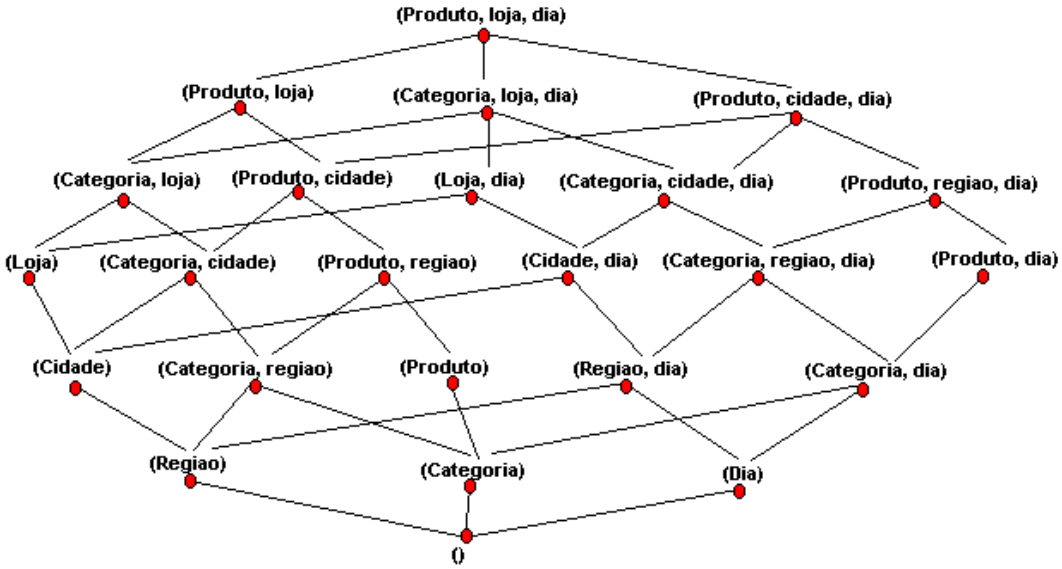


Figura 3.5: Lattice composto [MQM97]

3.1.7 Lattice com atributos de hierarquias de dimensão divididos

O lattice da hierarquia da dimensão loja, da figura 3.3, forma conjuntos completos com todas as lojas, todas as cidades e todas as regiões, entretanto conjuntos completos de um atributo podem ser divididos em subconjuntos de atributos de uma hierarquia de dimensão.

Por exemplo, pode-se desejar analisar separadamente algumas lojas dividindo-as em dois subconjuntos, um de lojas com vendas ao varejo e outro com vendas ao atacado.

Este modelo é denominado lattice com atributos de hierarquias de dimensão divididos [DJ98], pois divide o atributo loja da dimensão, em dois novos atributos: atacado e varejo.

A figura 3.6 apresenta um exemplo de divisão do atributo loja em dois atributos: atacado e varejo, feita através de uma seleção. A figura mostra, também, o produto direto da dimensão loja e tempo, gerando seis combinações diferentes.

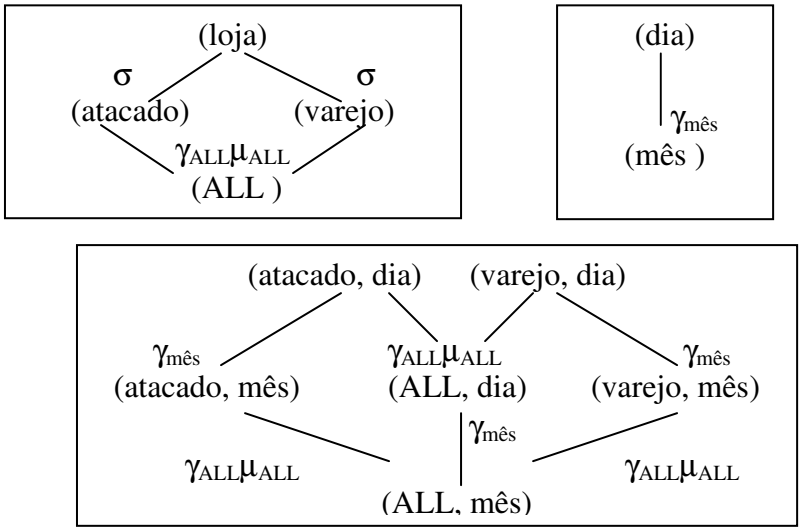


Figura 3.6: Lattice com atributos de hierarquias de dimensão divididos [DJ98]

A figura 3.6 mostra, também, que é possível obter dados de todas lojas por dia (ALL,dia), simplesmente unindo (μ) dados das lojas de atacado por dia (atacado, dia) e dados das lojas de varejo por dia (varejo, dia); e posteriormente agregando (γ) o resultado da união por dia.

Dados de todas as lojas por mês (ALL, mês) são obtidos, agregando (γ) os dados diários (ALL, dia) por mês ou unindo (μ) os dados das lojas de atacado por mês (atacado, mês) com os dados das lojas de varejo por mês (varejo, mês) e posteriormente agregando (γ) o resultado da união por mês.

O referido modelo introduziu duas novas relações de derivação: seleção e união. As relações de derivação são apresentadas na tabela 3.1 [DJ98].

Função	Operador
Agregação [HAR96]	γ
Seleção	σ
União	μ

Tabela 3.1: Relação de derivação [DJ98]

3.1.8 Modelo de um DWM

Definição 1 – DWM como um lattice de um cubo de dados

Um DW móvel é um lattice representando um cubo de dados, em que:

- Os vértices $v \in V(G)$ de um dígrafo G, também denominados **nós**, representam as diferentes combinações dos atributos de dimensão de uma tabela de fatos, que geram diferentes visões materializadas em uma plataforma móvel;
- O número de visões materializadas que podem ser geradas é igual 2^K, em que k significa o número de atributos de dimensão da tabela de fatos [MQM97];
- A visão que ocupa o nó raiz, também denominado vértice fonte, é chamada de tabela de fatos;
- Todas as outras visões são dependentes direta ou indiretamente da tabela de fatos e são denominadas tabelas sumarizadas.
- O nó raiz do lattice, ou tabela de fatos, é criado a partir de tabelas das fontes de dados;
- $V_j \leq V_i$, se e somente se, cada linha de V_j é um agregado de linhas de V_i, segundo diferentes subconjuntos dos atributos de dimensão da tabela V_i. Genericamente, uma visão pode ser criada usando a seguinte instrução SQL:
  - ✓ Create view V_j as

    Select non-aggregate-fieldname$_1$, ... non-aggregate-fieldname$_m$,

    aggregate-function(aggregate-fieldname$_n$),.....,

    aggregate-function(aggregate-fieldname$_z$)

    From V_i

    Group by non-aggregate-fieldname$_1$, ... non-aggregate-fieldname$_m$

- Existe uma relação de dependência entre nós (vértices) do lattice, permitindo a criação de uma visão a partir de uma ou mais visões, ou seja, $V_j \leq V_i$ implica que a visão V_j

pode ser criada a partir de uma visão V_i, conforme relação de dependência mostrada na figura 3.7.

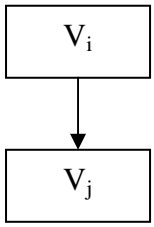


Figura 3.7: Dependência entre vértices de um lattice de um DWM

- As diferentes visões materializadas podem ser criadas através de SELECT-FROM-WHERE-GROUPBY, tendo idênticas funções de agregação. A cláusula FROM é substituída pela visão divergente V_i e a cláusula WHERE já foi utilizada para definir V_i, não sendo mais necessária. Os subconjuntos (2^K) dos atributos de dimensão da tabela de fatos definem as colunas das cláusulas SELECT e GROUPBY.
- Os atributos não agregados (non-aggregate-fieldname$_1$, ..., non-aggregate-fieldname$_m$) são subconjuntos dos atributos de dimensão da tabela V_i e chaves de uma agregação.

Definição 2 – DWM como um lattice composto

Um DWM pode ser, também, um lattice composto, através do produto direto dos lattices das hierarquias de dimensões, em que:

- Os vértices $\boldsymbol{v} \in V(G)$ de um dígrafo G, também denominados **nós**, representam as diferentes combinações do produto direto dos atributos que compõem as hierarquias das dimensões, gerando diferentes visões materializadas em uma plataforma móvel;
- A visão que ocupa o nó raiz é chamada de tabela de fatos; e as outras visões são denominadas tabelas sumarizadas;
- O nó raiz do lattice, ou tabela de fatos, é criado a partir de tabelas das fontes de dados;
- $V_j \leq V_i$, se e somente se, cada linha de V_j é um agregado de linhas de V_i segundo as hierarquias de tabelas de dimensões de V_i. Genericamente, uma visão pode ser criada usando a seguinte instrução SQL:
  - ✓ Create view V_j as

    Select non-aggregate-fieldname$_1$, ... non-aggregate-fieldname$_m$,

    aggregate-function(aggregate-fieldname$_n$),.....,

    aggregate-function(aggregate-fieldname$_z$)

From V_i, $DimensionTable_1$,..., $DimensionTable_p$
Where $JoinCondition_1$ and .... and $JoinCondition_p$
Group by $\underline{non\text{-}aggregate\text{-}fieldname_1}$, ... $\underline{non\text{-}aggregate\text{-}fieldname_m}$

- Em um lattice composto, algumas visões materializadas, precisam sofrer junções com uma ou mais tabelas de dimensão a fim de criar outras visões, ou seja, para criar a visão V_j é necessário utilizar a visão V_i com uma ou mais junções às tabelas de dimensão, conforme figura 3.8.

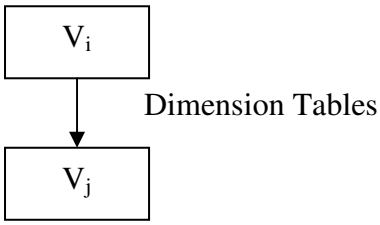


Figura 3.8: Dependência de um lattice com junções de tabelas de dimensão em um DWM

- As diferentes visões materializadas podem ser criadas através de SELECT-FROM-WHERE-GROUPBY, tendo idênticas funções de agregação. A cláusula FROM receberá, além da visão divergente (V_i), as tabelas de dimensões para gerar a visão convergente (V_j), a cláusula WHERE terá as condições de junção das tabelas de dimensões, enquanto os subconjuntos do produto direto irão compor as colunas da cláusula SELECT e GROUPBY.
- Os atributos não agregados (non-aggregate-$fieldname_1$, ..., non-aggregate-$fieldname_m$) são diferentes combinações de atributos das hierarquias de dimensões da tabela V_i e chaves da agregação.
- Se o lattice com atributos de hierarquias de dimensão é dividido (seção 3.1.7) então $\Sigma(V_j)_k \leq V_i$, onde cada linha de V_j é a união das linhas de cada $(V_i)_k$, seguido de uma agregação, em que k representa o número de visões que serão unidas. A visão V_j pode ser criada usando a seguinte instrução SQL:
  - ✓ Create view V_j as
    (Select $\underline{non\text{-}aggregate\text{-}fieldname_1}$, ... $\underline{non\text{-}aggregate\text{-}fieldname_m}$,
    aggregate-function(aggregate-$fieldname_n$),.....,
    aggregate-function(aggregate-$fieldname_z$)
    From (SELECT .. FROM V_{i1}) UNION ALL .... UNION ALL
    (SELECT .. FROM V_{ik}))
    Group by $\underline{non\text{-}aggregate\text{-}fieldname_1}$, ... $\underline{non\text{-}aggregate\text{-}fieldname_m}$

3.1.9 Modelo de um banco de dados no servidor proxy

Definição 1 – Banco de dados de um servidor proxy como um lattice de um cubo de dados

Um banco de dados no servidor proxy também pode ser visto como um lattice de um cubo de dados, em que:

- As alterações nas fontes de dados são capturadas pelo servidor proxy e inseridas em arquivos de log;
- As inserções ocorridas nas fontes de dados são registradas em logs de inserção (log_ins), as exclusões registradas em logs de exclusão (log_del) e as alterações enviam os dados antigos para o log de exclusão (log_del) e os dados novos para os log de inserção (log_ins);
- No servidor proxy, os dois logs (log_ins e log_del) são unidos em uma única visão denominada Δlog, usando o seguinte comando SQL:
  - ✓ Create View Δlog as
    (Select * from log_ins) Union all (Select * from log_del)
- Cada **nó**, do servidor proxy, denominado de tabela delta-sumário, representa um agrupamento de alterações ocorridas nas fontes de dados (Δlog), segundo diferentes critérios (dimensão);
- O nó raiz do lattice, ou tabela delta-sumário raiz, do servidor proxy é criado agrupando inserções e remoções das fontes (Δlog), segundo os critérios de menor granularidade em relação aos DWM existentes;
- $\Delta V_j \leq \Delta V_j$, se e somente se, cada linha de ΔV_j é um agregado de linhas de ΔV_i segundo diferentes subconjuntos dos atributos de dimensão da tabela ΔV_i. Genericamente, uma visão pode ser criada usando a seguinte instrução SQL:

  - ✓ Create view ΔV_j as
    Select $\underline{\text{non-aggregate-fieldname}_1}$, ... $\underline{\text{non-aggregate-fieldname}_m}$,
    aggregate-function(aggregate-fieldname$_n$),.....,
    aggregate-function(aggregate-fieldname$_z$)
    From $\underline{\Delta V_i}$
    Group by $\underline{\text{non-aggregate-fieldname}_1}$, ... $\underline{\text{non-aggregate-fieldname}_m}$

- Uma visão delta-sumário, diferente do nó raiz, pode ser obtida a partir de outras visões delta-sumário, ou seja, $\Delta V_j \leq \Delta V_i$ denotando que a visão ΔV_j pode ser criada a partir de uma visão ΔV_i, conforme relação de dependência mostrada na figura 3.9.

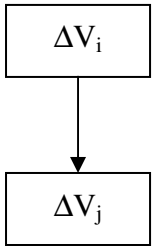


Figura 3.9: Dependência de um lattice no servidor proxy

- As diferentes visões delta-sumários podem ser criadas através de SELECT-FROM-WHERE-GROUPBY, tendo idênticas funções de agregação. A cláusula FROM é substituída pela visão divergente ΔV_i e a cláusula WHERE já foi utilizada para definir ΔV_i, não sendo mais necessária. Os subconjuntos (2^K) dos atributos de dimensão definem as colunas da cláusula SELECT e GROUPBY.
- Se a tabela ΔV_j for uma visão delta-sumário do nó raiz a cláusula FROM é modificada para Δlog.
- A visão ΔV_i é responsável pela atualização incremental da visão V_i de todos os DW Móvel que as possuam.

Definição 2 – Banco de dados de um proxy como um lattice composto

- Em um lattice composto, algumas visões delta-sumário, precisam sofrer junções com uma ou mais tabelas de dimensões a fim de criar outras visões delta-sumários, ou seja, para criarmos, a visão ΔV_j é necessário utilizar a visão ΔV_i com junções às tabelas de dimensão, conforme figura 3.10.

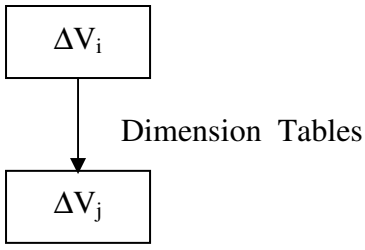


Figura 3.10: Dependência de um lattice com junções de tabelas de dimensão em um servidor proxy

- As diferentes visões delta-sumários podem ser criadas através de SELECT-FROM-WHERE-GROUPBY, tendo idênticas funções de agregação. A cláusula FROM receberá, além da visão divergente (ΔV_i), as tabelas de dimensões para criar a visão convergente (ΔVj), a cláusula WHERE terá as condições de junção das tabelas de dimensões, enquanto os subconjuntos do produto direto irão compor as colunas da cláusula SELECT e GROUPBY.
- Se o lattice dos atributos de hierarquias dimensão é dividido (seção 3.1.7) então $\Sigma(\Delta V_j)_k \leq \Delta V_i$, onde cada linha de ΔV_j é a união das linhas de cada $(\Delta V_i)_k$, seguido de uma agregação, em que k representa o número de visões que serão unidas. A tabela delta-sumário ΔV_j pode ser criada usando o seguinte comando SQL:

✓ Create view ΔV_j as
(Select non-aggregate-fieldname$_1$, ... non-aggregate-fieldname$_m$,
aggregate-function(aggregate-fieldname$_n$),.....,
aggregate-function(aggregate-fieldname$_z$)
From (SELECT .. FROM ΔVi_1) UNION ALL .... UNION ALL
(SELECT .. FROM ΔVi_k))
Group by non-aggregate-fieldname$_1$, ... non-aggregate-fieldname$_m$

3.2 Requisitos do Sistema

Para o desenvolvimento do MDWManager foram analisados os requisitos funcionais dos usuários finais, plataformas móveis, fontes de dados e servidor proxy, bem como os requisitos não funcionais.

3.2.1 Requisitos funcionais

As tabelas 3.2, 3.3, 3.4, 3.5 apresentam os requisitos funcionais do usuário final, plataformas móveis, fontes de dados e servidor proxy, respectivamente.

Usuário final	
Referência	**Funcionalidade**
UF1	O usuário deseja fazer consultas OLAP independente de conexão.
UF2	O tempo de resposta às consultas deve ser reduzido.
UF3	Não poderá haver interrupção para atualização do data warehouse móvel enquanto o usuário estiver fazendo uma consulta OLAP.
UF4	As atualizações devem ser feitas o mais rápido possível a fim de não atrasar o trabalho do usuário.
UF5	O usuário permite uma desatualização do data warehouse móvel de no máximo 24 horas.
UF6	O usuário terá um perfil de atualização do data warehouse em sua plataforma móvel.

Tabela 3.2: Requisitos funcionais do usuário final

Plataformas móveis	
Referência	**Funcionalidade**
PM1	A plataforma móvel deve ter um SGBD, um Data Warehouse e Ferramentas para consultas OLAP.
PM2	A plataforma móvel é responsável pela atualização incremental do DW móvel
PM3	A plataforma móvel poderá informar ao cliente a necessidade de reconstrução do DW (mensagem enviada fora da sessão OLAP)
PM4	DWs móveis solicitam ao servidor proxy que propague suas atualizações.
PM5	Os DWs móveis não acessam as fontes de dados.
PM6	A plataforma móvel mantém o número da versão de seu DWM.
PM7	A plataforma móvel solicita a atualização da versão de seu DWM no servidor proxy, após a atualização das visões materializadas.

Tabela 3.3: Requisitos funcionais das plataformas móveis

Fontes de dados	
Referência	**Funcionalidade**
FD1	As fontes de dados deverão ser independentes e autônomas.
FD2	As fontes devem informar a um servidor proxy as mudanças ocorridas nos dados.

Tabela 3.4: Requisitos funcionais das fontes de dados

Servidor proxy	
Referência	**Funcionalidade**
SP1	Um servidor proxy fixo deve capturar as alterações ocorridas nas fontes de dados através de logs de inserção e exclusão (deltas de modificação). O servidor proxy é a interface entre as fontes e os DWs móveis.
SP2	O servidor proxy manterá um controle de versão dos DWMs. Este controle é importante para determinar as atualizações propagadas para os DWMs, e as atualizações ainda não propagadas.
SP3	Os dados já enviados a todos os DWs móveis concernentes (uma espécie de "broadcasting") são removidas do servidor proxy.
SP4	O servidor proxy permitirá cadastramento dos DWs móveis.
SP5	Os logs têm tamanhos limitados.
SP6	O servidor proxy atende às solicitações de atualização dos DWMs (regime sob demanda).
SP7	Se os volumes do logs foram excedidos, seus dados devem ser propagados nas tabelas existentes no servidor proxy e posteriormente esvaziados.
SP8	O servidor proxy diminuirá o número de acessos às fontes de dados para calcular novos valores de mínimo e máximo através da manutenção de caches.
SP9	O servidor proxy deve propagar as alterações, em tabelas específicas, agrupando os dados com os mesmos critérios de agrupamentos existentes em visões materializadas nos DWM.
SP10	O servidor proxy não terá visões materializadas, mas somente as alterações das fontes.

Tabela 3.5: Requisitos funcionais do servidor proxy

3.2.2 Requisitos não funcionais

Embora o gerente dependa de alguns fatores tais como, largura de banda de rede, utilização de redes, configuração e utilização de vários recursos do servidor, configuração do dispositivo móvel, dentre outros, alguns requisitos não funcionais podem ser propostos. A tabela 3.6 mostra os requisitos não funcionais.

Requisitos não funcionais	
Referência	**Descrição**
RNF1	Devido ao alto custo de conexão dos dispositivos móveis com o servidor proxy, o tempo de conexão deve ser reduzido, diminuindo o volume de comunicação, através de agregações dos logs em tabelas com os mesmos critérios de agregação de visões existentes nos diversos DWMs. Estas pré-agregações representam um resumo das alterações ocorridas nas fontes.
RNF2	As funções de preparação e atualização serão desenvolvidas utilizando instruções SQL fortemente acoplados ao SGBD. Muitos DWMs estão instalados em dispositivos que possuem uma reduzida capacidade, portanto os processamentos, nestes dispositivos, devem ser reduzidos.
RNF3	O servidor proxy deve ser um equipamento com uma alta capacidade de processamento e armazenamento, portanto capaz de concentrar a maior parte dos processos responsáveis pela a atualização dos DWMs.
RNF4	A plataforma móvel necessita ter um Sistema de Gerência de Banco de Dados com ambiente completo de Data Warehouse, como por exemplo, ORACLE, IBM, INFORMIX etc.
RNF5	O servidor deve ser dotado de um Sistema de Gerência de Banco de Dados Relacional robusto e escalável.
RNF6	O MDWManager utilizará os mecanismos de segurança oferecidos pelos sistemas de gerência de banco de dados.

Tabela 3.6: Requisitos não funcionais

3.3 Arquitetura

A arquitetura de um DW tradicional não é capaz de lidar com as constantes desconexões das plataformas móveis, havendo uma necessidade de estendê-la, a fim de atender as restrições impostas pela mobilidade.

Para tratar as constantes desconexões das plataformas móveis (Mobile Hosts ou MHs), a arquitetura apresentada na figura 3.11, propõe um servidor proxy fixo, constantemente conectado às fontes de dados (S_1, S_2 e S_3).

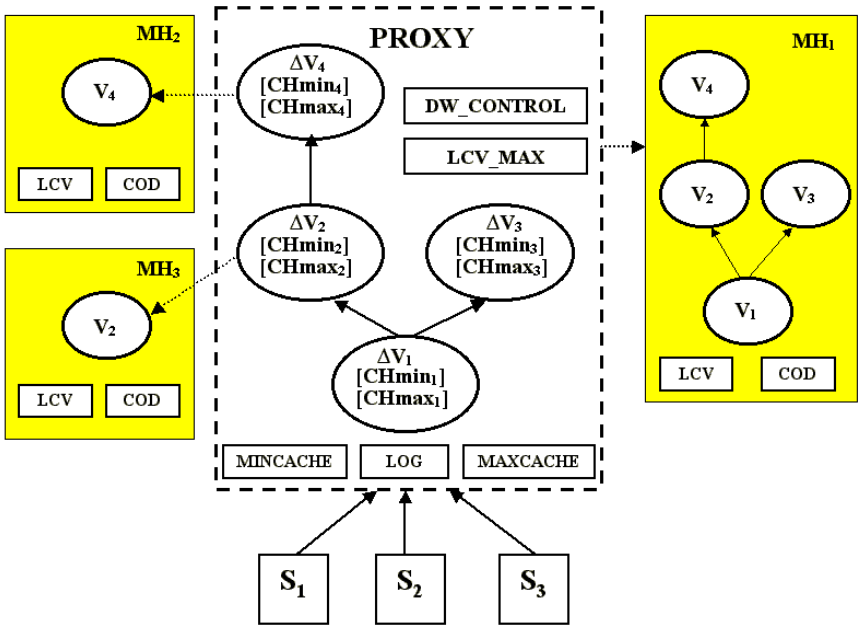


Figura 3.11: Arquitetura do Data Warehouse Móvel

O servidor proxy receberá, através de arquivos de logs (LOG), as mudanças ocorridas nas fontes de dados; e executará um pré-processamento periódico, agregando os dados destes logs em tabelas com os mesmos critérios de agregação das visões materializadas, residentes nos DWMs, atendendo ao requisito funcional SP9. Essas tabelas recebem o nome de *tabelas delta-sumários* (△V) [MQM97] e *cache-sumários* (CHmin e CHmax), representando o resumo dos dados que serão enviados para os DWMs.

O MDWManager elimina os acessos diretos entre os DWMs e as fontes de dados para atualizar visões materializadas que dependam diretamente das fontes, através da manutenção de caches no servidor proxy (Mincache, Maxcache, CHmin e CHmax), atendendo ao requisito PM5. Os caches são responsáveis pela manutenção de visões materializadas com funções de agregação de mínimo e máximo.

O servidor proxy enviará dados das tabelas delta-sumários e cache-sumários para as plataformas móveis, apenas no momento de uma solicitação de atualização de um DWM, atendendo aos requisitos funcionais SP1 e SP6.

O proxy é um equipamento com alta capacidade de processamento e armazenamento, atendendo o requisito RNF3.

A carga inicial de um DWM pode ser visto como um caso particular de atualização em que os dados podem ser obtidos através de duas estratégias:

- Carga inicial a partir do proxy: um servidor proxy pode conter um grande número de tabelas delta-sumários e cache-sumários, bem como um Data Warehouse fixo, dando ao proxy a responsabilidade pela carga inicial de DWMs.
- Carga inicial a partir de uma plataforma fixa: A arquitetura apresentada na figura 3.11 mostra somente a presença de plataformas móveis, entretanto é possível a utilização da arquitetura na atualização de DWs fixos. Estes DWs fixos podem ser utilizados para a carga inicial de DWMs.

Retomando o exemplo de DWMs da seção 1.2.1, vamos considerar a inclusão de uma nova plataforma móvel com DWM inicialmente vazio. A carga inicial deste DWM pode ser feita a partir do proxy ou de uma plataforma fixa dependendo da estratégia adotada.

A carga inicial das visões materializadas de um DWM será feita sob demanda da plataforma móvel através da obtenção dos dados completos existentes no proxy ou em plataformas fixas.

Outra característica importante do MDWManager é a possibilidade de gerenciar vários DWMs utilizando um controle de versão denominado LCV (*last change version*), atendendo ao requisito SP2.

O MDWManager permite a gerência de DWMs armazenados como hierarquias de visões materializadas (lattice). Na hierarquia de visões, a raiz é descrita como uma tabela de fatos e as outras visões, denominadas tabelas sumarizadas, são diretamente ou transitivamente derivadas da visão raiz ou tabela de fatos.

As visões residentes nas plataformas móveis, juntas, podem formar uma ou mais hierarquias (lattice) de tabelas sumarizadas, em que cada hierarquia é denominada *hierarquia global*.

Cada hierarquia global de tabelas sumarizadas terá, também, uma hierarquia equivalente no servidor proxy com tabelas delta-sumários.

Na figura 3.11, as visões materializadas V1, V2, V3 e V4 formam uma hierarquia global dos DWMs, enquanto as tabelas delta-sumários $\triangle$V1, $\triangle$V2, $\triangle$V3 e $\triangle$V4 representam a hierarquia equivalente no servidor proxy.

As tabelas de dimensão estão armazenadas nos DWMs e no servidor proxy.

A seguir, serão detalhados os elementos que compõem a arquitetura, nas plataformas móveis e no servidor proxy.

3.3.1 A plataforma móvel (Mobile Host)

As plataformas móveis, Mobile Hosts ou MHs são constituídas de tabelas sumarizadas ou visões materializadas (V), número de identificação do DWM (COD) e a versão atual do DWM (LCV).

- **Tabelas sumarizadas (V)**

As tabelas sumarizadas são tabelas que agregam dados de uma ou mais fontes, estando localizadas somente nas plataformas móveis. Estas tabelas são mantidas com dados provenientes do servidor proxy, não havendo necessidade de acesso às fontes, atendendo aos requisitos PM5 e SP10.

Como a proposta utiliza a abordagem incremental, algumas funções de agregação necessitam ser acrescentadas às funções existentes nas tabelas sumarizadas, outras substituídas, a fim de torná-las *self-maintainable* [MQM97] com relação à exclusão ou ajudá-las no processo de manutenção.

A tabela 3.7 apresenta algumas alterações que devem ser efetuadas nas tabelas sumarizadas, em que a coluna *tipo* contém as mudanças que devem ser feitas, podendo receber os valores *A* ou S, para acréscimos ou substituições, respectivamente.

Função de agregação existente	tipo	Modificações nas tabelas sumarizadas
SUM(expr)	A	COUNT(*)
COUNT(expr)	A	COUNT(*)
MIN(expr)	A	COUNT(*)
MAX(expr)	A	COUNT(*)
VARIANCE(expr)	S	SUM(expr), SUM(expr * expr) e COUNT(*)
STDDEV(expr)	S	SUM(expr), SUM(expr * expr) e COUNT(*)

Tabela 3.7: Alterações em tabelas sumarizadas

- **Número de identificação do DWM (COD)**

A tabela *COD* possui o número de identificação de um DW em uma plataforma móvel. Esta identificação é importante na atualização da tabela DW_CONTROL no servidor proxy após a atualização de um DWM. A tabela DW_CONTROL será detalhada adiante.

- **Versão do data warehouse móvel (LCV)**

A tabela *LCV* contém a versão atual de um DWM e será usada para determinar as tuplas das tabelas *delta-sumários* que deverão ser solicitadas ao proxy para atualizar suas visões materializadas, atendendo ao requisito SP6. Esta versão pode ser um timestamp ou uma seqüência de um SGBD.

3.3.2 Exemplo de motivação

Considere um cenário semelhante ao apresentado na figura 3.11, em que as fontes recebem dados de vendas de diferentes lojas. Os dados de vendas são armazenados em uma tabela *Vendas* com a seguinte estrutura:

Vendas(venda_id, produto, loja, data, qtd, preço),

em que os atributos são: o número de identificação da venda; o produto vendido; a loja e a data da venda; a quantidade vendida; e o preço do produto vendido.

As tabelas de dimensão: produtos, lojas e tempos; são armazenadas em todos os dispositivos móveis e no servidor proxy. As dimensões possuem as seguintes estruturas:

- Produtos (produto, nome_produto, categoria, nome_categoria);
- Lojas (loja, nome_loja, cidade, nome_cidade, região, nome_região);
- Tempos (dia, data, mês).

A união de todas as visões existentes nos três dispositivos móveis (MH1, MH2 e MH3) forma uma hierarquia global semelhante à apresentada em MH1 da figura 3.11. Nas visões de MH1, a tabela de fatos V1 é criada a partir das tabelas *Vendas* das fontes, enquanto as outras visões são criadas a partir de outras tabelas sumarizadas.

A visão V1, denominada PLD_VENDAS, representa a estatística de vendas por produto, loja e dia. Essa visão é criada com a seguinte instrução SQL:

- ✓
```
CREATE VIEW PLD_VENDAS(produto, loja, dia, qtd, receita, contador,
        min_qtd, max_qtd) AS
    SELECT produto, loja, dia, sum(qtd), sum(qtd * preço), count(*),
        min(qtd), max(qtd)
    FROM vendas, tempos
    WHERE vendas.data = tempos.data
    GROUP BY produto, loja, dia
```

A visão V2, denominada PLM_VENDAS, representa a estatística de vendas por produto, loja e mês. Essa visão é criada, a partir da visão PLD_VENDAS (V1), através da seguinte instrução SQL:

- ✓ CREATE VIEW PLM_VENDAS(produto, loja, mês, qtd, receita, contador, min_qtd, max_qtd) AS
  SELECT produto, loja, mês, sum(qtd), sum(receita), sum(contador), min(min_qtd), max(max_qtd)
  FROM PLD_vendas, tempos
  WHERE PLD_vendas.dia = tempos.dia
  GROUP BY produto, loja, mês

A visão V3, denominada PCD_VENDAS, representa a estatística de vendas agrupadas por produto, cidade e dia. Essa visão é criada, a partir da visão PLD_VENDAS (V1), através da seguinte instrução SQL:

- ✓ CREATE VIEW PCD_VENDAS(produto, cidade, dia, qtd, receita, contador, min_qtd, max_qtd) AS
  SELECT produto, cidade, dia, sum(qtd), sum(receita), sum(contador), min(min_qtd), max(max_qtd)
  FROM PLD_Vendas, lojas
  WHERE PLD_vendas.loja = lojas.loja
  GROUP BY produto, cidade, dia

A visão V4, denominada CatR_VENDAS, representa a estatística de vendas por categoria e região. Essa visão é criada a partir da visão PLM_VENDAS (V2) através da seguinte instrução SQL:

- ✓ CREATE VIEW CatR_VENDAS(categoria, região, qtd, receita, contador, min_qtd, max_qtd) AS
  SELECT categoria, região, sum(qtd), sum(receita),sum(contador), min(min_qtd), max(max_qtd)
  FROM PLM_vendas, lojas, produtos
  WHERE PLM_vendas.loja = lojas.loja
  and PLM_vendas.produto = Produtos.produto
  GROUP BY categoria, região

As tabelas COD de MH1, MH2 e MH3 recebem os valores 1, 2 e 3 respectivamente, representando o número de identificação de cada DWM.

As tabelas LCV, de cada dispositivo, recebem inicialmente o valor 0 (zero), pois não foram atualizadas utilizando o servidor proxy.

3.3.3 O servidor proxy

Uma das funções mais importantes do servidor proxy, além de extrair as mudanças ocorridas nas fontes, através dos arquivos de logs, é propagar estas mudanças em tabelas internas que serão utilizadas no processo de atualização dos DWMs, atendendo ao requisito SP7. Esse processo, denominado *preparação*, visa:

- Eliminar o acesso direto dos DWMs e as fontes de dados na atualização;
- Diminuir o volume de comunicação entre a plataforma móvel e o servidor proxy, pois resume a quantidade de alterações enviadas às plataformas móveis;
- Aumentar a eficiência da atualização do DWM, pois a preparação executada no servidor proxy diminui a carga de trabalho da plataforma móvel.

O servidor proxy é constituído de arquivos de logs (inserção e exclusão), caches centralizados de mínimo e máximo (MINCACHE e MAXCACHE), tabelas de controle dos DWMs (DW_CONTROL), número da versão atual dos dados extraídos das fontes (LCV_MAX), tabelas delta-sumários ($\triangle$V1, $\triangle$V2, $\triangle$V3 e $\triangle$V4) e tabelas cache-sumários de mínimo e máximo ($CHmin_n$ e $CHmax_n$). A seguir, são detalhados os componentes que formam o servidor proxy.

- **Arquivos de log**

Na arquitetura do MDWManager, o servidor proxy recebe as mudanças que ocorrem nas fontes através de arquivos de logs.

Quando ocorre uma inserção, exclusão ou alteração nas fontes, um trigger é disparado a fim de inserir valores em determinados arquivos de log no servidor proxy.

Dados inseridos provocam inserções em logs de inserção (log_ins), dados excluídos provocam inserções em logs de exclusão (log_del) e dados alterados provocam inserções dos valores novos em logs de inserção e dos valores antigos em logs de exclusão.

No MDWManager, os logs possuem tamanhos limitados, ou seja, quando um determinado volume dos logs for atingido, ocorre uma nova preparação de dados no proxy, atendendo ao requisito SP5.

A estrutura do log é apresentada na figura 3.12.

Figura 3.12: Estrutura da tabela de log

O campo *chave* é semelhante à chave da tabela de fatos de uma hierarquia, o *valor* contém o valor que foi inserido ou excluído nas fontes. Se o trigger for de inserção o *valor* é positivo e se for de exclusão o *valor* é negativo.

O *contador* é necessário, quando existe um COUNT como operador de agregação na tabela de fatos ou em qualquer outra tabela sumarizada da hierarquia; e é usado para indicar o número de inserções e exclusões feitas. Para inserções, o contador recebe o valor +1 e para exclusões o contador recebe o valor –1. O *lcv* registra o timestamp (ou seqüência) da alteração da fonte.

A tabela 3.8 mostra como os logs são construídos a partir das funções existentes na tabela de fatos.

Funções na tabela de fatos	Log_ins	Log_del
COUNT(*)	+1	-1
COUNT(expr)	se expr é nulo 0 senão +1	se expr é nulo 0 senão -1
SUM(expr)	+expr	-expr
MIN(expr)	+expr	+expr
MAX(expr)	+expr	+expr

Tabela 3.8: Definição da construção de logs

No exemplo de motivação da seção 3.3.2, a tabela de fatos PLD_VENDAS, da hierarquia global, possui a chave produto, loja, dia; além das colunas sum(qtd), sum(qtd*preço), count(*), min(qtd) e max(qtd).

A tabela 3.9 mostra como ocorre a construção do log_ins e log_del do exemplo de motivação, a partir das funções existentes na tabela de fatos PLD_VENDAS.

Funções na tabela de fatos	log_ins	log_del
Chave	Produto, loja e dia	Produto, loja e dia
SUM(qtd)	+ qtd	- qtd
SUM(qtd * preço)	+ (qtd * preço)	- (qtd * preço)
COUNT(*)	+ 1	- 1
MIN(qtd)/MAX(qtd)	+ qtd	+ qtd

Tabela 3.9: Definição dos logs da tabela PLD_VENDAS

Portanto, os log_ins e log_del podem ser construídos como a seguir.

Log_ins(produto, loja, dia, +Novo qtd, +Novo (qtd*preço), +1, +Novo qtd, lcv)

Log_del(produto, loja, dia, -Velho qtd, -Velho (qtd*preço), -1, +Velho qtd, lcv)

Múltiplas atualizações de logs são possíveis, onde cada grupo de log (inserção e alteração) é responsável pela atualização de uma determinada hierarquia global, ou seja, se existir uma única hierarquia global, somente será necessário um grupo de log (log_ins e log_del), porém se existirem várias hierarquias globais deve haver arquivos de logs para cada tabela de fatos de uma hierarquia global individual.

- **Tabela Cache Centralizado**

Os caches centralizados de mínimo (MINCACHE) e máximo (MAXCACHE) contêm, respectivamente, os menores e maiores valores de um campo particular de uma determinada visão materializada.

Enquanto os caches, em Chan *et al* [CLS00], estão localizados nos diversos DWs, aumentando o volume de comunicação com o proxy e a carga de trabalho para atualizá-los, os caches centralizados do MDWManager estão localizados apenas no servidor proxy, não havendo nenhum processo de atualização de cache realizado nas plataformas móveis.

Os caches centralizados são atualizados por meio dos logs, atendendo ao requisito SP7 e têm como objetivo diminuir os acessos entre o servidor proxy e as fontes de dados para calcular novos valores de mínimos e máximos de visões materializadas, melhorando a performance da preparação de dados, atendendo ao requisito SP8.

Por exemplo, quando valores mínimos ou máximos de um determinado agrupamento de uma tabela sumarizada são excluídos, o novo valor da função deve ser calculado pelo servidor proxy usando os caches existentes, somente acessando às fontes quando os dados, de

uma dada chave, não existem nos caches centralizados. A estrutura do cache centralizado é apresentada na figura 3.13.

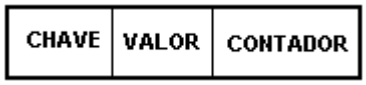


Figura 3.13: Estrutura do cache centralizado

O campo *chave* é a chave da visão materializada de menor granularidade da hierarquia global que possua funções de agregação de mínimo ou máximo; *valor* contém um valor mínimo ou máximo de um campo particular agregado; e o *contador* contém o número de ocorrências do *valor* mínimo ou máximo nas fontes de dados. A chave desta tabela é formada pelos campos *chave* e *valor*.

No exemplo de motivação, observa-se que a tabela de menor granularidade que possui funções de mínimo e máximo sobre o campo *qtd*, é tabela de fatos PLD_VENDAS. Neste caso, deverão existir duas tabelas de caches centralizados, uma para mínimo e outra para máximo, com as seguintes estruturas:

PLD_VENDAS_MINCACHE(produto, loja, dia, qtd, contador) e;

PLD_VENDAS_MAXCACHE(produto, loja, dia, qtd, contador).

As atualizações dos caches centralizados são feitas a partir de uma agregação dos dados dos logs com o mesmo critério de agregação dos caches centralizados (*chave e valor*). Caso a tupla, obtida com a agregação do logs, já exista no cache, o *contador* do cache centralizado é incrementado ao contador obtido na agregação do logs.

Se nenhuma tupla com a mesma chave está no cache centralizado, ela é inserida na tabela de cache somente quando o valor agregado do log, é menor que o mais alto valor do cache centralizado mínimo com a mesma chave ou maior do que o mais baixo valor do cache centralizado máximo com a mesma chave.

Cada hierarquia é composta de um cache centralizado mínimo e máximo, os quais são responsáveis pela atualização das tabelas cache-sumários ($CHmin_n$ e $CHmax_n$) existentes no servidor proxy.

- **Tabela cache-sumário ($CHmin_n$ e $CHmax_n$)**

As tabelas sumarizadas que possuem funções de agregação de mínimo e máximo terão tabelas cache-sumários de mínimo e máximo, respectivamente, no servidor proxy, com os mesmos critérios de agregação.

Essas tabelas contêm um resumo dos caches centralizados; e possuem valores únicos de mínimos ou máximos de determinados campos das tabelas sumarizadas que possuem estas funções de agregação.

As tabelas cache-sumários e delta-sumários são responsáveis pela atualização das tabelas sumarizadas dos DWMs.

A estrutura da tabela cache-sumário é apresentada na figura 3.14.

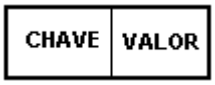


Figura 3.14: Estrutura da tabela cache-sumário

O campo *chave* contém a chave da tabela sumarizada que ela mantém e o *valor* contém o valor mínimo ou máximo de um campo, de uma determinada chave da tabela sumarizada.

As tabelas cache-sumários podem ser atualizadas a partir dos caches centralizados ou a partir de outras tabelas cache-sumários, conforme uma hierarquia.

Para atualizar um determinado cache-sumário, tuplas do cache centralizado ou de outra tabela cache-sumário ancestral são agrupadas com o mesmo critério de uma tabela sumarizada.

A partir do agrupamento de tuplas do cache centralizado mínimo ou de tuplas da tabela cache-sumário-mínimo ancestral, obtém-se o menor valor de um campo; e a partir do agrupamento de tuplas do cache centralizado máximo ou de tuplas da tabela cache-sumário-máximo ancestral, obtém-se o maior valor de um campo, formando as tabelas cache-sumários de mínimo e máximo respectivamente, responsáveis pela atualização dos valores das funções de agregação de mínimo e máximo de uma determinada tabela sumarizada.

A figura 3.15 mostra a forma de se obter tuplas de tabelas cache-sumários.

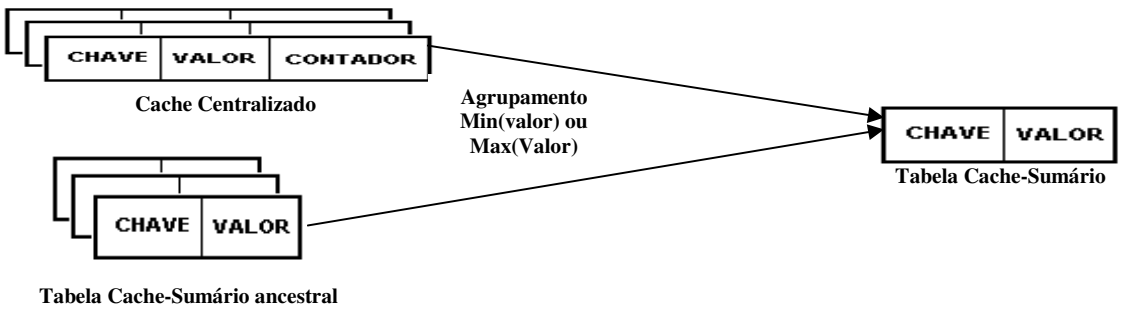


Figura 3.15: Criação da tabela cache-sumário

No exemplo de motivação, todas as tabelas sumarizadas possuem funções de mínimo e máximo para o atributo *qtd*, portanto as tabelas cache-sumários terão as seguintes estruturas:

PLD_VENDAS_MIN(produto, loja, dia, qtd);
PLD_VENDAS_MAX(produto, loja, dia, qtd);
PCD_VENDAS_MIN(produto, cidade, dia, qtd);
PCD_VENDAS_MAX(produto, cidade, dia, qtd);
PLM_VENDAS_MIN(produto, loja, mês, qtd);
PLM_VENDAS_MAX(produto, loja, mês, qtd);
CatR_VENDAS_MIN(categoria, região, qtd);
CatR_VENDAS_MAX(categoria, região, qtd).

Além dos cache-sumários ancestrais ou dos caches centralizados, podem ser necessárias as presenças de tabelas de dimensão na atualização de cache-sumários descendentes.

A figura 3.16 mostra a hierarquia de tabelas cache-sumários e suas junções às tabelas de dimensão, baseado no exemplo de motivação. As arestas rotuladas representam as dimensões necessárias nas junções.

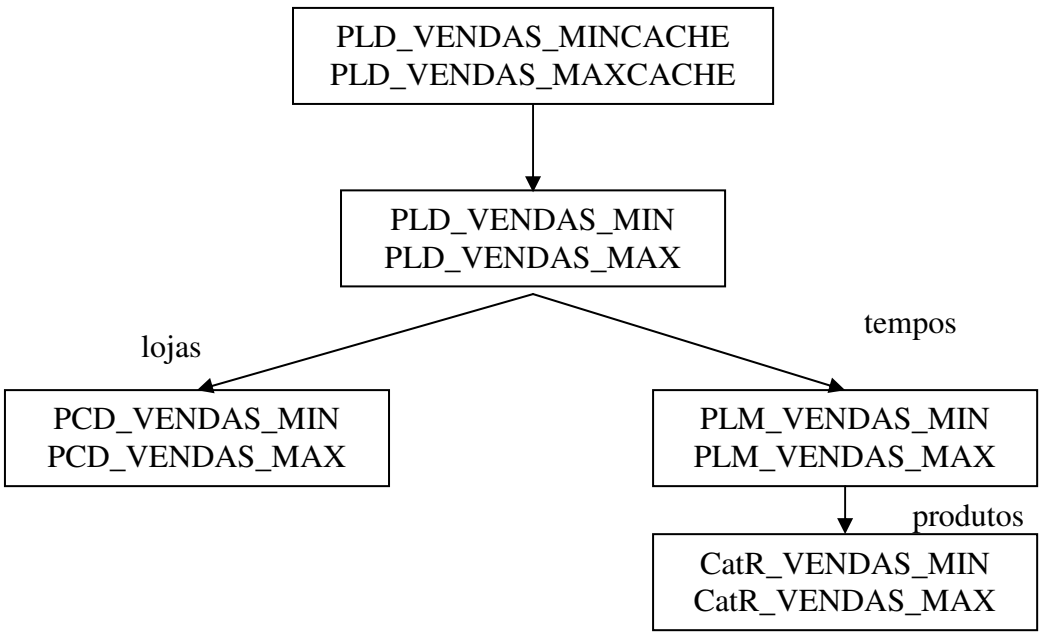


Figura 3.16: Hierarquia das tabelas cache-sumários

- **Tabelas Delta-Sumário (△V)**

As tabelas delta-sumários usam os mesmos critérios de agregação das tabelas sumarizadas do DWM; e são atualizadas a partir dos arquivos de logs (log_ins e log_del), atendendo ao requisito SP7, ou a partir de outras tabelas delta-sumários, de acordo com a hierarquia [MQM97].

Essas tabelas representam o resumo das mudanças das fontes que devem ser enviadas para as plataformas móveis, diminuindo o volume de comunicação entre a plataforma móvel e o servidor proxy, aumentando a velocidade de atualização e o tempo de disponibilidade dos DWMs.

As tabelas delta-sumários são utilizadas para atualizar as visões materializadas nos dispositivos móveis. As tabelas cache-sumários, também, serão utilizadas na atualização, caso as visões possuam funções de agregação de mínimo ou máximo.

A estrutura da tabela delta-sumário é semelhante à tabela sumarizada que ela mantém, porém, não contém valores mínimos ou máximos, pois estes valores estão nos cache-sumários. Uma outra alteração é a inserção da coluna *lcv* que armazena o maior lcv de um agrupamento de logs.

No exemplo de motivação, as tabelas delta-sumários terão as seguintes estruturas:

SD_PLD_VENDAS(produto, loja, dia, qtd, receita, contador,lcv);
SD_PCD_VENDAS(produto,cidade,dia, qtd, receita,contador,lcv);
SD_PLM_VENDAS(produto, loja, mês, qtd, receita, contador,lcv);
SD_CatR_VENDAS(categoria, região, qtd, receita, contador,lcv).

Os dados dos logs agrupados são inseridos nas tabelas delta-sumários, não havendo alterações em tuplas, garantindo a manutenção de vários DWMs, independente dos seus níveis de atualização.

- **Tabela DW_CONTROL**

A tabela DW_CONTROL contém o número de identificação dos dispositivos móveis e a versão atual de cada DW. Esta tabela é usada para determinar a menor versão dos DWMs;

e a partir dela, descartar tuplas das tabelas delta-sumários já utilizadas por todos os DWMs, atendendo ao requisito SP3.

Uma outra vantagem da referida tabela é determinar a necessidade de reconstrução do DWM, caso a versão atual do DWM seja menor do que a menor versão existente no proxy, obtida a partir do menor lcv da tabela DW_CONTROL, atendendo ao requisito PM3.

A tabela DW_CONTROL é criada pelo administrador do banco de dados através do cadastramento dos DWMs existentes e atualizadas quando os DWMs são atualizados. A estrutura da tabela DW_CONTROL é apresentada na figura 3.17.

Figura 3.17: Estrutura da tabela DW_CONTROL

O campo *código* contém o número de identificação de um DWM e o *lcv* contém a versão atual de cada DWM.

- **Tabela LCV_MAX**

A tabela LCV_MAX contém o maior lcv dos logs utilizados na preparação. A tabela LCV_MAX é utilizada pelo dispositivo móvel, durante a atualização do DWM, para verificar se a versão local (tabela lcv da plataforma móvel) está atualizada em relação aos dados contidos no servidor proxy (LCV_MAX), bem como, determinar os dados existentes no servidor proxy que serão solicitados, atendendo aos requisitos SP2 e PM4.

Caso os valores do lcv do DWM e do LCV_MAX do proxy sejam diferentes, o DWM solicitará tuplas às tabelas delta-sumário com colunas lcv maior do que o lcv local e menor ou igual ao LCV_MAX.

3.4 Algoritmos de *preparação* e *atualização*

Uma das características importantes do MDWManager é executar, no servidor proxy, uma grande parte do processo de manutenção dos DWMs. Esse processo é denominado de *preparação*.

Posteriormente, os dados preparados pelo servidor proxy são enviados para a plataforma móvel, mediante solicitação, para efetuar a manutenção do DWM. O processo que executará a manutenção é denominado de *atualização*.

Todos os algoritmos utilizados são fortemente acoplados ao SGBD, a fim de usufruir de seus mecanismos de otimização.

3.4.1 Algoritmo *preparação*

O algoritmo *preparação*, mostrado na figura 3.18, é responsável pela propagação das alterações das fontes, em tabelas internas do proxy, que serão utilizadas na atualização dos DWMs.

```
Preparação(v_size)
 /* Criação da tabela delta_log e um índice para a tabela*/
 Create Table delta_log as (Select * from log_ins) Union all
          (Select * from log_del);
 Create index delta_log_idx on (non_aggregate_fieldname1, ...,
              non_aggregate_fieldnamep, expression);
 /* Atualização e Reconstrução de cache centralizado */
atualiza_cache(v_size);
reconstrução_cache(v_size);

 /* Determinar a ordem para percorrer um grafo */
ordenação(1);

 /*  Determinar a versão mínima existente no servidor proxy */
 v_lcv_min ← Select min(lcv) from DW_CONTROL;
 /* Atualiza as tabelas delta-sumário e cache_sumário /*
propaga_nos(v_lcv_min);

 /*Obtém maior lcv da tabela delta_log e atualiza LCV_MAX*/
 v_lcv_max ← select max(lcv) from delta_log;
 Update lcv_max set lcv = v_lcv_max;
 /* Eliminação de tuplas de logs já utilizadas */
 Delete from log_ins where lcv <= v_lcv_max;
 Delete from log_del where lcv <= v_lcv_max;
End;
```

Figura 3.18: Algoritmo *preparação*

A preparação é iniciada com a criação da tabela delta_log através da união dos logs de inserção com os logs de exclusão e com a criação de um índice para a tabela delta_log, que tem como objetivo melhorar a performance da atualização dos caches centralizados.

Em seguida é executado o procedimento de atualização do cache centralizado (*atualiza_cache*) em função de dados inseridos, excluídos e alterados nas fontes de dados. O

parâmetro *v_size* denota o tamanho do cache, conforme Chan *et al* [CLS00], em que a soma dos contadores de cada chave do cache centralizado é menor ou igual ao *v_size*, ou seja, $\forall c$, $\pi_{chave=c}$ (sum(contador)) <= v_size.

As exclusões ocorridas nas fontes podem eliminar todos os valores de uma determinada chave do cache centralizado, havendo a necessidade de uma reconstrução, utilizando as fontes de dados, executado pelo procedimento *reconstrução_cache*.

O procedimento *ordenação* tem como objetivo definir a ordem de atualização das tabelas delta-sumários e cache-sumários baseado em um algoritmo de varredura de grafos direcionados (dígrafo).

O procedimento *propaga_nos* é responsável pela inserção de dados nas tabelas delta-sumários e atualização das tabelas cache-sumários. Um outro procedimento executado é a eliminação de tuplas, das tabelas delta-sumários, utilizadas por todos os DWMs (v_lcv_min).

Finalmente, a preparação obtém a maior versão da tabela delta_log (*v_lcv_max*), a fim de registrar na tabela *LCV_MAX,* a maior versão *preparada* no servidor proxy.

A variável *v_lcv_max* é utilizada, também, para descartar tuplas dos logs utilizadas na *preparação*.

- **Algoritmo *atualiza_cache***

Na arquitetura apresentada na seção 3.3, o cache centralizado é formado pela chave da tabela sumarizada de menor granularidade da hierarquia global que possua funções de agregação de mínimo ou máximo, valores candidatos a mínimo ou máximo e um contador que contém o número de tuplas existentes nas fontes de dados para um determinado valor candidato.

No algoritmo *atualiza_cache*, a chave da tabela sumarizada é apresentada como $non\_aggregate\_fieldname_{1,}$ ....., $non\_aggregate\_fieldname_{p}$, o valor candidato como *expression e* o contador como *count*.

O algoritmo atualiza o cache centralizado de mínimo e máximo através dos seguintes passos:

1. Criação de tabelas temporárias de mínimo (min_temp) e máximo (max_temp), agrupando os dados da tabela delta_log com os mesmos critérios de agregação dos caches centralizados mínimo e máximo, respectivamente. As quantidades de valores candidatos são determinadas pela variável *v_size,* conforme [CLS00];

2. Criação de uma tabela min_del, utilizando a tabela temporária criada (min_temp) e o cache centralizado de mínimo, onde são armazenadas tuplas de min_temp que possuam valores candidatos menores que o maior valor existente no cache centralizado de mínimo para uma determinada chave ou que não exista neste;
3. Criação de uma tabela max_del, utilizando a tabela temporária criada (max_temp) e o cache centralizado de máximo, onde são armazenadas tuplas de max_temp que possuam valores candidatos maiores que o menor valor existente no cache centralizado de máximo para uma determinada chave ou que não exista neste;
4. Criação de uma tabela min_ins (max_ins). A criação é iniciada com a união das tuplas de min_del (max_del) com as tuplas do cache centralizado mínimo (máximo) que possuam valores chaves armazenados em min_del (max_del). Posteriormente, as tuplas são agrupadas com o mesmo critério de agregação do cache centralizado mínimo (máximo), não mantendo tuplas onde a soma dos contadores resulte no valor zero. Finalmente, os dados são gravados na tabela min_ins (max_ins);
5. Eliminação de tuplas do cache centralizado mínimo (máximo) que tenham valores chaves existentes na tabela min_del (max_del);
6. Inserção de tuplas de min_ins (max_ins) no cache centralizado mínimo (máximo).

A figura 3.19 mostra o algoritmo de atualização do cache centralizado de mínimo e máximo (*atualiza_cache*), em que *expression* representa o argumento da função mínimo ou máximo e *v_size* denota o tamanho do cache centralizado.

```
atualiza_cache(v_size)

-- Passo 1 (Criação das tabelas min_temp e max_temp)
CREATE TABLE MIN_TEMP [MAX_TEMP] AS
  (SELECT a.non_aggregate_fieldname1, ...,  a.non_aggregate_fieldnamep,
                a.expression, a.count
    FROM (select non_aggregate_fieldname1, ...,non_aggregate_fieldnamep,
       abs(expression) as expression,sum(count) as count
       FROM delta_log group by on_aggregate_fieldname1,..,
                non_aggregate_fieldnamep, abs(expression)) a
    WHERE ||  vsize ||   >= (SELECT sum(count) from delta_log d
       where a.non_aggregate_fieldname1=d.non_aggregate_fieldname1 and ...
       and a.non_aggregate_fieldnamep=d.non_aggregate_fieldnamep
        and abs(d.expression)<=abs(a.expression)
       [abs(d.expression)>=abs(a.expression])));
```

```
-- Passo 2 e 3 (Criação das tabelas min_del e max_del)
CREATE TABLE MIN_DEL [MAX_DEL] AS
(SELECT a.non_aggregate_fieldname1, ...,  a.non_aggregate_fieldnamep,
            a.expression, a.count
FROM (SELECT non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep,
      expression, count  FROM MIN_TEMP [MAX_TEMP]) a
LEFT OUTER JOIN
      (SELECT non_aggregate_fieldname1,...,non_aggregate_fieldnamep,
            max(expression) as MAX_V
            [min(expression) as MIN_V]
            FROM MINCACHE [MAXCACHE]
      WHERE (non_aggregate_fieldname1,....,non_aggregate_fieldnamep)
      IN (SELECT non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep
            FROM MIN_TEMP [MAX_TEMP])
      GROUP BY non_aggregate_fieldname1,...,non_aggregate_fieldnamep) b
      ON a.non_aggregate_fieldname1=b.non_aggregate_fieldname1 and ...
      and a.non_aggregate_fieldnamep=b.non_aggregate_fieldnamep
WHERE  b.MAX_V >= a.expression or b.MAX_V is null
      [b.MIN_V <= a.expression or b.MIN_V is null] )


-- Passo 4  (Criação das tabelas min_ins e max_ins)
CREATE TABLE MIN_INS [MAX_INS] AS
  (SELECT non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep,
    expression, sum(count) as count
      FROM ((SELECT * FROM MIN_DEL [MAX_DEL]) UNION ALL
            (SELECT * FROM MINCACHE  [MAXCACHE]
      WHERE (non_aggregate_fieldname1,....,
            non_aggregate_fieldnamep, expression)
      IN (SELECT non_aggregate_fieldname1,...,
              non_aggregate_fieldnamep, expression
              FROM MIN_DEL [MAX_DEL] )))
   GROUP BY non_aggregate_fieldname1,..,non_aggregate_fieldnamep, expression
   HAVING sum(count)>0)


-- Passo 5  (Exclusão de tuplas nos caches centralizados)
DELETE FROM MINCACHE [MAXCACHE]   WHERE
(non_aggregate_fieldname1,..,non_aggregate_fieldnamep,expression)
  IN (SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
            expression FROM MIN_DEL [MAX_DEL])


-- Passo 6  (Inserção de tuplas nos caches centralizados)
INSERT INTO MINCACHE [MAXCACHE] (SELECT * FROM MIN_INS[MAX_INS]);
```

Figura 3.19: Algoritmo *atualiza_cache*

A seção C.7, do apêndice C, mostra a atualização dos caches centralizados de mínimo (PLD_VENDAS_MINCACHE) e máximo (PLD_VENDAS_MAXCACHE) definidos na seção 3.3.3, do exemplo de motivação da seção 3.3.2.

- **Algoritmo *reconstrução_cache***

O *algoritmo reconstrução_cache* tem como objetivo reconstruir o cache centralizado de mínimo e máximo, quando tuplas chaves são excluídas do cache centralizado, através dos seguintes procedimentos:

1. Inserção de valores chaves presentes na tabela delta_log e ausentes no cache centralizado de mínimo e máximo, nas tabelas ausência_min e ausência_max, respectivamente;
2. Inserção de dados das fontes, nas tabelas insere_min e insere_max, que possuam valores chaves iguais ao conteúdo das tabelas ausencia_min e ausencia_max respectivamente, agrupados com o mesmo critério de agregação do cache centralizado de mínimo e máximo;
3. Inserção, de tuplas da tabela insere_min e insere_max, no cache centralizado de mínimo e máximo respectivamente, levando em consideração o tamanho desejado do cache centralizado (v_size).

A figura 3.20 mostra o algoritmo de reconstrução dos caches centralizados.

```
Reconstrucao_cache(v_size)
-- Passo 1  (Inserção nas tabelas Ausencia_min e Ausencia_max)
INSERT INTO AUSENCIA_MIN [AUSENCIA_MAX]
SELECT b.non_aggregate_fieldname1,..,b.non_aggregate_fieldnamep
FROM (SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep
    FROM MINCACHE [MAXCACHE] GROUP BY
    non_aggregate_fieldname1,.,non_aggregate_fieldnamep) a
RIGHT OUTER JOIN
    (SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep
    FROM delta_log GROUP BY
    non_aggregate_fieldname1,..,non_aggregate_fieldnamep) b
ON a.non_aggregate_fieldname1 = b.non_aggregate_fieldname1 and ...
                and a.non_aggregate_fieldnamep = b.non_aggregate_fieldnamep
WHERE a.non_aggregate_fieldname1 IS NULL;
```

```
-- Passo 2  (Inserção nas tabelas Insere_min e Insere_max)
INSERT INTO INSERE_MIN [INSERE_MAX]
SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
            expression, count(*) as count
    FROM fonts_tables, dimension_tables   WHERE joinconditions and
           (non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
IN (SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep
    FROM AUSENCIA_MIN [AUSENCIA_MAX])
GROUP BY non_aggregate_fieldname1,..,non_aggregate_fieldnamep,expression

-- Passo 3  (Inserção de novos dados nos caches centralizados)
INSERT INTO MINCACHE [MAXCACHE]
(SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep
            expression, count  FROM INSERE_MIN [INSERE_MAX] A
    WHERE ||VSIZE ||>=(SELECT SUM(count)
             FROM INSERE_MIN [INSERE_MAX] B
    WHERE a.non_aggregate_fieldname1 = b.non_aggregate_fieldname1
       and...and a.non_aggregate_fieldnamep=b.non_aggregate_fieldnamep
       and b.expression <= a.expression [b.expression >= a.expression] ))
```

Figura 3.20: Algoritmo *reconstrução_cachê*

- **Algoritmo *ordenação***

As estruturas hierárquicas das tabelas delta-sumários e cache-sumários são representadas, no algoritmo, por uma estrutura de dados denominada de *nós*, apresentada na figura 3.21, em que o campo *id* representa a identificação de um determinado nó da hierarquia; *filho(1), ..., filho(n)* possuem as identificações dos nós descendentes do nó corrente; e os campos *atualiza(1),...., atualiza(m)* possuem informações sobre os nós ancestrais responsáveis pela atualização das tabelas delta-sumários e cache-sumários correntes.

No algoritmo *ordenação*, apresentado na figura 3.22, é feita uma varredura por profundidade em todos os nós do dígrafo, levando em consideração apenas os campos *id* e *filhos*, com o objetivo de definir a ordem de atualização das tabelas delta-sumários e cache-sumários, considerando a relação de dependência entre essas tabelas.

Figura 3.21: Estrutura de dados *nós*

A *ordenação* é iniciada com a verificação da existência de nós descendentes (idx ≠ null). Caso possuam, seus nós são todos percorridos. Caso contrário, o identificador do nó corrente (*id*) é inserido em uma tabela denominada de *ordem*, se ainda não foi inserido.

```
ordenacao(idx)
begin
  /* se existem filhos, a varredura continua */
  if idx ≠ null
    filho ← select filho from nos where id = idx;
    if filho ≠ null
      for i : 1..filho.count loop
        ordenacao(filho(i));
      end for;
    end if;
  end if;
  /* Verificar se já está na tabela ordem */
  qtd ← select count(*) from ORDEM where NO_ORDEM = idx;
  if (qtd = 0) then
      /* Insere na tabela ordem */
      insert into ORDEM (NO_ORDEM ) values (idx);
  end if;
end.
```

Figura 3.22: Algoritmo *ordenação*

Baseado no exemplo de motivação da seção 3.3.2 e nos campos *id* e *filho* da estrutura de dados *nós* da figura 3.21, a figura 3.23 mostra a hierarquia das tabelas delta_sumários e cache-sumários.

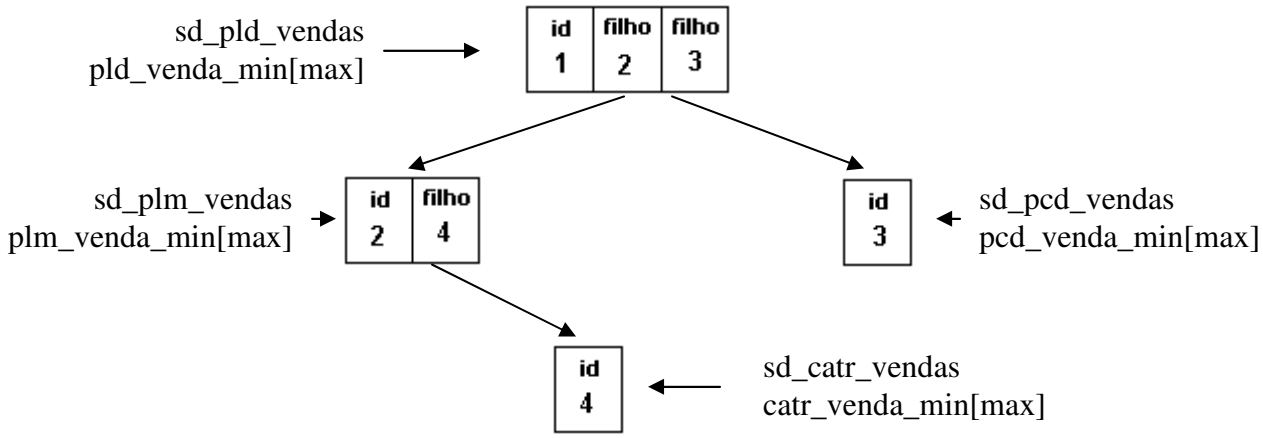


Figura 3.23:Hierarquia da estrutura de dados *nós* parcial

- **Algoritmo *propaga_nos***

O algoritmo *propaga_nos* utiliza a estrutura de dados *nós* da figura 3.21 completa. O campo *atualiza* é apresentado na figura 3.24, em que a coluna *Tabela-Delta-Sumário* contém

o nome da visão delta-sumário que será atualizada e o campo *Tabela-Delta-Sumário-Base* contém o nome da tabela delta-sumário ancestral que será usada para atualizar a tabela delta-sumário corrente.

Figura 3.24: Formato do campo *atualiza*

O campo *Insere-Delta-Sumario* contém uma instrução que insere dados agrupados de logs em uma tabela delta-sumário.

Os campos *Exclua-Cache-Sumário-Mínimo* e *Exclua-Cache-Sumário-Máximo* contêm instruções que eliminam tuplas das tabelas cache-sumário-mínimo e cache-sumário-máximo, respectivamente, cujos valores chaves foram alterados nas tabelas delta-sumário.

Finalmente, os campos *Insere-Cache-Sumário-Mínimo* e *Insere-Cache-Sumário-Máximo* contêm instruções que inserem tuplas nos cache-sumário-mínimo e cache-sumário-máximo, respectivamente.

As instruções inseridas, na estrutura de dados *atualiza,* não terão a cláusula UPDATE, pois para usá-la, seria necessário a utilização de sub-consultas complexas, diminuindo a eficiência do algoritmo, conforme demonstrado em Labio *et al* [LYC+99].

Se o nó corrente tiver mais de um ancestral, a estrutura de dados terá mais de um campo *atualiza*, ou seja, um nó que possui m ancestrais, terá os campos *atualiza(1)* até *atualiza(m)*.

O algoritmo *propaga_nos*, mostrado na figura 3.25, tem como objetivo propagar, de forma otimizada, as alterações das fontes em tabelas delta-sumários e cache-sumários baseado na estrutura de dados *nós*.

```
propaga_nos(v_lcv_min)
begin
 ordem_leitura ← select no_ordem from ordem order by rownum desc;
 while existir tuplas em ordem_leitura
  atualiza ← select atualiza from nos where id = ordem_leitura;
  tamanho_base ← 0;
  tamanho ← 0;
  for j: 1..atualiza.count
      if atualiza(j).tabela_delta_sumario_base ≠ null
          tamanho ← select count(*) from  atualiza(j).tabela_delta_sumario_base;
      end if;
      if tamanho_base = 0 or tamanho_base > tamanho then
          tamanho_base ← tamanho;
          ordem ← j;
```

```
        end if;
    end for;
    delta ← atualiza(ordem).tabela_delta_sumario;
    insere_delta_sumario ← atualiza(ordem).Insere_delta_sumario;
    exclua_cache_sumario_min ← atualiza(ordem).Exclua_cache_sumario_minimo;
    exclua_cache_sumario_max←atualiza(ordem).Exclua_cache_sumario_maximo;
    insere_cache_sumario_min ← atualiza(ordem).Insere_cache_sumario_minimo;
    insere_cache_sumario_max ←atualiza(ordem).Insere_cache_sumario_maximo;
    execute insere_delta_sumario;
    if not (exclua_cache_sumario_min = null)  then
        execute exclua_cache_sumario_min ;
    end if;
    if not (exclua_cache_sumario_max = null)  then
        execute exclua_cache_sumario_max;
    end if;
    if not (insere_cache_sumario_min = null)  then
        execute insere_cache_sumario_min;
    end if;
    if not (insere_cache_sumario_max = null)  then
        execute insere_cache_sumario_max;
    end if;
    Delete From  || delta ||  where lcv <=  v_lcv_min;
  end for;
end;
```

Figura 3.25: Algoritmo *propaga_nos*

O algoritmo *propaga_nos* é iniciado com a ordenação inversa da tabela *ordem*. Essa inversão tem como objetivo determinar a ordem para executar a atualização das tabelas delta-sumários e cache-sumários.

Em seguida, cada nó é avaliado, verificando o tamanho de cada tabela delta-sumário ancestral para determinar a tabela que mais rapidamente atualizará a tabela delta-sumário corrente.

Posteriormente, instruções contidas na estrutura de dados *nós* para inserir tuplas nas tabelas delta-sumários, eliminar tuplas das tabelas cache-sumários de mínimo e máximo e inserir novos dados nas tabelas cache-sumários são executadas.

Finalmente, tuplas das tabelas delta-sumário já utilizadas, por todos os dispositivos móveis, são excluídas de acordo como parâmetro *v_lcv_min* enviado para o *algoritmo propaga_nos*.

A estrutura de dados *nós* possui instruções que atualizam as tabelas delta-sumários e cache-sumários. Essas instruções executam os seguintes procedimentos:

1. Atualização da tabela delta-sumário. Para atualizar uma tabela delta-sumário, dados de uma tabela delta_log ou de uma tabela delta-sumário ancestral (lattice) são agrupados por um determinado critério de agregação. Nessa agregação, as expressões e o contador são somados; e o maior lcv do agrupamento é obtido, a fim de informar o momento exato da última alteração ocorrida em uma determinada chave da tabela delta-sumário.
2. Atualização da tabela cache-sumário. Para atualizar uma tabela de cache_sumário, inicialmente tuplas da tabela cache_sumário que possuem valores chaves recentemente alterados, na tabela delta-sumário, são excluídas. Posteriormente, agrupam-se os dados do cache centralizado ou de um cache_sumário ancestral com o mínimo ou máximo valor de uma expressão, para atualizar o cache_sumário mínimo ou máximo corrente, respectivamente.

A figura 3.26 mostra as instruções SQL para atualizar as tabelas delta-sumários e cache_sumários de mínimo e máximo.

```
-- Passo 1 (Inserção de dados nas tabelas delta-sumários)
Insert into delta-sumário (non_aggregate_fieldname1,..,
        non_aggregate_fieldnamep, expression1, ...., expressionm, count, lcv)
  (Select non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
        sum(expression1) as expression1, ...., sum(expressionm)
        as expressionm, sum(count) as count, max(lcv) as lcv
        from delta_log [ou delta_sumario_ancestral]
        group by non_aggregate_fieldname1,.., non_aggregate_fieldnamep)
-- Passo 2 (Exclusão e inserção de dados nas tabelas cache-sumários)
Delete from cache_sumario_min [cache_sumario_max]
      where (non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
              IN (Select non_aggregate_fieldname1,..,non_aggregate_fieldnamep
             From delta_sumario  WHERE  lcv >= (select min(lcv) from delta_log)
             group by non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
Insert into cache_sumario_min [cache_sumario_max]
       (Select non_aggregate_fieldname1,..,non_aggregate_fieldnamep
          min(expression) as expression [max(expression) as expression] From
          Mincache (ou cache_sumario_min_ancestral)
               [Maxcache (ou cache_sumario_max_ancestral)], Dimension_tables
          Where JOINCONDITIONS  and (non_aggregate_fieldname1, ....,
          non_aggregate_fieldnamep) IN (select non_aggregate_fieldname1,..,
          non_aggregate_fieldnamep   from delta_sumario
          WHERE lcv >= (select min(lcv) from delta_log)))
```

Figura 3.26: Instruções para atualizar as tabelas delta-sumários e caches-sumários

A seção C.8, do apêndice C, mostra a estrutura de dados *nós*, da figura 3.21, preenchida com dados e instruções SQL usadas na atualização das tabelas delta-sumários e cache-sumários do exemplo de motivação da seção 3.3.2.

3.4.2 Algoritmo *atualização*

A hierarquia das tabelas sumarizadas é representada no algoritmo *atualização* por uma estrutura de dados denominada *nós_atualiza*, apresentada na figura 3.27, em que o campo *id* representa a identificação de um determinado nó da hierarquia; *filho(1),..., filho(n)* identificam os nós descendentes do nó corrente; o campo *Tabela-Sumário* contém o nome da tabela sumarizada que será atualizada, o campo *Tabela-sumário-exclusão-temp* contém uma instrução responsável pela criação de uma tabela temporária com tuplas que deverão ser excluídas da tabela sumarizada; o campo *Tabela-sumário-inserção-temp* contém uma instrução responsável pela criação de uma tabela temporária com tuplas que deverão ser inseridas na tabela sumarizada.

Os campos *Tabela-sumário-exclusão* e *Tabela-sumário-inserção* possuem instruções responsáveis pela exclusão e inserção de tuplas, respectivamente, na tabela sumarizada.

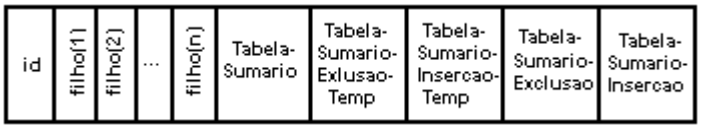


Figura 3.27: Estrutura de dados *nós_atualiza*

A atualização do DWM utiliza tabelas específicas (delta-sumário-local) com dados do servidor proxy, obtidos de tabelas delta-sumários e cache-sumários.

O algoritmo *atualização*, mostrado na figura 3.28, é iniciado com seleção do nó raiz da estrutura de dados *nós_atualiza*, composta de instruções que atualizam a tabela de fatos de um DWM. Em seguida, instruções de atualização das tabelas sumarizadas descendentes são executadas.

```
atualizacao(idx)
begin
  if (idx ≠ null)
   st ← select * from nos_atualiza where id= idx
```

```
   filho ← st.filho;
   qt ← filho.count;
   tabela_sumario ← st. tabela _sumario;
   tabela_sumario_exclusao_temp ← st.tabela_sumario_exclusao_temp;
   tabela_sumario_inserção_temp ← st.tabela_sumario_insercao_temp;
   tabela_sumario_exclusao  ← st.tabela_sumario_exclusao;
   tabela_sumario_inserção  ← st.tabela_sumario_insercao;
   execute tabela_sumario_exclusão_temp;
   execute tabela_sumario_inserção_temp;
   execute tabela_sumario_exclusão;
   execute tabela_sumario_inserção;
   if (qt ≠ 0)
     for i: 1..qt
       atualizacao(filho(i));
     end for;
   end if;
end.
```

Figura 3.28: Algoritmo *atualização*

As instruções contidas na estrutura de dados *nós_atualiza* foram inspiradas em Labio *et al* [LYC+99], denominado de *Stored Procedure com instalação de subconjuntos.* Estas instruções executam os seguintes procedimentos:

1. Criação de uma tabela *∇tabela_sumarizada* contendo todas as tuplas da tabela sumarizada que são afetadas pela tabela delta-sumário-local;
2. Criação de uma tabela *∆tabela_sumarizada* com o resultado da aplicação da tabela delta-sumário-local, com valores de mínimos e máximos, à tabela *∇tabela_sumarizada*;
3. Eliminação de tuplas da tabela sumarizada, com valores chaves presentes na tabela ∇tabela_sumarizada;
4. Inserção de tuplas da tabela ∆tabela_sumarizada na tabela sumarizada.

A figura 3.29 apresenta instruções SQL para atualizar tabelas sumarizadas.

```
-- Passo 1 (Criação da tabela ∇tabela_sumarizada)
CREATE TABLE ∇tabela_sumarizada AS
        (Select * FROM tabela_sumarizada
           where (non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
    IN (select non_aggregate_fieldname1,..,non_aggregate_fieldnamep
        from Delta-sumário-local))
```

```
-- Passo 2 (Criação da tabela Δ tabela_sumarizada)
CREATE TABLE  Δtabela_sumarizada AS
  (select non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
        expression1, ..., expressionm,
        min_expression1,...., min_expressionn
        max_expression1, ..., max_expressionq, count
  from (select non_aggregate_fieldname1,..,a.non_aggregate_fieldnamep,
       sum(expression1)  as expression1, ...., sum(expressionm)  as expressionm,
       min(min_expression1) as min_expression1, ....,
       min(min_expressionn) as min_expressionn,
       max(max_expression1) as max_expression1, ....,
       max(max_expressionq) as max_expressionq, sum(count) as count
    from ((select non_aggregate_fieldname1,..,
    non_aggregate_fieldnamep, expression1, .., expressionm,
    null as min_expression1,....,
    null as min_expressionn, null as max_expression1, ....,
    null as max_expressionq, count from  ∇tabela_sumarizada)
union all
     (select non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
       expression1, .., expressionm, min_expression1,....,
      min_expressionn, max_expression1, ..., max_expressionq,
     count from delta-sumário-local))
group by non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
having sum(count) > 0)

-- Passo 3 (Eliminação de tuplas da tabela sumarizada)
DELETE FROM TABELA_SUMARIZADA
    WHERE (non_aggregate_fieldname1,..,non_aggregate_fieldnamep) IN
    (SELECT non_aggregate_fieldname1,..,non_aggregate_fieldnamep
          FROM ∇tabela_sumarizada)


-- Passo 4 (Inserção de tuplas na tabela sumarizada)
INSERT INTO TABELA_SUMARIZADA  (SELECT * FROM
                  Δtabela_sumarizada)
```

Figura 3.29: Instruções para atualização de tabelas sumarizadas

A seção C.9, do apêndice C, mostra a estrutura de dados *nós_atualiza*, da figura 3.26, preenchida com dados e instruções SQL usadas na atualização das tabelas sumarizadas do exemplo de motivação da seção 3.3.2.

Uma outra abordagem para efetuar a atualização de tabelas sumarizadas, inspirada em [LYC+99], é substituir, completamente, o antigo conteúdo da tabela sumarizada utilizando os dados da tabela delta-sumário. Essa solução é denominada de *Stored Procedure com recriação.* Esta nova abordagem executa as seguintes tarefas:

1. Criação de uma tabela Δtabela_sumarizada com a aplicação da tabela delta-sumário à tabela sumarizada completa;
2. Exclusão da tabela sumarizada do banco de dados;
3. Substituição do nome da tabela Δtabela_sumarizada pela nova tabela sumarizada;
4. Criação dos índices.

A figura 3.30 apresenta instruções SQL para atualizar tabelas sumarizadas, utilizando esta nova abordagem.

```
-- Passo 1 (Criação da tabela Δ tabela_sumarizada)
CREATE TABLE  Δtabela_sumarizada AS
  (select non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
        expression1, ..., expressionm,
        min_expression1,...., min_expressionn
        max_expression1, ..., max_expressionq, count
  from (select non_aggregate_fieldname1,..,a.non_aggregate_fieldnamep,
       sum(expression1)  as expression1, ...., sum(expressionm)  as expressionm,
       min(min_expression1) as min_expression1, ....,
       min(min_expressionn) as min_expressionn,
       max(max_expression1) as max_expression1, ....,
       max(max_expressionq) as max_expressionq, sum(count) as count
    from ((select non_aggregate_fieldname1,..,
    non_aggregate_fieldnamep, expression1, .., expressionm,
    null as min_expression1,....,
    null as min_expressionn, null as max_expression1, ....,
    null as max_expressionq, count from   tabela_sumarizada)
union all
     (select non_aggregate_fieldname1,..,non_aggregate_fieldnamep,
       expression1, .., expressionm, min_expression1,....,
      min_expressionn, max_expression1, ..., max_expressionq,
      count from delta-sumário-local))
group by non_aggregate_fieldname1,..,non_aggregate_fieldnamep)
having sum(count) > 0)


-- Passo 2 (Eliminação de tuplas da tabela sumarizada)
DROP TABLE Tabela_sumarizada;


-- Passo 3 (Substituir o nome da tabela Δtabela_sumarizada para  nova tabela)
ALTER TABLE Δtabela_sumarizada RENAME TO tabela_sumarizada
```

Figura 3.30: Instruções para atualização de tabelas sumarizadas

3.5 Estratégia de comunicação entre DWMs e proxy

Os dados alterados das fontes são obtidos, pelas plataformas móveis, através de uma estratégia de comunicação, mostrado na figura 3.31 como um diagrama de seqüência UML.

A comunicação é iniciada com uma solicitação de conexão da plataforma com o servidor proxy. Após a conexão aceita, a plataforma móvel solicita o número da maior versão disponível no proxy (*lcv_max*).

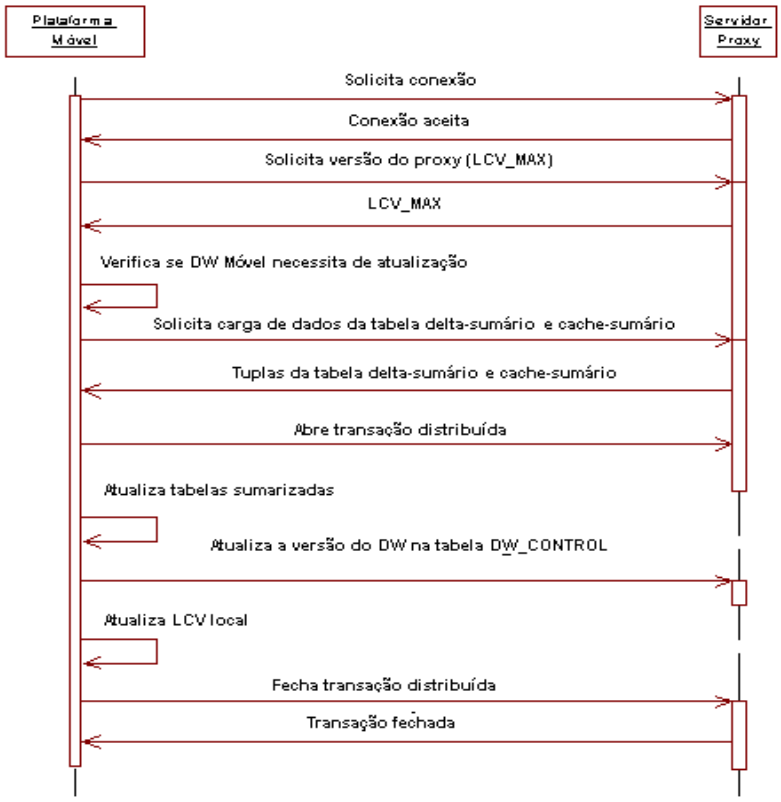


Figura 3.31: Estratégia de comunicação entre a plataforma móvel e o proxy

Em seguida, a plataforma verifica se a versão local está diferente da versão disponível no servidor proxy. Se a versão for igual, o DW da plataforma está atualizado, caso contrário, todos as tuplas das tabelas delta-sumário com a coluna lcv maior que a versão local e menor ou igual à versão disponível no proxy (lcv_max) serão solicitadas ao servidor proxy, juntamente com tuplas das tabelas cache-sumário.

As tuplas obtidas são gravadas na plataforma móvel em tabelas delta-sumários temporárias especiais. Essas tabelas delta-sumários são especiais pois suas tuplas terão

valores de mínimos e máximos de expressões obtidas a partir dos cache-sumários, caso as tabelas sumarizadas, que serão atualizadas, tenham estas funções de agregação.

Posteriormente, uma transação distribuída é aberta e as tabelas sumarizadas são atualizadas.

Finalmente, a versão do DWM é atualizada na tabela DW_CONTROL do servidor proxy e na tabela local de versão, da plataforma móvel (lcv); e a transação distribuída é fechada.

O MDWManager utiliza o protocolo de efetivação em duas fases (two-phase commit protocol) [LHK99].

3.5.1 Extração de dados do proxy

A extração de dados no proxy é baseada em uma hierarquia representada por uma estrutura de dados denominada *nós_busca*, apresentada na figura 3.32. Nesta estrutura, o campo *id* representa a identificação de um determinado nó da hierarquia, *filho(1),..., filho(n)* identificam os nós descendentes do nó corrente, o campo *Tabela-Delta-Sumário* contém o nome da tabela delta-sumário temporária que será criada na plataforma móvel e o campo *Tabela-Delta-Sumário-Criação* possui uma instrução responsável pela criação da tabela delta-sumário temporária.

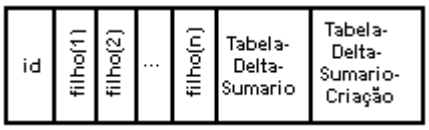


Figura 3.32: Estrutura de dados *nós_busca*

O algoritmo *obter_dados_proxy*, mostrada na figura 3.33, obtém dados das tabelas delta-sumários e cache-sumários necessários para a atualização das tabelas sumarizadas do DWM.

```
obter_dados_proxy(idx, v_lcv)
begin
  if (idx ≠ null)
    st ← select * from nos_busca where id= idx;
    filho ← st.filho;
    qt ← filho.count;
    tabela_delta_sumario ← st. tabela_delta_sumario;
    tabela_delta_sumario_criação ← st. tabela_delta_sumario_criação +
                                        "where lcv> v_lcv";
```

```
    execute 'drop table ' + tabela_delta_sumario;
    execute tabela_delta_sumario_criação;
  end if;
  if (qt ≠ 0)
     for i: 1..qt
        obter_dados_proxy(filho(i));
     end for;
   end if;
end.
```

Figura 3.33: Algoritmo *obter_dados_proxy*

A figura 3.34 apresenta a instrução para obter dados no servidor proxy. Essa instrução solicita dados a uma tabela delta-sumário e cache-sumário, criando localmente uma tabela delta-sumário temporária.

```
Create table tabela_delta-sumário-local as
   Select a.non_aggregate_fieldname1, ...,  a.non_aggregate_fieldnamep,
   a.expression1, ..., a.expressionm, b.expression1 as b.min_expression1, ....
   b.expressionn as b.min_expressionn,c.expression1 as b.min_expression1, ....
   c.expressionq as b.min_expressionq, a.count
from  (Select non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep,
        sum(expression1) as expression1,....,
        sum(expressionm) as expressionm, sum(count) as count
        from tabela_delta-sumário
        Group by non_aggregate_fieldname1, ...,
                                    non_aggregate_fieldnamep) a
Left outer join
            (Select non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep,
            expression1,...., expressionn from cache_sumário_min1, ....,
            cache_sumário_minn
            where cache_sumário_min1.non_aggregate_fieldname1 =
            cache_sumário_min2.non_aggregate_fieldname1 and .... and
            cache_sumário_min1.non_aggregate_fieldnamep =
            cache_sumário_min2.non_aggregate_fieldnamep and .... and
            .............
            cache_sumário_minn-1.non_aggregate_fieldname1 =
            cache_sumário_minn.non_aggregate_fieldname1 and .... and
            cache_sumário_minn-1.non_aggregate_fieldnamep =
            cache_sumário_minn.non_aggregate_fieldnamep) b
on a.non_aggregate_fieldname1 = b.non_aggregate_fieldname1 and ....
            and  a.non_aggregate_fieldnamep = b.non_aggregate_fieldnamep
Left outer join
            (Select non_aggregate_fieldname1, ...,  non_aggregate_fieldnamep,
            expression1,...., expressionq from cache_sumário_max1,...,
            cache_sumário_maxq,
            where cache_sumário_max1.non_aggregate_fieldname1 =
            cache_sumário_max2.non_aggregate_fieldname1 and .... and
            cache_sumário_max1.non_aggregate_fieldnamep =
```

```
            cache_sumário_max2.non_aggregate_fieldnamep and .... and
            .....
            cache_sumário_maxq-1.non_aggregate_fieldname1 =
            cache_sumário_maxq.non_aggregate_fieldname1 and .... and
            cache_sumário_maxq-1.non_aggregate_fieldnamep =
            cache_sumário_maxq.non_aggregate_fieldnamep) c
on a.non_aggregate_fieldname1 = c.non_aggregate_fieldname1 and ....
      and  a.non_aggregate_fieldnamep = c.non_aggregate_fieldnamep
```

Figura 3.34: Instrução para obter dados no proxy

A seção C.10, do apêndice C, mostra a estrutura de dados *nós_busca*, da figura 3.32, preenchida com dados e instruções SQL usadas na obtenção de dados para atualizar as tabelas sumarizadas do exemplo de motivação da seção 3.3.2.

3.6 Considerações sobre requisitos

Nesta seção, as tabelas 3.10 e 3.11 detalham as ações implementadas na construção do MDWManager, que permitiram o atendimento dos requisitos funcionais e não funcionais, respectivamente.

Requisitos Funcionais	
Referência	**Ações implementadas**
UF1	Os DWMs permitem que os usuários efetuem consultas OLAP, mesmo durante longos períodos de desconexão.
UF2	A extração incremental de dados, executadas pelas plataformas móveis; e os algoritmos utilizados na atualização do DWM permitem um aumento no número de visões materializadas, aumentando a eficiência das consultas OLAP.
UF3 e UF6	A independência da plataforma móvel, em relação ao servidor proxy, permite a criação de aplicativos com perfis de atualização para cada usuário.
UF4	Os algoritmos utilizados no MDWManager aumentam o tempo de disponibilidade do DWM para consultas OLAP.
PM2	A manutenção do DWM é realizada pelo procedimento de *atualização* na plataforma móvel, utilizando algoritmos incrementais.
PM3 e PM6	O DWM recebe um número de versão (tabela LCV) que é responsável, dentre outras coisas, pela verificação da necessidade de reconstrução. Caso o conteúdo da tabela LCV do DWM seja menor do que o mínimo lcv da tabela DW_CONTROL, a plataforma informará a necessidade de reconstruir o DWM.

PM4	A plataforma móvel é o único componente da arquitetura que solicita dados ao servidor proxy. Estes dados são utilizados na atualização um DWM.
PM5	As tabelas delta-sumários e caches, localizadas no servidor proxy, impedem o acesso direto entre as plataformas móveis e as fontes de dados.
PM6	A tabela LCV, da plataforma móvel, tem como objetivo principal determinar os dados, do proxy, necessários na atualização de um DWM.
PM7	Após a atualização do DWM, uma *estratégia de comunicação* solicita a atualização da versão de seu DWM, na tabela DW_CONTROL do servidor proxy.
FD1 e FD2	A utilização de triggers permite que todas as alterações ocorridas nas fontes sejam enviadas imediatamente ao servidor proxy, através de arquivos de logs específicos, garantindo sua independência e autonomia.
SP1	Na arquitetura do MDWManager, o servidor proxy recebe as mudanças que ocorrem nas fontes através de arquivos de logs. Quando ocorre uma inserção, exclusão ou alteração nas fontes, um trigger é disparado a fim de inserir valores em determinados arquivos de log no servidor proxy.
SP2	A tabela DW_CONTROL manterá um controle das versões de todos os DWMs. Esta tabela pode ser usada para determinar as atualizações propagadas e não propagadas para os DWMs.
SP3	O procedimento de *preparação* pode determinar tuplas, das tabelas delta-sumários, já utilizadas por todos os DWMs, podendo ser removidas. A tabela DW_CONTROL é responsável pelo fornecimento desta informação.
SP4	As identificações dos DWMs são registradas pelo administrador de banco de dados na tabela DW_CONTROL.
SP5 e SP7	O procedimento de preparação é executado, automaticamente, quando os logs chegam a um determinado volume. Na preparação, os dados dos logs são propagados nas tabelas delta-sumários e caches; e, depois, esvaziados.
SP6	O servidor proxy executa as instruções SQL enviada pela plataforma móvel (algoritmo obter_nos_proxy), fornecendo dados necessários das tabelas delta-sumáro e cache-sumário à plataforma móvel a fim de atualizar o DWM.
SP8	Os caches centralizados do servidor proxy permitem a diminuição do número de acessos às fontes de dados para calcular novos valores de funções de agregação

	de mínimos ou máximos.
SP9	A preparação, executada pelo servidor proxy, propaga os dados dos logs em tabelas delta-sumários e cache-sumários com os mesmos critérios de agregação das visões materializadas existentes nas plataformas móveis.
SP10	A adoção de tabelas delta-sumários, com descartes, garantem que o servidor proxy não terá visões materializadas, mas somente as alterações das fontes.

Tabela 3.10: Atendimento dos requisitos funcionais

Requisitos Não Funcionais	
Referência	**Ações implementadas**
RNF1	A preparação executada pelo servidor proxy permite a redução do volume de dados enviados para o DWM.
RNF2	Os algoritmos desenvolvidos para o MDWManager garantem um desempenho eficiente na preparação de dados e na atualização do DWM, comprovado através de diversas investigações.
RNF3	A preparação, executada no servidor proxy, aumenta a carga de trabalho do servidor proxy e diminui o trabalho realizado pela plataforma na atualização do DWM.
RNF5 e RNF6	O proxy é uma aplicação relacional sobre o SGBD Oracle 9i.

Tabela 3.11: Atendimento dos requisitos não funcionais

Capítulo 4

Avaliação Experimental

A avaliação experimental teve como objetivo principal determinar a melhor abordagem para a preparação de dados no servidor proxy e para atualização de DWs nas plataformas móveis. A melhor abordagem foi aquela que proporcionou um desempenho mais eficiente entre os algoritmos do MDWManager.

Na avaliação do desempenho da preparação de dados, foram utilizadas duas diferentes abordagens: cursor e stored procedure e na avaliação da atualização de um DWM três diferentes abordagens foram avaliadas: cursor e duas variações de stored procedure.

Além das avaliações de desempenho, foi investigada a carga de trabalho do servidor proxy, quando submetido às solicitações individuais e concorrentes, nas atualizações de DWMs.

4.1 Plano de testes

Para alcançar os objetivos da avaliação experimental alguns aspectos foram considerados em nossas investigações:

4.1.1 Arquiteturas

Dois tipos de arquiteturas foram utilizados na avaliação experimental. A primeira arquitetura, denominada *Arquitetura (1)*, era composta de um servidor proxy fixo e uma plataforma móvel (mobile host)

Esta arquitetura foi utilizada para avaliar as abordagens da preparação dos dados no servidor proxy e na atualização das visões materializadas nas plataformas móveis.

A arquitetura seguinte, denominada *Arquitetura (2)*, foi composta por um servidor proxy fixo e quatro plataformas móveis.

Esta arquitetura foi utilizada para investigar a carga de trabalho do servidor proxy quando submetido às solicitações individuais e concorrentes para atualizações de DWMs.

4.1.2 Algoritmos

O algoritmo *preparação*, no servidor proxy, manteve um cache centralizado de mínimo e máximo para uma determinada expressão, com três candidatos (três tuplas) para cada chave; uma tabela delta-sumário; e cache-sumários de mínimo e máximo para manter uma determinada tabela sumarizada.

O algoritmo *atualização*, na plataforma móvel, manteve um DWM com uma tabela sumarizada que possui valores de funções de agregação de mínimo e máximo.

A *estratégia de comunicação* foi avaliada através de investigações sobre carga de trabalho do servidor proxy quando submetido a diferentes demandas dos DWMs.

4.1.3 Abordagens da preparação

A preparação dos dados, executado no servidor proxy, utilizou duas abordagens: *cursor* e *stored procedure*, para atualizar os caches e uma tabela delta-sumário.

A abordagem *stored procedure* é apresentada na subseção 3.4.1, onde são executados os seguintes passos:

- Atualizações dos caches centralizados de mínimo e máximo, utilizando os logs;
- Propagação dos logs em tabelas delta-sumário;
- Atualizações dos cache-sumários com os novos dados dos caches centralizados.

A abordagem *cursor* efetuou as mesmas tarefas, porém, utilizando repetições baseadas em cursor.

4.1.4 Abordagens da atualização

As atualizações executadas nas plataformas móveis, utilizaram três abordagens para efetuar a manutenção de tabelas sumarizadas:

- Cursor;
- *Stored procedure (1)*, mostrada na subseção 3.4.2 como *stored procedure com instalação de subconjuntos*, consiste na criação de tabelas temporárias que serão aplicadas às tabelas sumarizadas. Esta abordagem somente altera porções modificadas nas tabelas sumarizadas ou inserem dados novos nesta;
- *Stored procedure (2)*, mostrada na subseção 3.4.2 como *stored procedure com recriação*, consiste na aplicação das tabelas delta-sumários às tabelas sumarizadas inteiras.

4.1.5 Volumes do banco de dados fonte

Na avaliação experimental, as fontes de dados foram formadas por tabelas com 500.000, 1.000.000 e 1.500.000 tuplas.

4.1.6 Volumes das visões materializadas

As investigações foram feitas sobre dois tipos de visões materializadas:

- Visão materializada com taxa de compressão fixa. Essa visão materializada cresce na mesma proporção do banco de dados (BD) fonte. Os volumes dessas visões materializadas são determinados por uma taxa de compressão fixa. Por exemplo, para uma tabela fonte de 1.000.000 de tuplas, com uma taxa de compressão de 20%, o volume de uma visão materializada correspondente é 20% x 1.000.000 = 200.000, ou seja, em média 1 tupla da visão materializada agrega 5 (1.000.000 / 200.000) tuplas do BD fonte;
- Visão materializada com volume fixo. Esta visão materializa geralmente são pequenas e de tamanho fixo independente do volume do BD fonte. Por exemplo, uma visão materializada em que os critérios de agregação são Categoria de Produtos e Região onde está localizada a loja, possui um volume pequeno e constante em função dos números de categorias e lojas serem reduzidos e relativamente fixos.

Nas investigações sobre atualizações de visões materializadas com taxa de compressão fixa, os volumes variaram de acordo com o tamanho do BD fonte e a taxa de compressão considerada.

Nos estudos sobre as atualizações de visões materializadas fixas, as visões tinham 500, 1.000, 2.000, 4.000, 6.000, 8.000 e 10.000 tuplas.

Finalmente, nas investigações que avaliaram a carga de trabalho do servidor proxy, as visões materializadas com volumes fixos estavam vazias antes da primeira solicitação e com 50.000, 100.000 e 150.000 tuplas, após a primeira e a segunda solicitação.

4.1.7 Taxas de compressão das visões materializadas

A preparação e a atualização de visões materializadas com volumes variáveis utilizaram taxas de compressão de 10% e 20%.

4.1.8 Taxas de atualização das visões materializadas

A taxa de atualização é o percentual de modificação de uma visão materializada, ou seja, uma taxa de atualização *r* de uma visão materializada, significa que a quantidade de atualização é *r* x volume da visão.

Nas investigações sobre a preparação de dados no servidor proxy e atualização de DWM, as taxas de atualização de visões materializadas com taxa de compressão fixa variaram de 0,2% a 1% com incremento de 0,2%; e de 2% a 20%, com incremento de 2%, enquanto as taxas de atualização de visões materializadas com volume fixo variaram de 10% a 100%.

Nos estudos sobre a carga de trabalho do servidor proxy, a taxa de atualização das visões materializadas foi de 100%.

4.1.9 Volumes das tabelas delta-sumários

Nos estudos sobre a carga de trabalho do servidor proxy, as plataformas móveis solicitaram 50.000, 100.000 e 150.000 tuplas às tabelas delta-sumário com iguais conteúdos.

Na solicitação seguinte, as plataformas móveis solicitaram as mesmas quantidades de tuplas às tabelas delta-sumários com 100.000, 200.000 e 300.000 tuplas.

4.1.10 Ambiente operacional

O servidor proxy foi instalado em um Pentium III - 750 MHz com memória RAM de 512 Mbytes e dotado de um SGBD Oracle 9i. As quatro plataformas móveis possuíam semelhantes configurações e estavam conectadas ao proxy por uma rede local de 100 Mbps.

4.2 Avaliação da preparação de dados

As figuras 4.1(a), 4.1(b) e 4.1(c) mostram os desempenhos das propagações de dados das fontes, em caches e em tabelas delta-sumários, que serão usados na atualização de visões materializadas com taxas de compressão fixa de 10% e 20% em relação aos volumes dos BDs fontes com 500.000, 1.000.000 e 1.500.000 tuplas respectivamente, variando as taxas de atualização de 0,2% a 1%, com incremento de 0,2%.

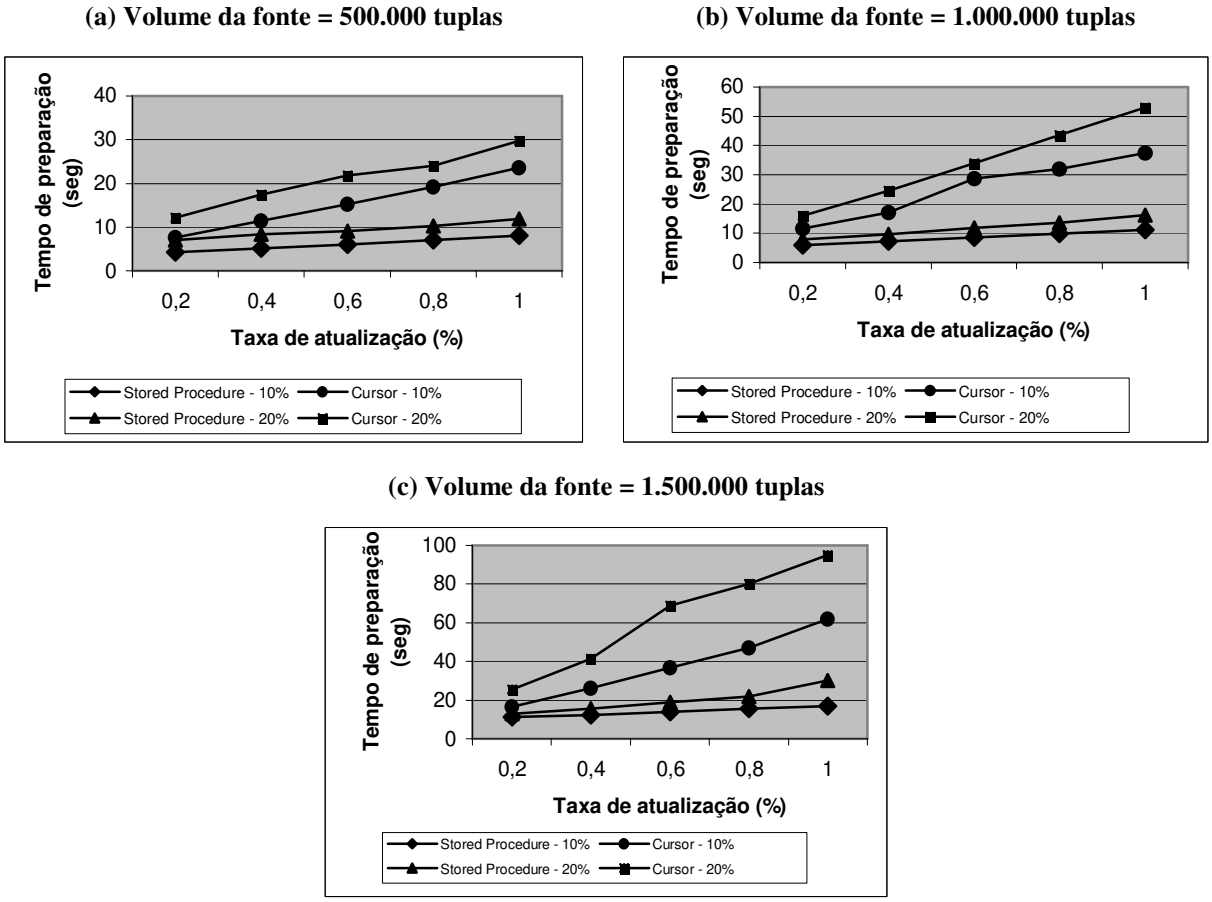


Figura 4.1: Performance do algoritmo de preparação de dados com taxas de atualização de 0,2 a 1%

Os gráficos da figura 4.1 mostram que a abordagem *stored procedure* é superior à abordagem *cursor* na atualização de tabelas caches (centralizado e sumário) e delta-sumários mantidas no servidor proxy. As inclinações das curvas demonstram ainda que *stored procedure* é a abordagem mais adequada independente do volume do BD fonte.

As figuras 4.2(a), 4.2(b) e 4.2(c) mostram os desempenhos das propagações de dados no proxy, que serão usados na atualização de visões materializadas com taxas de compressão fixa de 10% e 20% em relação aos volumes dos BDs fontes com 500.000, 1.000.000 e 1.500.000 tuplas respectivamente, variando as taxas de atualização de 2% a 20%, com incremento de 2%.

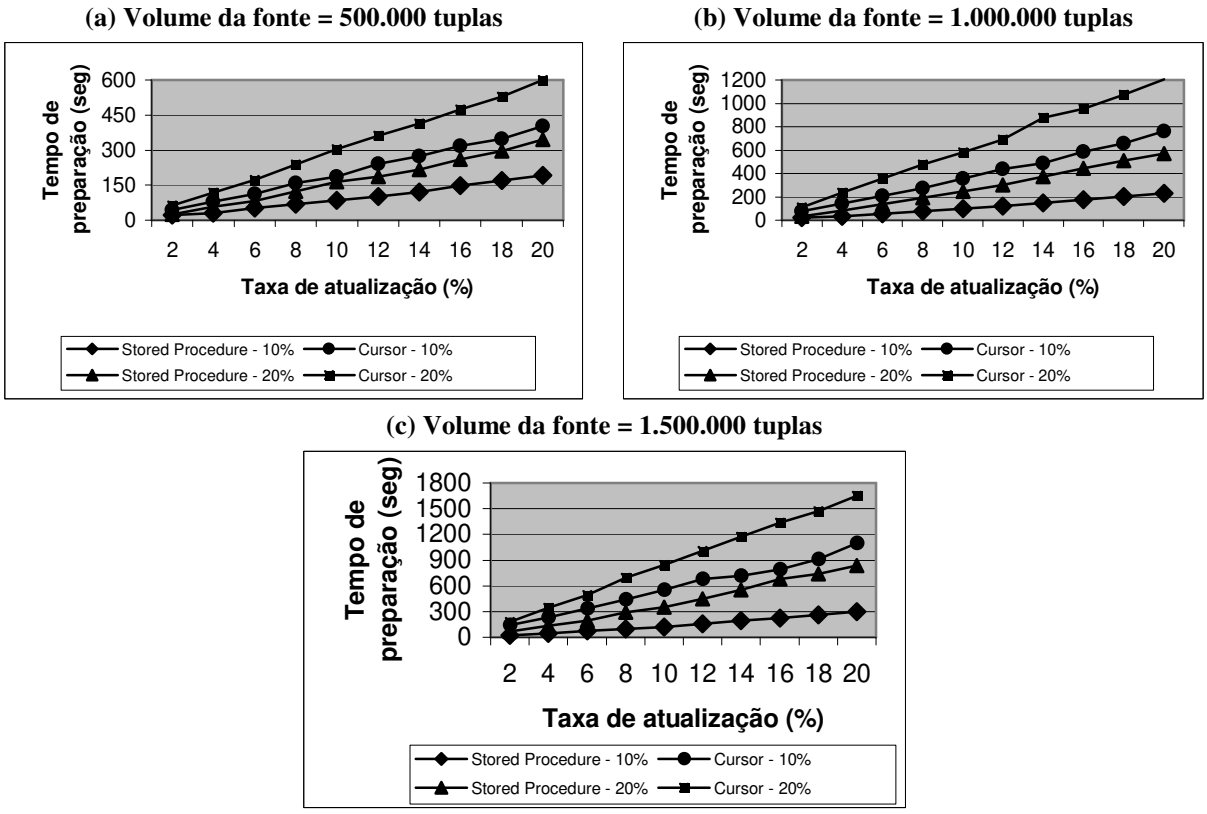


Figura 4.2: Performance do algoritmo de preparação de dados com taxas de atualização de 2% a 20%

Os gráficos da figura 4.2, semelhante ao estudo anterior, mostram que a abordagem *stored procedure* é superior à abordagem *cursor* na atualização de tabelas caches (centralizado e sumário) e delta-sumários mantidas no servidor proxy. As inclinações das curvas demonstram, também, que *stored procedure* é mais adequada independente do volume do banco de dados fonte.

4.2.1 Conclusão dos estudos sobre preparação de dados

Os estudos realizados indicaram que a abordagem *stored procedure* constitui a forma mais adequada para a propagação de dados em caches e em tabelas delta-sumário que serão utilizados na atualização de visões materializadas com taxa de compressão de 10% e 20%; e taxas de atualização variando de 0,2% a 20%, independente do volume do banco de dados fonte.

O melhor desempenho da implementação usando *stored procedure* demonstrou que a otimização utilizada pelo SGBD para executar as instruções INSERT e DELETE em lote é mais eficiente do que execução de instruções baseadas em cursor.

4.3 Avaliação da atualização de DWMs

A avaliação dos algoritmos de atualização de DWM envolveu investigações sobre a manutenção de visões materializadas com taxas de compressão fixa e com volumes fixos.

4.3.1 Visões materializadas com taxas de compressão de 10% e 20%

As figuras 4.3(a), 4.3(b) e 4.3(c) mostram o desempenho na atualização de visões com uma taxa de compressão de 10% e 20% e taxas de atualização variando de 0,2% a 1%, com incremento de 0,2%, em relação aos volumes das fontes com 500.000, 1.000.000 e 1.500.000 tuplas respectivamente.

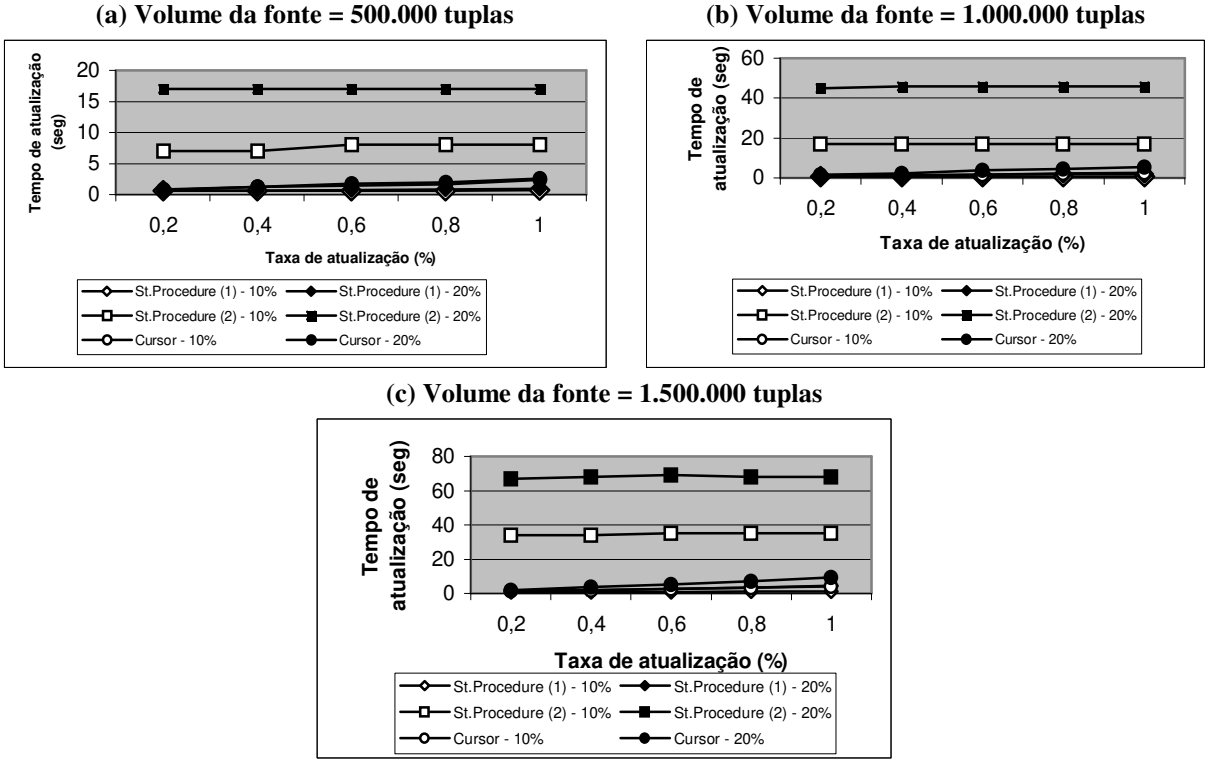


Figura 4.3: Performance do algoritmo de atualização de DWMs com taxas de atualização de 0,2% a 1%

Os estudos demonstraram que o desempenho da abordagem *stored procedure (1)* é superior às demais abordagens na atualização de visões materializadas com taxas de compressão fixas.

Para investigar, com mais eficiência, a influência do tamanho das fontes de dados no desempenho das atualizações de visões materializadas, nós variamos os volumes dos BDs

fontes de 500.000 a 1.500.000 tuplas e as taxas de atualização de 0,6% a 0,2%. O produto dos volumes das fontes com as taxas de atualização devem totalizar 300.000, ou seja, fontes com 500.000, 1.000.000 e 1.500.000 tuplas têm 0,6%, 0,3% e 0,2% de taxas de atualização, respectivamente.

A figura 4.4 mostra que as abordagens *stored procedure (1)* e *cursor*, na atualização de visões materializadas com taxas de compressão de 10% e 20%, possuem uma menor suscetibilidade às variações dos volumes das fontes, demonstrando que a abordagem *stored procedure (1)* é superior às demais, em função do melhor desempenho e da menor sensibilidade à variação do volume do BD fonte.

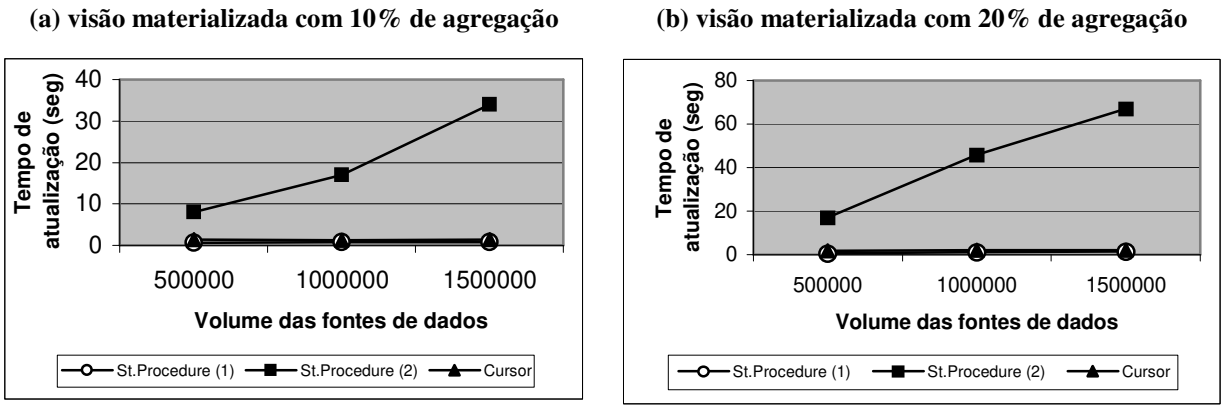
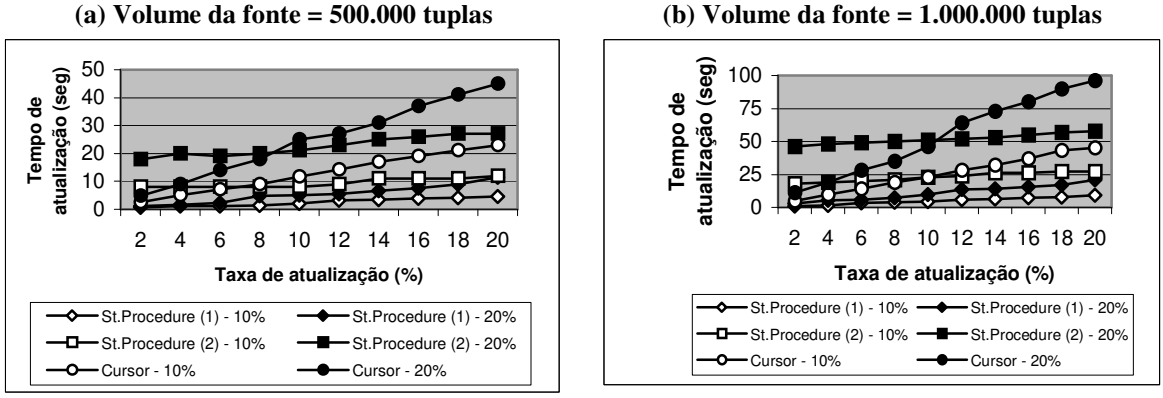


Figura 4.4: Performance do algoritmo de atualização de DWMs com taxas de atualização de 0,2% e 1% variando o volume das fontes de dados

As figuras 4.5(a), 4.5(b) e 4.5(c) mostram o desempenho na atualização de visões com uma taxa de compressão de 10% e 20% e taxas de atualização variando de 2% a 20% com incremento de 2%, em relação aos volumes das fontes com 500.000, 1.000.000 e 1.500.000 tuplas respectivamente.



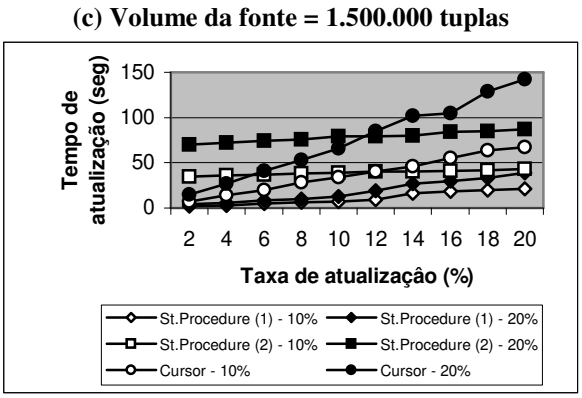


Figura 4.5: Performance do algoritmo de atualização de DWMs com taxas de atualização de 2% a 20%

Os estudos demonstraram que a abordagem *stored procedure (1)* é superior às demais abordagens, na atualização das visões materializadas dos DWMs.

Para investigar a influência do tamanho das fontes de dados no desempenho das atualizações de visões materializadas, nós variamos os volumes das fontes de 500.000 a 1.500.000 tuplas e as taxas de atualização de 12% a 4%. O produto do volume das fontes com as taxas de atualização devem totalizar 6.000.000, ou seja, fontes com 500.000, 1.000.000 e 1.500.000 tuplas têm 12%, 6% e 4% de taxas de atualização, respectivamente.

A figura 4.6 mostra que as abordagens *stored procedure (1)* e *cursor*, na atualização de visões materializadas com taxas de compressão de 10% e 20%, possuem uma menor suscetibilidade às variações dos volumes de dados das fontes, demonstrando que a abordagem *stored procedure (1)* é superior às demais, em função do melhor desempenho e da menor sensibilidade à variação do volume do BD fonte.

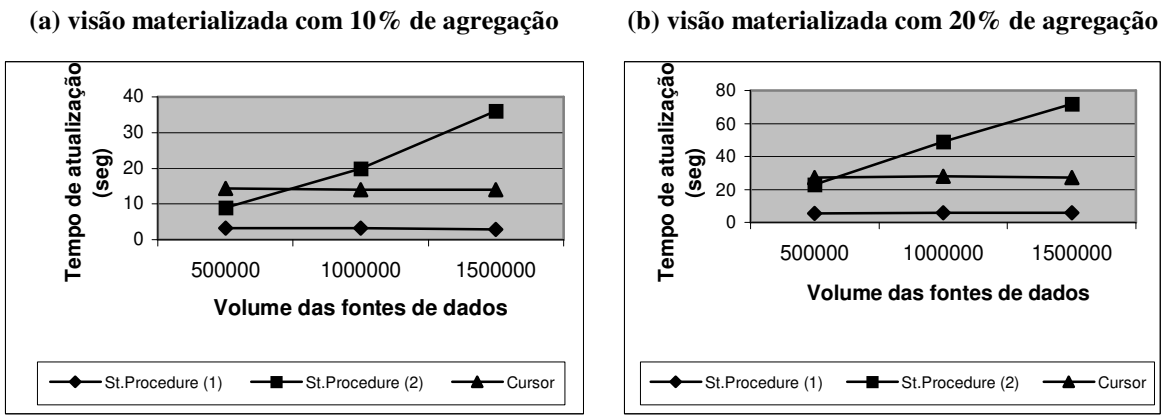


Figura 4.6: Performance do algoritmo de atualização de DWMs com taxas de atualização de 2% e 20% variando o volume das fontes de dados

4.3.2 Visões materializadas com volume fixo

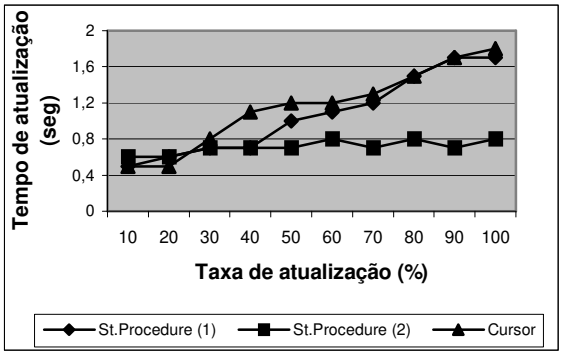
A figura 4.7 mostra os desempenhos das atualizações de visões materializadas com volumes fixos variando de 500 a 10000 tuplas e taxas de atualização variando de 10% a 100%, com incremento de 10%.

A figura 4.7(a) mostra o desempenho na atualização de uma visão materializada com 500 tuplas, em que se verifica a superioridade da abordagem *stored procedure (2)*, notadamente quando a taxa de atualização ultrapassa 25%, enquanto a abordagem *cursor* apresenta melhor desempenho para atualizações abaixo dessa taxa.

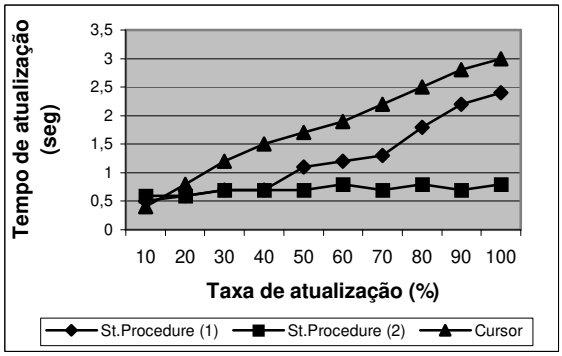
A figura 4.7(b) apresenta o desempenho na atualização de uma visão com 1000 tuplas, onde se verifica a superioridade da abordagem *stored procedure (2)*, principalmente quando a taxa de atualização ultrapassa 15%, enquanto a abordagem *cursor* apresenta melhor desempenho para taxas de atualização menores.

As figuras 4.7(c), 4.7(d), 4.7(e), 4.7(f) e 4.7(g) apresentam os desempenhos das atualizações de visões que possuem 2000, 4000, 6000, 8000 e 10000 tuplas respectivamente. Nestas atualizações, verificamos um melhor desempenho da abordagem *stored procedure (2)* independente da taxa de atualização.

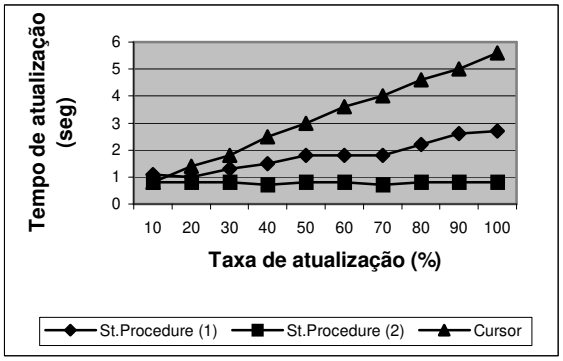
(a) Visão materializada com 500 tuplas



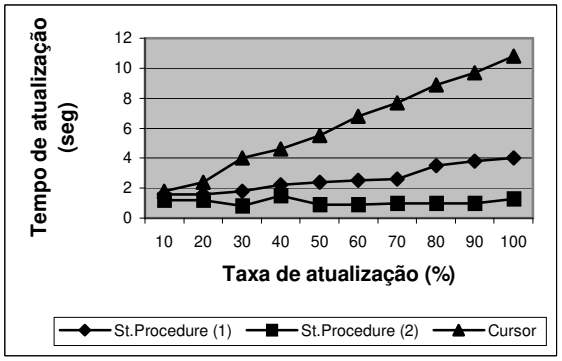
(b) Visão materializada com 1.000 tuplas



(c)Visão materializada com 2.000 tuplas



(d)Visão materializada com 4.000 tuplas



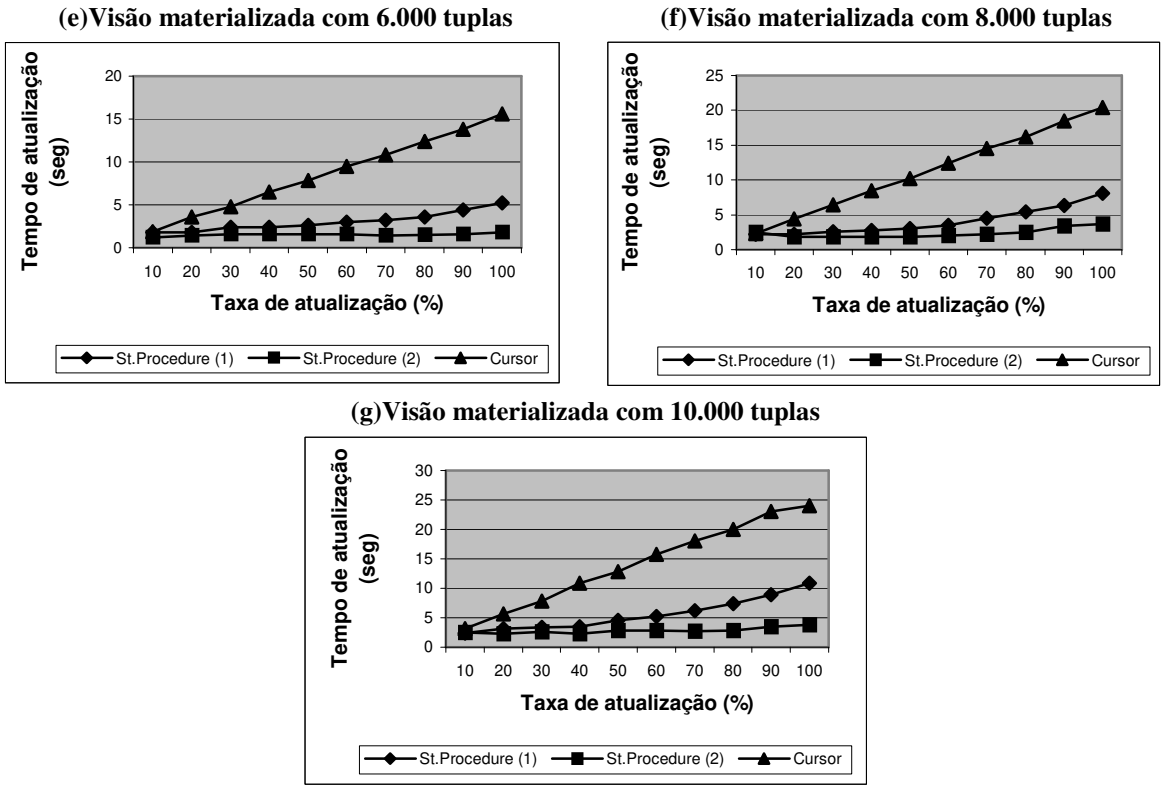


Figura 4.7: Performance do algoritmo de atualização de DWMs, com visões materializadas fixas

4.3.3 Conclusão dos estudos sobre atualização de DWMs

Os estudos realizados indicaram que a abordagem s*tored procedure (1)* é a mais adequada para a atualização de visões materializadas com taxas de compressão de 10% e 20%; e taxas de atualização variando de 0,2% a 20%, em função do melhor desempenho e da menor suscetibilidade à variação do volume do BD fonte.

O fraco desempenho da abordagem *stored procedure (2)*, na atualização de visões materializadas com taxa de compressão fixa, tem como principal causa a criação completa de uma nova visão materializada a partir da visão antiga, sendo portanto uma implementação altamente suscetível à variação do volume das fontes de dados.

Os estudos indicaram, também, que a abordagem *stored procedure (2)* possui um melhor desempenho na atualização de visões materializadas pequenas e com volumes fixos variando de 500 a 10.000 tuplas e taxas de atualização variando de 10% a 100%.

O desempenho constante da abordagem *stored procedure (2)*, na atualização de visões materializadas com volumes fixos, reside no número, também fixo, de tuplas lidas necessárias para criar uma nova visão materializada.

4.4 Carga de trabalho do servidor proxy

Nessa investigação foi avaliado o desempenho do servidor proxy quando submetido a diferentes demandas provenientes de quatro plataformas móveis, utilizando a *arquitetura (2)*.

4.4.1 Solicitações individuais das plataformas móveis

Inicialmente as plataformas MH1, MH2, MH3 e MH4 solicitaram dados, individualmente, às tabelas delta-sumários no servidor proxy com o objetivo de inseri-los em visões materializadas vazias.

Posteriormente, novas solicitações foram feitas, às tabelas delta-sumário, a fim de atualizar as referidas visões materializadas.

Entre as duas solicitações, as fontes foram alteradas, duplicando os volumes das tabelas delta-sumários.

A figura 4.8 apresenta os tempos médios para se obter dados no proxy de forma individualizada, variando o número de tuplas solicitadas, em que se verifica um aumento no tempo de transferência de 20% entre a primeira e segunda solicitação. Esse aumento deveu-se ao fato da seleção dos dados, na segunda solicitação, ter sido feita sobre um maior volume de dados.

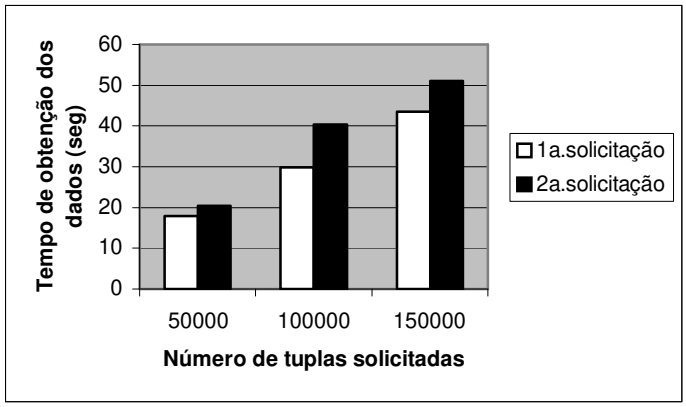


Figura 4.8: Tempo da obtenção de dados no servidor proxy

A figura 4.9 apresenta os números médios de tuplas obtidas por segundo na primeira e na segunda solicitação. Nesse processo está incluído o tempo usado pelo SGBD para executar a consulta no servidor proxy.

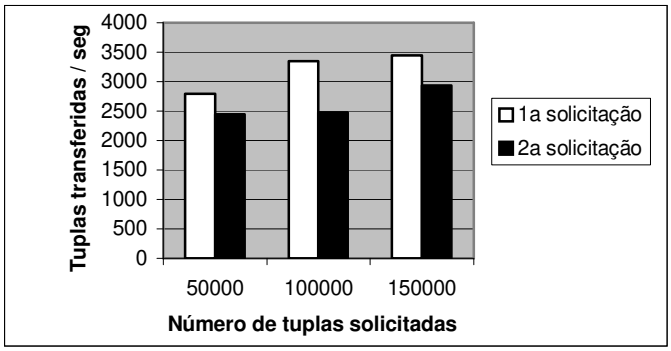


Figura 4.9: Taxa de transferência de tuplas

Nesse estudo, verificou-se a diminuição de tuplas transferidas entre a primeira e a segunda solicitação. Essa diferença reside no tempo despendido pelo SGBD na seleção de dados no proxy, pois esta foi feita sobre um maior número de tuplas existentes nas tabelas delta-sumário.

Outra observação importante é a elevação do número de tuplas transferidas por segundo, quando do aumento do volume das tabelas delta-sumários. Esse aumento demonstra que a carga de trabalho do servidor proxy, para processar uma consulta, não aumenta na mesma proporção do número de tuplas solicitadas.

4.4.2 Solicitações concorrentes de quatro DWMs

Nessa investigação, as plataformas MH1, MH2, MH3 e MH4 solicitaram dados ao proxy, de forma concorrente, a fim de atualizar suas visões.

A figura 4.10 apresenta os tempos médios para se obter dados no proxy de forma concorrente, variando o número de tuplas solicitadas, onde se verifica um aumento no tempo de transferência de 18% entre a primeira e a segunda solicitação. Esse aumento deveu-se ao fato da seleção dos dados, no servidor proxy, ter sido feita sobre um maior volume de dados na tabela delta-sumário.

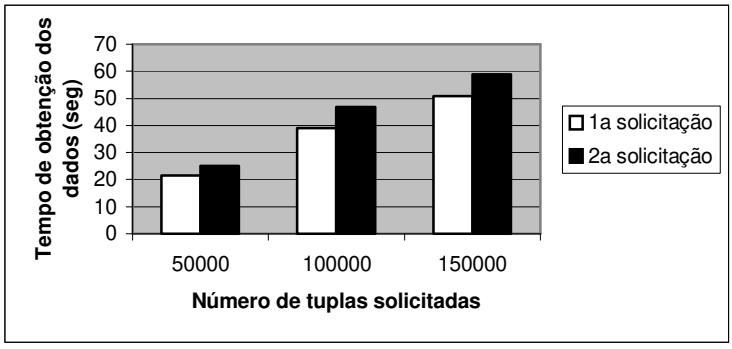


Figura 4.10: Tempo da obtenção de dados no servidor proxy

Nesses estudos observou-se, também, um aumento médio de 20% da carga de trabalho do servidor proxy para atender solicitações concorrentes em relação às solicitações individuais.

A figura 4.11 apresenta os números médios de tuplas obtidas no servidor proxy, por segundo, de forma concorrente, na primeira e na segunda solicitação. Nesse processo está incluído o tempo usado pelo SGBD para seleção dos dados no servidor proxy.

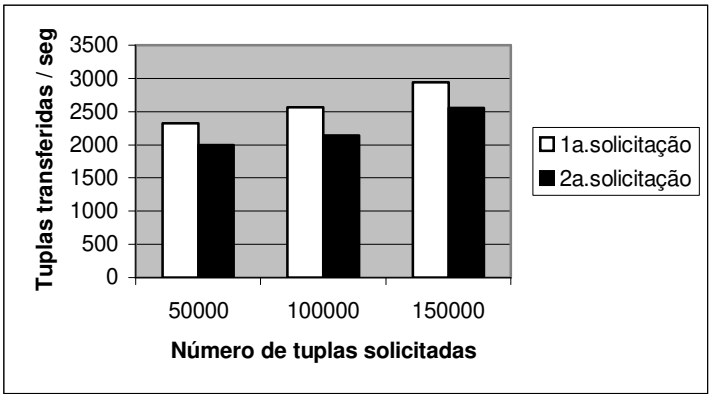


Figura 4.11: Taxa de transferência de tuplas

Esta figura mostra a diminuição da taxa de transferência entre a primeira e a segunda solicitação. Essa diferença reside no tempo despendido pelo SGBD na seleção de dados no proxy, pois esta foi feita sobre um maior número de tuplas existentes nas tabelas delta-sumário.

A elevação da taxa de transferência quando do aumento do volume da tabela delta-sumário demonstrou que a carga de trabalho do servidor proxy não aumenta na mesma proporção do número de tuplas solicitadas.

4.4.3 Atualização de DWMs

A figura 4.12 mostra o tempo para a plataforma móvel proceder às inserções e alterações nas visões materializadas. Esta figura mostra que as inserções em visões vazias são feitas com tempos semelhantes para solicitações individuais (ind) e concorrentes (con). O mesmo acontece com relação às alterações das visões, pois o processo é executado totalmente na plataforma móvel.

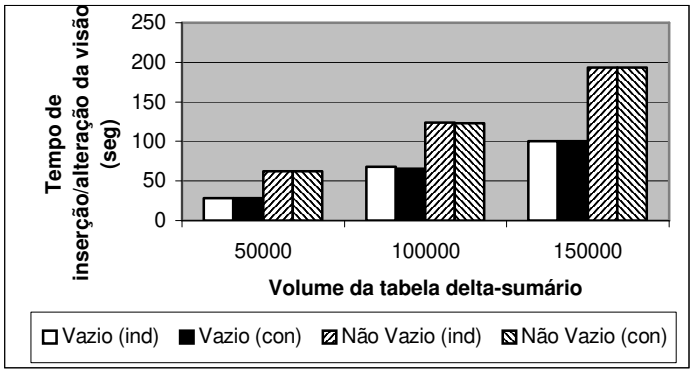


Figura 4.12: Performance do algoritmo de atualização de DWMs, com ou sem concorrência

A figura mostra um acréscimo, médio, de 95% no tempo para efetuar uma alteração em uma visão materializada em relação ao tempo para executar somente inserções, pois uma alteração é feita através de uma exclusão e uma posterior inserção.

A figura 4.13 mostra que o número de inserções em visões vazias é superior ao número de tuplas alteradas em visões não vazias, pois como já relatado anteriormente, as alterações são processadas com exclusões e posteriores inserções.

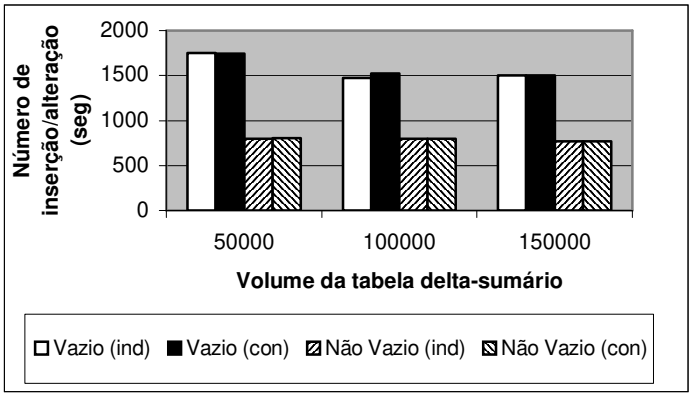


Figura 4.13: Número de inserções e alterações em visões materializadas

4.4.4 Conclusão dos estudos sobre a carga do servidor proxy

Os estudos realizados mostraram que o aumento da carga de trabalho do servidor proxy para quatro conexões simultâneas não pode ser considerado significante em relação ao processo total de atualização de DWM, representando um aumento de apenas 5% no tempo de atualização total (obtenção + atualização propriamente dita) quando a solicitação é

concorrente. Entretanto, convêm salientar, que o custo e a velocidade da rede podem ser fatores limitantes no processo de atualização de DWs em ambientes móveis.

Capítulo 5

Conclusão

Neste trabalho, desenvolvemos um software, denominado MDWManager — *Mobile Data Warehouse Manager* — que é responsável pela manutenção incremental de data warehouses móveis (DWMs).

Um DWM é um DW que reside em uma plataforma móvel, que pode ser um laptop, ou notebook, ou PDA, permitindo que usuários efetuem consultas OLAP mesmo durante longos períodos de desconexão das fontes de dados, sem a necessidade de poderosas plataformas móveis.

Plataformas móveis impõem várias restrições ao funcionamento de DWMs. Essas restrições são levadas em consideração pelo MDWManager, entre as quais, citamos: desconexão constante das fontes de dados; em geral, limitada capacidade das plataformas móveis; alto custo e baixa velocidade dos canais de comunicação.

A arquitetura do MDWManager, inspirada na pesquisa de Stanoi *et al* [SAE+99], é composta, basicamente, de um servidor proxy fixo, o qual é responsável pela extração de dados das fontes e pela manutenção de controles internos necessários à atualização de visões materializadas nos dispositivos móveis — DWM.

O proxy é uma aplicação relacional sobre o SGBD Oracle9i. O processo de manutenção das tabelas de controle do proxy, denominado *preparação*, tem como finalidades principais:

- Eliminar o acesso direto entre DWMs e fontes de dados;
- Restringir o acesso entre o servidor proxy e as fontes de dados, em função da presença de caches;
- Diminuir o tempo de acesso e o volume de dados transferidos entre as plataformas móveis e o servidor proxy, transferindo para as plataformas móveis apenas dados sumariados;

- Aumentar o desempenho da atualização dos DWMs, realizando o maior número possível de tarefas no proxy.

No que diz respeito às estruturas de controle no proxy, a idéia de registrar valores das funções de agregação MIN e MAX, em um cache centralizado, foi influenciada pelo trabalho de Chan *et al* [CLS00]. Diferentemente, entretanto, da abordagem de [CLS00], em que os caches ficam localizados nas plataformas móveis, na nossa solução os caches são centralizados no proxy, desta forma diminuindo o tráfego entre o DWMs e o proxy, e dispensando as plataformas móveis do trabalho de atualização dos caches.

Outros componentes da estrutura de controle no proxy são as tabelas delta-sumário, utilizado por Mumick *et al* [MQM97], que sumarizam os dados alterados nas fontes de dados; e os cache-sumários que resumem os dados dos caches centralizados. Os dados dessas tabelas são enviados para as plataformas móveis, sob demanda, para atualizar seus DWMs.

Outra característica importante do MDWManager é a sua capacidade de gerenciar vários DWMs simultaneamente, utilizando um controle de versão denominado LCV (*last change version*), semelhante ao LCN (*last change number*) utilizado por [CLS00].

MDWManager oferece algoritmos incrementais segundo o enfoque de Labio *et al* [LYC+99], fortemente acoplados a SGBDs, tornando a preparação dos dados no servidor proxy e atualização dos DWMs mais eficientes em relação a algoritmos fracamente acoplados a SGBDs.

Ao contrário da abordagem de [LYC+99], que atualiza visões materializadas apenas com as funções SUM e COUNT, nós propomos algoritmos que atualizem visões materializadas com as funções de agregação clássicas SUM, COUNT, AVG, MIN e MAX, bem como outras funções de agregação disponíveis no SQL.

A eficácia de nossos algoritmos foi mostrada por meio de diversas avaliações experimentais.

A comunicação entre os DWMs e o servidor proxy é feita sob demanda das plataformas móveis, através de uma estratégia de comunicação que obtém dados das tabelas do servidor proxy de forma eficiente, com um reduzido volume de dados na rede e uma carga de trabalho relativamente pequena do servidor proxy, tudo isto evidenciado pelas investigações experimentais.

A tabela 5.1 apresenta um resumo comparativo das propostas estudadas [MQM97, CLS00, LYC+99, SAE+99, Ora02], assim como as principais características do MDWManager, baseado na tabela 2.6.

Característica	Propostas					
	MQM97	**CLS00**	**LYC+99**	**SAE+99**	**Oracle**	**MDWManager**
Gerência de DWMs simultâneos	Não	Não	Não	Sim	Sim	Sim
Servidor proxy fixo extraindo alterações das fontes e executando um pré-processamento.	Não	Não	Não	Sim	Não	Sim
Manutenção de visões materializadas com funções de agregações mínimo e máximo.	Sim	Sim	Não	Não	Somente inserções	Sim
Isolamento completo entre o DWM e as fontes.	Não	Não	Não	Não	Não	Sim
Atualização de visões materializadas através tabelas delta-sumário.	Sim	Não	Sim	Não	Não	Sim
Algoritmos acoplados ao SGBD.	Não	Não	Sim	Não	Sim	Sim
Processamento de atualização não concentrado no DW.	Não	Não	Não	Sim	Não	Sim
Propagação de alterações no servidor proxy otimizada automaticamente	Não	Não	Não	Não	Não	Sim
Estratégia de comunicação entre DWM e Proxy	Não	Não	Não	Não	Não	Sim
Novas funções de agregação	Não	Não	Não	Não	Parcial	Sim

Tabela 5.1: Comparativo das propostas

5.1 Trabalhos Futuros

Embora o trabalho tenha atingido os objetivos definidos, alguns estudos futuros podem ser realizados. Entre eles:

- As instruções SQL utilizadas na preparação de dados, obtenção de dados no servidor proxy e atualização dos DWMs são criadas manualmente, havendo a necessidade de desenvolver uma ferramenta para criar as hierarquias das tabelas sumarizadas e a partir dela, automaticamente, criar as estruturas dos logs e povoar as estruturas de dados com as instruções SQL necessárias para efetuar a preparação de dados, a obtenção dos dados no proxy e atualização dos DWMs;

- O modelo de DWM e arquitetura definida na proposta prevêem a existência de várias hierarquias de visões materializadas em um DWM e de tabelas delta-sumários no servidor proxy, porém o MDWManager foi criado suportando uma única hierarquia de tabelas delta-sumário no servidor proxy, havendo a necessidade de efetuar pequenas modificações que permitam múltiplas hierarquias;

- As estruturas de dados utilizadas pelo MDWManager permitem a manutenção de visões materializadas com diversos tipos de funções de agregação (SUM, COUNT, AVG, MIN, MAX, REGR, STDDEV, CORR dentre outras), entretanto existem funções de agregação (RANK, PERCENT_RANK, CUME_DIST dentre outras) que necessitam ser investigadas a fim de determinar se o MDWManager pode suportar estas funções;

- As investigações experimentais foram realizadas levando em consideração o número de tuplas, portanto sugerimos novas investigações no que tange a quantidade de bytes;

- A carga de trabalho do servidor proxy foi avaliada através de quatro solicitações simultâneas, havendo a necessidade de aumentar o número de solicitações concorrentes a fim de melhor avaliar o desempenho do servidor proxy.

Referências Bibliográficas

[Bar99] Barbará, D. ***Mobile Computing and Databases – A survey***. IEEE Transactions on Knowledge Engineering. páginas 108-117, 1999.

[BM90] Blakeley, J. A.; Martin, N. L. ***Join Index, Materialized View, and Hybrid- Hash Join: A Performance Analysis***. ICDE 1990: 256-263.

[Boa96] BoaVentura Netto, P. O. ***Grafos: Teoria, Modelos, Algoritmos***. Edgard Blv Cher, 418p, 1996.

[BP98] Badrinath, B. R.; Phatak, S. H. ***An Architecture for Mobile Databases***. Department of Computer Science Technical Report DCS-TR-351, Department of Computer Science, Rutgers University, New Jersey, 1998.

[Bre97] Breitner, C. A. ***Data warehousing and OLAP: Delivering Just-In-Time Information for Decision Support***, Proceedings of the 6th Intl. Workshop for Oeconomertics, Junho, Karlsruhe, Alemanha. 1997.

[BT98] Bokun, M.; Taglienti, C. ***Incremental Data Warehouse Updates***. 1998. Disponível em:<http://www.fortunecity.com/skyscraper/ oracle/699/orahtml/ dmreview/may98_60.html>. Acesso em 19/09/2002.

[CAM98] Campos, M, L. M.; Filho, A. V. R. ***Data Warehouse*** (tutorial). UFRJ, 1998. Disponível em:<http://genesis.nce.ufrj.br/dataware/tutorial.html>. Acessado em 08/08/2002.

[CBS99] Connolly, T.; Begg, C.; Strachan, A. ***Database Systems, A practical approach to design, implementation and management***, 2a, Edição, Addison-Wesley, 1999.

[CD97] Chaudhuri, C.; Dayal, U. ***An Overview of Data Warehousing and OLAP Technology***, SIGMOD Record, v. 26, nº. 1, pg. 65-74, 1997.

[CFG00] Cetintemel, U.; Franklin, M. J.; Giles, C. L. ***Self-Adaptive User Profiles for Large-Scale Data Delivery***. ICDE 2000: 622-633.

[CKL+97] Colby, L. S.; Kawaguchi, A.; Lieuwen, D. F.; Mumick, I.; Ross, K. ***Supporting Multiple View Maintenance Policies***. ACM SIGMOD *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2):405-416, Junho 1997.

[CLS00] Chan, M.; Va Leong, H.; Si, A. ***Incremental Update to Aggregated Information for Data Warehouses over Internet***. DOLAP 2000.

[Cou01] Cougo, P. S. ***Modelagem Conceitual e Projeto de Bancos de Dados***. Ed. Campus, 2001.

[CS94] Chaudhuri, S.; Shim, K. ***Including Group-By in Query Optimization***. In Proceedings of the twentieth International Conference on Very Large Darabases (VLDB), pages 354-366, Santiago, Chile, 1994.

[DER98] DeRose, S. ***XQuery: A unified syntax for linking and querying general XML documents***. In Query Languages 98 - The Query Languages Workshop, December 5, 1998.

[Dev97] Devlin, B. ***Data warehouse: from architecture to implementation***. Massachusetts: Addisson Wesley Longman, 1997. 432p.

[Dix97] Dixon, M. ***An overview of document mining technology***, 1997. Disponível em: <http://www.geocities.com/ResearchTriangle/ Thinktank/1997/mark/writings/dixm97_dm.ps>. Acesso em: 10/11/2002.

[DJ98] Derks, W.; Jonker, W. ***Practical Experiences with Materialized Views in a Very Large Data Store for Telecommunication Service Management***. DEXA Workshop 1998: 881-886

[DSB+99] Dinter, B.; Sapia, C.; Blaschka, M.; Höfling, G. ***OLAP Market and Research: Initiating the Cooperation***, Journal of Computer Science and Information Management, Vol. 2, No. 3, 1999.

[Fio98] Fiore, P. ***Everyone is talking about data warehousing***, Evolving Enterprise, 1998. Disponível em: <http://www.lionhrtpub.com/ee/ee-spring98/fiore.html>. Acesso em 11/10/2002.

[FR98] Fernandez, J.; Ramamritham, K. ***Adaptive Dissemination of Data in Real-Time Asymmetric Communication Environments***, submitted for publication, 1998.

[Fur73] Furtado, A. L. ***Teoria dos grafos: algoritmos***. Ed. USP, 168p, 1973

[GBL+96] Gray, J.; Bosworth, A.; Layman, A.; Pirahesh, H. ***Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tabs, and Sub-Totals.*** Proceedings of ICDE '96, New Orleans, February 1996.

[GHQ95] Gupta, A.; Harinarayan, V.; Quass, D. ***Aggregate-Query Processing in a Data Warehousing Environments***. In *Proceedings of th 21st International VLDB Conference*, pages 359-369, 1995.

[Gig01] Giguère, E. ***Mobile Data Management: Challenges of Wireless and Offline Data Access.*** In Proceedings of the 17th International Conference on Data Engineering, 2001.

[GJM97] Gupta, A.; Jagadish, H. V.; Mumick, I. S. ***Maintenance and self-maintenance of outerjoin views.*** In Next Generation Information Technology and Systems, 1997.

[GM99] Gupta, H.; Mumick, I. S. ***Incremental Maintenance of Aggregate and Outerjoin Expressions***. Techincal Report, Stanford University, 1999.

[HAR96] Harinarayan, V.; Rajaraman, A.; Ullman, J. D. ***Implementing Data Cubes Efficiently***. SIGMOD Conference, 1996.

[HH01] Hasan, H.; Hyland, P. ***Using OLAP and multidimensional data for decision making***. IT Professional , vol: 3, Set.-Out. 2001 Page(s): 44 -50.

[HHD99] Humphries, M.; Hawkins, M. W.; Dy, M. C. ***Data warehousing- Architecture and Implementation***, New Jersey: Prentice Hall, 1999.

[HKZ02] Helal, S.; Khushraj, A.; Zhang, J. ***Incremental Hoarding and Reintegration in Mobile Environments***. 2002 Symposium on Applications and the Internet (SAINT). Janeiro 2002.

[INM96] Inmon, W. H. ***Building the data warehouse***. New York: John Wiley & Sons, 1996. 401 p.

[JPB+02] Júnior O. de G. F.; Pacheco, R. C. S.; Barbosa D. M.; Todesco, J. L. ***Abordando o uso da Orientação a Objetos em um Sistema de Data Warehouse***. II Congresso Brasileiro de Computação – CBComp 2002.

[KAM93] Kamal, A. E. ***Modelling multi-bus interconnection networks using state aggregation***. Computer Systems Science and Engineering, 8(1):57-63, January 1993.

[KIM96] Kimball, R. ***The Data Warehouse Toolkit***. John Wiley & Sons,1996

[KIM98] Kimball, R, et al. ***The Data Warehouse Lifecicle Toolkit: expert methods for designing, developing, and deploying data warehouses.*** New York: John Wiley & Sons, 1998.

[KS99] Korth, H. F.; Silberschatz, A. ***Sistemas de Bancos de Dados***. 3. ed. São Paulo: Makron Books, 1999.

[LHK99] Lim, J. B.; Hurson, A. R.; Kavi, K. M.: ***Concurrent Data Access in Mobile Heterogeneous Systems***. HICSS 1999.

[LLS02] Lee, K.C.K.; Va Leong, H.; Si, A ***Semantic data access in an asymmetric mobile environment Mobile Data Management***, 2002. Proceedings. Third International Conference, 2002.

[LSV98] Lee, K. C. K.; Si, A.; Va Leong, H. ***Incremental view update for a mobile data warehouse***. SAC 1998: 394-399.

[LYC+99] Labio, W. J.; Yang J.; Cui, Y.; Garcia-Molina, H. ***Performance Issues in Incremental Warehouse Maintenance***. Techincal Report, Stanford University, 1999.

[LYL+02] Lee, G.; Yeh, M. S.; Lo, S. C.; Chen, A. L. P. ***A Strategy for Efficient Access of Multiple Data Items in Mobile Environments***. Proc. 3rd International Conference on Mobile Data Management, pp. 71-78, Singapore, January 2002.

[Moe97] Moerkotte, G. ***Small Materialized Aggregates: A Light Weight Index Structure***. In: Very Large Data Bases Conference (VLDB´98), 24, 1998, New York City, New York, USA. 1997. p. 476-487.

[Mfa96] McFadden, F. R. ***Data Warehouse for EIS: Some Issues and Impacts***. HICSS (2) 1996: 120-129.

[MQM97] Mumick, I. S.; Quass, D.; Mumick B. S. ***Maintenance of Data Cubes and Summary Tables in a Warehouse***. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, 1997, pp. 100-111.

[MSR+99] Mohania M. K.; Samtani S.; Roddick J. F.; Kambayashi, Y. ***Advances and Research Directions in Data Warehousing Technology***, Australian Journal of Information Systems, 1999.

[OG95] O´Neill, P.; Graefe, G. ***Multi-Table Joins Through Bitmapped Join Indexes***. In *SIGMOD Record*, pages 311-320, 1995.

[OHP00] Oh, J.; Hua K. A.; Prabhakara, K. ***A New Broadcasting Technique for An Adaptive Hybrid Data Delivery in Wireless Mobile Network Environment.*** *Proc. of 19th IEEE International Performance, Computing and Communications Conference*. pp. 361-367. Fev. 20 - 22, 2000. Phoenix, Arizona.

[Ora02] ***Banco de dados Oracle 9i***. Disponível em : <http://www.oracle.com>. Acesso em 12/07/2002.

[PAK+02] ***Pacific Asia Conference on Knolwedge Discovery and Data Mining 2002***, Workshop. Disponível em <http://www.mathcs.sjsu.edu/faculty/tylin/ pakdd_workshop.html>. Acesso em 25/11/2002.

[PB94] Pitoura, E.; Bhargava, B. ***Building Information Systems for Mobile Environments.*** 3rd ACM International Conference on Information and Knowledge Management (CIKM94), pp. 371-378, Novembro 1994.

[PC99] Pitoura, E.; Chrysanthis, P. K. ***Scalable processing of read-only transactions in broadcast push***. In Proc. of the 19th IEEE Int'l Conference on Distributed Computing Systems, pages 432-441, Junho 1999.

[POE98] Poe, V.; Klauer, P.; Brobst, S. ***Building a Data Warehouse for Decision Suport.*** 2nd edição. New Jersey: Prentice Hall PTR. 1998. 285p.

[Qua97] Quass, D. ***Materialized Views in Data Warehouses***. PhD thesis, Stanford University, Department of Computer Science, 1997.

[SAE+99] Stanoi, I.; Agrawal, D.; El Abbadi, A.; Phatak, S.; H., Badrinath B. R. ***Data Warehousing Alternatives for Mobile Environments***. MobiDE 1999: 110-115.

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

[ST97] Su, C.; Tassiulas, L. ***Broadcast Scheduling for Information Distribution***. INFOCOM 1997: 109-117.

[TV00] Trecordi, V.; Verticale, G. ***An Architecture for Effective Push/Pull Web Surfing***. ICC (2) 2000: 1159-1163.

[TS97] Theodoratos, D.; Sellis, T. K. ***Data Warehouse Configuration***. VLDB 1997: 126-135.

[WB98] Welling, G.; Badrinath, B. R. ***An Architecture for Exporting Environment Awareness to Mobile Compuing Applications***. IEEE Transactions on Softwrae Engineering, Vol 24, No. 5, May 1998.

[WEL96] Weldon, J. L. ***Choosing Tools for Multidimensional Data***. ***Database Programming and Design***, V.9, N.2, fevereiro 1996.

[WGL+96] Wiener, J.L.; Gupta, H.; Labio, W.J.; Zhuge, Y.; Garcia-Molina, H.; Widom J. ***A system prototype for warehouse view maintenance***. In The Workshop on Materialized Views, pages 26-33, Montreal, Canada, June 1996.

[Wid95a] Widom, J. ***Data Engineering, Special Issue on Materialized Views and Data Warehousing***, volume 18(2). IEEE, 1995.

[Wid95b] Widom, J. ***Research Problems in Data Warehousing***. Proceedings of the Fourth International Conference on Information and Knowledge Management (CIKM '95), pages 25-30, Baltimore, Maryland, November 1995.

[WXL+01] Wei, X.; Xiaofei, X.; Lei, S.; Quanlong, L.; Hao, L. ***Business intelligence based group decision support system***. Info-tech and Info-net, 2001. Volume: 5 , pp 295 -300.

Apêndice A

Implementações no servidor proxy

A.1 Tabela DW_CONTROL

```
CREATE TABLE DW_CONTROL
    ( codigo              number(6)
                  CONSTRAINT  codigo_dw_control_null NOT NULL,
      lcv                 number(12),
                 CONSTRAINT    dw_control_pk  PRIMARY KEY (codigo))
```

A.2 Tabela LCV_MAX

```
CREATE TABLE LCV_MAX
    ( lcv                 number(12)   CONSTRAINT  lcv_max_null NOT NULL)
```

A.3 Tabela *nós (*Estrutura de dados *Nós)*

```
CREATE TYPE  filho_tipo as varray(10) of number(3)

CREATE TYPE atualiza_tipo as object (tabela_delta_sumario varchar2(20),
                                     tabela_delta_sumario_base varchar2(20),
                                     insere_delta_sumario  varchar2(800),
                                     exclua_cache_sumario_minimo  varchar2(800),
                                     insere_cache_sumario_minimo  varchar2(800),
                                     exclua_cache_sumario_maximo  varchar2(800),
                                     insere_cache_sumario_maximo  varchar2(800))


CREATE TYPE atualiza_array as table of atualiza_tipo

CREATE TYPE nos_tipo as object (chave  number(3),filho  filho_tipo,
                                        atualiza atualiza_array)

CREATE TABLE nos of nos_tipo  nested table atualiza store as atualiza_tab
```

A.4 Tabela Ordem

```
CREATE TABLE ORDEM
    ( no_ordem          number(3)     CONSTRAINT  ordem_null NOT NULL)
```

A.5 Procedure preparação

```
CREATE OR REPLACE PROCEDURE preparacao(vsize in number) IS
    v_lcv_min       number(12);
    v_lcv_max       number(12);
    begin
        /* Criação do sd_log */
        execute immediate 'delete from SD_LOG';
        execute immediate 'insert into SD_LOG  (select * from log_ins
                            union all select * from log_del)';
        /* Atualização do Cache Centraliado */
        atualiza_cache(vsize);
        reconstrucao_cache(vsize);

        /* Definição da ordem de atualização das tabelas delta e cache-sumário */
        delete from ordem;
        ordenacao(1);

        /* Definição da menor versão disponível no proxy */
        select nvl(min(lcv),0) into v_lcv_min from DW_CONTROL;

        /* Atualização da tabela delta-sumário e cache-sumário */
        propaga_nos(v_lcv_min);

        /* Definição da maior versão do sd_log */
        select max(lcv) into v_lcv_max from sd_log;

        /* Atualiza o lcv_max do servidor proxy */
        if not (v_lcv_max is null) then
            execute immediate 'Update lcv_max set lcv = ' || v_lcv_max ;
        end if;

        /* Eliminação das tuplas dos logs utilizadas */
        execute immediate 'Delete from log_ins where lcv <= ' || v_lcv_max ;
        execute immediate 'Delete from log_del where lcv <= ' || v_lcv_max ;
        commit;
        exception
            when OTHERS THEN
                rollback;
end;
```

A.6 Procedure ordenação

```
CREATE OR REPLACE procedure ordenacao(idx in number) is
    v_chave     number(3);
    v_filho             filho_tipo;
    v_qtd       number(3);
    begin
        /* se existem filhos, a varredura continua */
        if not (idx is null) then
            select chave, filho into v_chave, v_filho from nos where chave = idx;
            if  (v_filho IS NOT NULL) and  (v_chave IS NOT NULL) then
                for i in 1..v_filho.count loop
                        ordenacao(v_filho(i));
                end loop;
            end if;
        end if;
        /* Verificar se já está na tabela ordem */
        select count(*) into v_qtd from ordem where NO_ORDEM = idx;
        if (v_qtd = 0) then
            insert into ORDEM (NO_ORDEM ) values (idx);
        end if;
    end;
```

A.7 Procedure propaga_nos

```
CREATE OR REPLACE PROCEDURE propaga_nos(vlcv in number) is
  tamanho                                 number(12);
  tamanho_base                            number(12);
  v_chave                                 number(3);
  v_no_ordem                              number(4);
  v_ordem                                 number(4);
  v_filho                                 filho_tipo;
  v_atualiza                              atualiza_array;
  v_tabela_delta_sumario                  varchar2(20);
  v_tabela_delta_sumario_base             varchar2(20);
  v_insere_delta_sumario                  varchar2(800);
  v_exclua_cache_sumario_minimo           varchar2(800);
  v_insere_cache_sumario_minimo           varchar2(800);
  v_exclua_cache_sumario_maximo           varchar2(800);
  v_insere_cache_sumario_maximo           varchar2(800);
  cursor cursor_ordem is select no_ordem from ordem order by rownum desc;
  begin
     open cursor_ordem;
     tamanho:=0;
     v_ordem:=1;
     loop
```

```
        fetch cursor_ordem into v_no_ordem;
        exit when cursor_ordem%NOTFOUND;
        select chave, filho, atualiza into v_chave, v_filho, v_atualiza from nos where
                chave = v_no_ordem;
        tamanho_base := 0;

        /* Definição da tabela delta-sumário ancestral adequada */
        FOR i IN 1..v_atualiza.count LOOP
            execute immediate 'select count(*) as contagem from ' ||
                                v_atualiza(i).tabela_delta_sumario_base into tamanho';
            if tamanho_base = 0 or tamanho_base > tamanho then
                tamanho_base := tamanho;
                v_ordem := i;
            end if;
        END LOOP;

        /* Carregando as variáveis */
        v_tabela_delta_sumario := v_atualiza(v_ordem).tabela_delta_sumario;
        v_tabela_delta_sumario_base :=
v_atualiza(v_ordem).tabela_delta_sumario_base;
        v_insere_delta_sumario := v_atualiza(v_ordem).insere_delta_sumario;
        v_exclua_cache_sumario_minimo :=

    v_atualiza(v_ordem).exclua_cache_sumario_minimo;
        v_insere_cache_sumario_minimo :=

    v_atualiza(v_ordem).insere_cache_sumario_minimo;
        v_exclua_cache_sumario_maximo :=

    v_atualiza(v_ordem).exclua_cache_sumario_maximo;
        v_insere_cache_sumario_maximo :=

    v_atualiza(v_ordem).insere_cache_sumario_maximo;

        /* Inserindo dados nas tabelas delta-sumário */
        execute immediate v_insere_delta_sumario;

        /* Atualizando os cache-sumários mínimos */
        if (v_exclua_cache_sumario_minimo is not null) then
            execute immediate v_exclua_cache_sumario_minimo;
        end if;
        if (v_insere_cache_sumario_minimo is not null) then
            execute immediate v_insere_cache_sumario_minimo;
        end if;

        /* Atualizando os cache-sumários máximos */
        if (v_exclua_cache_sumario_maximo is not null) then
            execute immediate v_exclua_cache_sumario_maximo;
        end if;
        if (v_insere_cache_sumario_maximo is not null) then
```

```
            execute immediate v_insere_cache_sumario_maximo;
        end if;

        /* Eliminando tuplas das tabelas delta-sumário usadas por todos os DWM */
        execute immediate 'Delete From ' || v_tabela_delta_sumario ||' where lcv <='
                || vlcv;
    end loop;
end;
```

Apêndice B

Implementações na plataforma móvel

B.1 Tabela *COD*

```
CREATE TABLE COD
    ( codigo            number(6)     CONSTRAINT  cod_null NOT NULL)
```

B2 Tabela LCV

```
CREATE TABLE LCV
    ( lcv               number(12)   CONSTRAINT  lcv_null NOT NULL)
```

B.3 Tabela *nós_atualiza* (Estrutura de dados *nós_atualiza*)

```
CREATE TABLE nos_atualiza (chave number(3), filho   filho_tipo,
                           tabela_sumario                    varchar2(20),
                           tabela_sumario_exclusao_temp      varchar2(800),
                           tabela_sumario_insercao_temp      varchar2(800),
                           tabela_sumario_exclusao           varchar2(800),
                           tabela_sumario_insercao           varchar2(800))
```

B.4 Tabela *nós_busca* (Estrutura de dados *nós_busca*)

```
CREATE TABLE nos_busca (chave number(3), filho   filho_tipo,
                        tabela_delta_sumario                 varchar2(20),
                        tabela_delta_sumario_criacao         varchar2(800))
```

B.5 Atualiza.java

Protótipo usado para validar a arquitetura do MDWManager.

```
// Pacotes
import java.sql.*;
import java.awt.*;
import javax.sql.*;
import oracle.sql.ARRAY;
import java.lang.*;
import oracle.jdbc.*;
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

class Atualiza extends javax.swing.JFrame{
  JTextArea informar = new JTextArea(4, 40);
  Statement stmt,stmt1;
  Connection Connr,Connl;
  String url,url1,url2,url3;
  String nome_log,senha_log;
  int    v_lcv_max;
  int    ordem[] = new int [50];
  int    ordem_pos;

  public Atualiza() throws SQLException{
    super("Atualização de DW");
    setSize(400, 300);
    JPanel theMain  = new JPanel(new GridLayout(1,1));
    getContentPane().add("Center",theMain);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    JPanel pane = new JPanel();
    JLabel InformarLabel = new JLabel("Ocorrencias: ",SwingConstants.CENTER);
    informar.setLineWrap(true);
    informar.setWrapStyleWord(true);
    JScrollPane areaScrollPane = new JScrollPane(informar);
      areaScrollPane.setVerticalScrollBarPolicy(
                  JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
      areaScrollPane.setPreferredSize(new Dimension(200, 200));
      areaScrollPane.setBorder(
        BorderFactory.createCompoundBorder(
          BorderFactory.createCompoundBorder(
                    BorderFactory.createTitledBorder("Ocorrencias"),
                    BorderFactory.createEmptyBorder(1,1,1,1)),
          areaScrollPane.getBorder()));

    theMain.add(areaScrollPane);
    setContentPane(theMain);
    show();
  }
```

```
public boolean abrir(String nome,String senha) throws SQLException
{
   nome_log = nome;
   senha_log = senha;

  try {
    url = "jdbc:oracle:oci8:@banco_local";
    url1 = System.getProperty("JDBC_URL");
    if (url1 != null)
        url = url1;

    DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

    // Conectar banco de dados
    Connr =  DriverManager.getConnection (url, nome_log, senha_log);
    Connr.setAutoCommit(false);

    // Criar um Statement
    stmt = Connr.createStatement ();
    } catch (Exception e) {
      informar.append("Error: " + e.toString() + e.getMessage() + "\n");
      return false;
    }
    return true;
}

public void processo() throws SQLException
{
  int  v_lcv_proxy,v_lcv_dw,v_cod_dw;
  v_lcv_dw =0;
  v_cod_dw=0;
  v_lcv_proxy=0;

  // Obter versão local do DWM
  ResultSet ver_local = stmt.executeQuery ("select lcv from LCV");
  if (ver_local.next())
        v_lcv_dw = ver_local.getInt("lcv");
  else
  {
        JOptionPane.showMessageDialog(null,"Controle de versao local (lcv) vazio");
        System.exit(0);
  }
  ver_local.close();

  // Obter identificação local
  ResultSet cod_local = stmt.executeQuery ("select cod from CODIGO");

  if (cod_local.next())
    v_cod_dw = cod_local.getInt("cod");
  else
```

```
    {
      JOptionPane.showMessageDialog(null,"Identificacao do DW Movel vazia");
      System.exit(0);
    }
    cod_local.close();

    // Obter versão do Proxy
    ResultSet ver_proxy = stmt.executeQuery ("select nvl(min(lcv),0) as mlcv from
                                  dw_control@link_proxy where lcv >=0");
    if (ver_proxy.next()) v_lcv_proxy = ver_proxy.getInt("mlcv");
    ver_proxy.close();

    // Verificar a necessidade de reconstrução
    if (v_lcv_dw < v_lcv_proxy) {
      // Reconstrucao
      JOptionPane.showMessageDialog(null,"DW necessita ser reconstruido.");
      System.exit(0);
    }

    //  Obter dados do Proxy
    Obter_dados_proxy(0,v_lcv_dw);

    // Atualizar o DWM
    atualizacao(0);

    // Atualiza a nova versão do DWM
    stmt.executeUpdate ("Update lcv set lcv = " + v_lcv_max);

    // Atualiza a nova versão do DWM na tabela de controle no Proxy
(DW_CONTROL)//
    stmt.executeUpdate("Update dw_control@link_proxy set lcv = " + v_lcv_max + "
              where cod = " + v_cod_dw);

     // Commit
    Connr.commit();
    Connr.setAutoCommit(true);
  }

  public void Obter_dados_proxy(int id, int v_dw) throws SQLException{
   String dc,dt;
   Statement st;
   int qt,pos;
   ARRAY filhos;
   try {
     if (id >=0) {

       // Criar um Statement
       st = Connr.createStatement ();
       ResultSet ver_linha = stmt.executeQuery ("select * from nos_busca where
                                                          chave=" + id);
```

```
      if (ver_linha.next()){

       // Carregar variáveis
       filhos =  (ARRAY) ver_linha.getObject("filho");
       qt = filhos.length();
       dt = ver_linha.getString("tabela_delta_sumario");
       dc = ver_linha.getString("tabela_delta_sumario_criacao");

       // Excluir a tabela delta-sumário
       try {
         st.executeUpdate("drop table " + dt);
       } catch (Exception e) {}
       try {
         pos=dc.indexOf("?");
         // Executar a transferência de dados
         st.executeUpdate(dc.substring(0,pos-1)+ " " + v_dw + " " +
                          dc.substring(pos+1));
       } catch (Exception e) {}

       // Obter dados dos nós descendentes
       if (qt != 0) {
         int[] numfilhos = filhos.getIntArray();
         for (int j=0; j < numfilhos.length; j++)
            Obter_dados_proxy(numfilhos[j],v_dw);
       }
      }
     }
    } catch (Exception e) {
      System.out.println("Error: " + e.toString() + e.getMessage());
    }
  }  // Obter_dados_proxy


  // Atualizar tabelas sumarizadas
  public void atualizacao(int id) throws SQLException{
   String st, tab_sum, tab_sum_exc_temp, tab_sum_ins_temp, tab_sum_ins,
tab_sum_exc;
   ARRAY filhos;
   int qt;
   if (id >=0) {
      // Result linha
      st = "select * from nos_atualiza where chave=" + id;
      ResultSet ver_linha = stmt.executeQuery (st);
      if (ver_linha.next()){
        // Carregar variáveis
        filhos =  (ARRAY) ver_linha.getObject("filho");
        qt = filhos.length();
        tab_sum = ver_linha.getString("tabela_sumario");
       tab_sum_exc_temp = ver_linha.getString("tabela_sumario_exclusao_temp");
       tab_sum_ins_temp = ver_linha.getString("tabela_sumario_insercao_temp");
```

```
      tab_sum_exc = ver_linha.getString("tabela_sumario_exclusao");
      tab_sum_ins = ver_linha.getString("tabela_sumario_insercao");

       try{
         // Excluir tabela de exclusão temporária
         stmt.executeUpdate("drop table " + tab_sum + "_del");
       } catch (Exception e) {}

       // Criar tabela de exclusão temporária
       stmt.executeUpdate(tab_sum_exc_temp);

       try{
         // Excluir tabela de inserção temporária
         stmt.executeUpdate("drop table " + tab_sum + "_ins");
       } catch (Exception e) {}

       // Criar tabela de inclusão temporária
       stmt.executeUpdate(tab_sum_ins_temp);

      // Excluir dados da tabela sumarizada
       stmt.executeUpdate(tab_sum_exc);

      // Inserir dados na tabela sumarizada
       stmt.executeUpdate(tab_sum_ins);

       // Atualizar tabelas sumarizadas descendentes
       if (qt != 0) {
         int[] numfilhos = filhos.getIntArray();
         for (int j=0; j < numfilhos.length; j++)
            atualizacao(numfilhos[j]);
       }
     }
  }
 }



  // Função que verifica se DWM está atualizado
  public boolean isAtualizado() throws Exception  {
    int v_lcv_dw = 0;
    int v_lcv_proxy = 0;
    // Result set local
    try {
    ResultSet ver_local = stmt.executeQuery ("select lcv from LCV");
    if (ver_local.next())
       v_lcv_dw = ver_local.getInt("lcv");
    ResultSet ver_proxy = stmt.executeQuery ("select lcv from
LCV_MAX@link_proxy");
    if (ver_proxy.next())
       v_lcv_proxy = ver_proxy.getInt("lcv");
```

```
    } catch (Exception e) {return (1 = = 0);}
    return (v_lcv_dw = = v_lcv_proxy);
  }

  // Procedimento que verifica constantemente o nível de atualização do
  // servidor proxy em relação do DWM
  public void verificacao()  throws Exception{
    boolean  existe = false;
    String username,terminal_local,hora_inicio,hora_fim;
    int terminal_session,pos,lapso;
    ResultSet user_session,horario_session;
    int hour,minute;
    lapso=0;
    int resposta;
    int horai=0;
    int horaf=0;
    int minutoi=0;
    int minutof=0;
    boolean first_time=true;
    try{
     username = "";
     terminal_local = "";
     terminal_session = 0;
     ResultSet env_session = stmt.executeQuery ("select userenv('terminal') as
terminal,user
                                                    from dual");
     if (env_session.next()){
       terminal_local = env_session.getString("terminal");
       username = env_session.getString("user");
     }
     while (true){
       GregorianCalendar calendar = new GregorianCalendar();
       hour = calendar.get(Calendar.HOUR_OF_DAY) * 3600;
       minute =  calendar.get(Calendar.MINUTE) * 60;
       horario_session = stmt.executeQuery ("select * from horas");
       if (horario_session.next()){
         hora_inicio = horario_session.getString("hora_inicio");
         hora_fim = horario_session.getString("hora_fim");
         existe=true;
         try {
           pos=hora_inicio.indexOf(":");
           horai = Integer.parseInt(hora_inicio.substring(0,pos)) * 3600;
           minutoi =Integer.parseInt(hora_inicio.substring(pos+1))* 60;
           pos=hora_fim.indexOf(":");
           horaf = Integer.parseInt(hora_fim.substring(0,pos)) * 3600;
           minutof = Integer.parseInt(hora_fim.substring(pos+1))* 60;
           } catch (Exception e) {existe=false;}
       }
       if (existe){
         if ((horai+minutoi) > (hour + minute)){
```

```
        lapso = ((horai+minutoi) - (hour + minute)) * 1000;
       }
       else
        if ((horaf+minutof) < (hour + minute)){
         lapso = (89940 - (hour + minute)) * 1000;
        }
        else
         lapso = 20000;
      }
      else{
       if (!first_time)
        lapso = 20000;
       first_time=false;
      }
      user_session = stmt.executeQuery ("select nvl(COUNT(*),0) as num_session
        from V$SESSION where username= '" + username + "' and terminal= '" +
                        terminal_local + "'");
      if (user_session.next()){
       terminal_session = user_session.getInt("num_session");
      }
      try {
       Thread.sleep(lapso);
      } catch (InterruptedException e) {}
      if (!(isAtualizado())){
       if (terminal_session > 1) {
        informar.append("DW desatualizado, porém usuário está em uma outra
sessão \n");
       }
       else
       {
        resposta =0;
        if (! existe)
         resposta = JOptionPane.showConfirmDialog(null,"Deseja atualizar o
                        DW?","Atualização do
                        DW",JOptionPane.YES_NO_OPTION,
                        JOptionPane.QUESTION_MESSAGE);
        if (resposta == 0){
          processo();
        }
       }
      }
     }
   } catch (Exception e) {      System.out.println("Error: " + e.toString() +
e.getMessage());}

  }

  public static void main (String args []) throws Exception  {
    int NumError;
    boolean aberto;
```

```
    NumError=0;
    String username;
    String password;
    aberto = (1 = = 0);
   try {
     ExitWindow exit = new ExitWindow();
     Atualiza atualiza = new Atualiza();
     atualiza.addWindowListener(exit);
     while ((NumError < 3) & (! aberto )){
       username = JOptionPane.showInputDialog("Entre o nome do usuário");
       password = JOptionPane.showInputDialog("Entre a senha do usuário");
       aberto = atualiza.abrir(username,password);
       NumError++;
     }
     if ((NumError > 3) | (! aberto))
       System.exit(0);
     atualiza.verificacao();
     atualiza.pack();
     atualiza.setVisible(true);
   } catch (Exception e) {}
  }

}

  class ExitWindow extends WindowAdapter {
     public void windowClosing(WindowEvent e){
       System.exit(0);
     }
  }
```

Apêndice C

Exemplo de um DWM

C.1 Tabela das fontes de dados

Vendas(venda_id, produto, loja, data, qtd, preço),

C.2 Tabelas de dimensão

Produtos (produto, nome_produto, categoria, nome_categoria);

Lojas (loja, nome_loja, cidade, nome_cidade, região, nome_região);

Tempos (dia, data, mês).

C.3 Visões materializadas

```
/* Visão PLD_VENDAS
CREATE VIEW PLD_VENDAS(produto, loja, dia, qtd, receita, contador,
            min_qtd, max_qtd) AS
SELECT produto, loja, dia, sum(qtd), sum(qtd * preço), count(*),
            min(qtd), max(qtd)
        FROM vendas, tempos
        WHERE vendas.data = tempos.data
        GROUP BY produto, loja, dia

/* Visão PLM_VENDAS
CREATE VIEW PLM_VENDAS(produto, loja, mês, qtd,  receita, contador,
        min_qtd, max_qtd) AS
SELECT produto, loja, mês, sum(qtd), sum(receita), sum(contador),
            min(min_qtd), max(max_qtd)
        FROM PLD_vendas, tempos
        WHERE PLD_vendas.dia = tempos.dia
        GROUP BY produto, loja, mês

/* Visão PCD_VENDAS
CREATE VIEW PCD_VENDAS(produto, cidade, dia, qtd, receita, contador,
        min_qtd, max_qtd) AS
```

```
SELECT produto, cidade, dia, sum(qtd), sum(receita),  sum(contador),
            min(min_qtd), max(max_qtd)
        FROM PLD_Vendas, lojas
        WHERE PLD_vendas.loja = lojas.loja
        GROUP BY produto, cidade, dia

/* Visão CatR_VENDAS

CREATE VIEW CatR_VENDAS(categoria, região, qtd, receita, contador,
            min_qtd, max_qtd) AS
SELECT categoria, região, sum(qtd), sum(receita),sum(contador),
            min(min_qtd), max(max_qtd)
        FROM PLM_vendas, lojas, produtos
        WHERE PLM_vendas.loja = lojas.loja
                    and PLM_vendas.produto = Produtos.produto
        GROUP BY categoria, região
```

C.4 Caches centralizados

PLD_VENDAS_MINCACHE(produto, loja, dia, qtd, contador) e;

PLD_VENDAS_MAXCACHE(produto, loja, dia, qtd, contador).

C.5 Tabelas Delta-sumários

SD_PLD_VENDAS(produto, loja, dia, qtd, receita, contador,lcv);

SD_PCD_VENDAS(produto,cidade,dia, qtd, receita,contador,lcv);

SD_PLM_VENDAS(produto, loja, mês, qtd, receita, contador,lcv);

SD_CatR_VENDAS(categoria, região, qtd, receita, contador,lcv).

C.6 Tabelas Cache-sumários

PLD_VENDAS_MIN(produto, loja, dia, qtd);

PLD_VENDAS_MAX(produto, loja, dia, qtd);

PCD_VENDAS_MIN(produto, cidade, dia, qtd);

PCD_VENDAS_MAX(produto, cidade, dia, qtd);

PLM_VENDAS_MIN(produto, loja, mês, qtd);

PLM_VENDAS_MAX(produto, loja, mês, qtd);

CatR_VENDAS_MIN(categoria, região, qtd);

CatR_VENDAS_MAX(categoria, região, qtd).

C.7 Procedure atualiza_cache

Exemplo de atualização do Cache centralizado de mínimo e máximo (PLD_VENDAS_MIN e PLD_VENDAS_MAX)

```
CREATE OR REPLACE PROCEDURE atualiza_cache (VSIZE IN NUMBER) IS
begin
    execute immediate 'drop table MIN_TEMP';
    execute immediate 'drop table MAX_TEMP';
    execute immediate 'drop table MAX_DEL';
    execute immediate 'drop table MIN_DEL';
    execute immediate 'drop table MIN_INS';
    execute immediate 'drop table MAX_INS';
    execute immediate ' CREATE TABLE MIN_TEMP as
        (SELECT a.produto, a.loja, a.dia, a.qtd, a.contador, a.lcv
        from (select produto,loja,dia,abs(qtd) as qtd,sum(contador) as contador,
                max(lcv) as lcv from sd_log group by produto,loja,dia,abs(qtd)) a
                where '  ||  vsize ||  ' >= (select sum(contador) from sd_log d
                        where a.produto=d.produto and a.loja=d.loja and a.dia=d.dia
                                and abs(d.qtd)<=abs(a.qtd)))';

    execute immediate ' CREATE TABLE MAX_TEMP as
        (SELECT a.produto, a.loja, a.dia, a.qtd, a.contador, a.lcv
        from (select produto,loja,dia,abs(qtd) as qtd,sum(contador) as contador,
                max(lcv) as lcv from sd_log group by produto,loja,dia,abs(qtd)) a
                where  ' ||  vsize || '  >= (select sum(contador) from sd_log d
                        where a.produto=d.produto and a.loja=d.loja and a.dia=d.dia
                                and abs(d.qtd)>=abs(a.qtd)))';

    execute immediate 'CREATE TABLE MIN_DEL AS
        (SELECT a.produto,a.loja,a.dia,a.qtd,a.contador,a.lcv
                from (select produto,loja,dia,qtd,contador,lcv from min_temp) a
        left outer join
                (SELECT produto,loja,dia,max(qtd) AS MAX_V FROM
                        PLD_VENDAS_MINCACHE
                where (produto,loja,dia) in (select produto,loja,dia from min_temp)
                GROUP BY produto,loja,dia) b
        on a.produto=b.produto AND a.loja=b.loja AND a.dia=b.dia
        where b.MAX_v>=a.qtd or b.max_v is null)';

    execute immediate 'CREATE TABLE MAX_DEL AS
        (SELECT a.produto,a.loja,a.dia,a.qtd,a.contador,a.lcv
                from (select produto,loja,dia,qtd,contador,lcv from max_temp) a
        left outer join
                (SELECT produto,loja,dia,min(qtd) AS MIN_V FROM
                        PLD_VENDAS_MAXCACHE
                where (produto,loja,dia) in (select produto,loja,dia from max_temp)
```

```
            GROUP BY produto,loja,dia) b
        on a.produto=b.produto AND a.loja=b.loja AND a.dia=b.dia
        where b.MIN_v<=a.qtd or b.min_v is null)';

    execute immediate 'CREATE TABLE MIN_INS AS
        (select produto,loja,dia,qtd,sum(contador) as contador,MAX(LCV) AS LCV
            from ((select * from MIN_DEL) union all
                (select * from PLD_VENDAS_MINCACHE
                where (produto,loja,dia,qtd)
                IN (select produto,loja,dia,qtd from min_del)))
            group by produto,loja,dia,qtd
            having sum(contador)>0);

    execute immediate 'CREATE TABLE MAX_INS AS
        (select produto,loja,dia,qtd,sum(contador) as contador,MAX(LCV) AS LCV
            from ((select * from MAX_DEL) union all
                (select * from PLD_VENDAS_MAXCACHE
                where (produto,loja,dia,qtd)
                IN (select produto,loja,dia,qtd from max_del)))
            group by produto,loja,dia,qtd
            having sum(contador)>0);

    execute immediate 'delete from pld_vendas_mincache where
        (produto,loja,dia,qtd) in (select produto,loja,dia,qtd from min_del)';

    execute immediate 'delete from pld_vendas_maxcache where
        (produto,loja,dia,qtd) in (select produto,loja,dia,qtd from max_del)';

    execute immediate 'insert into pld_vendas_mincache (select * from min_ins)';
    execute immediate 'insert into pld_vendas_maxcache (select * from max_ins)';
end;
```

C.8 Estrutura de dados *nós*

```
    Id = 1;
    Filho(1) = 2
    Filho(2) = 3
    Atualiza(1) = Delta_sumario = 'SD_PLD_VENDAS'
    Delta_sumario_base = null
    Insere_delta_sumario = ' insert into sd_pld_vendas (produto,loja,dia,
                    qtd, receita, contador, lcv)
                    (select produto,loja,dia,sum(qtd) as qtd,
                    sum(receita) as receita, sum(contador) as contador,
                    max(lcv) as lcv from sd_log group by produto,loja,dia)'

    Exclua_cache_sumario_mínimo = ' delete from pld_vendas_min
                    where (produto,loja,dia) in (Select produto,loja,dia
                    From sd_pld_vendas WHERE  lcv >=
                    (select min(lcv) from sd_log) group by produto,loja,dia)'
```

```
Insere_cache_sumario_mínimo = ' insert into pld_vendas_min
                    (Select produto, loja, dia, min(qtd) as qtd
                    From PLD_Vendas_Mincache Where (produto, loja, dia)
                    in (select produto,loja,dia from sd_pld_vendas
                            WHERE lcv >= (select min(lcv) from sd_log))
                    Group by produto, loja, dia)'

Exclua_cache_sumario_maximo = ' delete from pld_vendas_max
                    where (produto,loja,dia) in (Select produto,loja,dia
                    From sd_pld_vendas WHERE  lcv >=
                    (select min(lcv) from sd_log) group by produto,loja,dia)'

Insere_cache_sumario_maximo = ' insert into pld_vendas_max
                (Select produto, loja, dia, min(qtd) as qtd
                From PLD_Vendas_Maxcache Where (produto, loja, dia)
                        in (select produto,loja,dia from sd_pld_vendas
                                WHERE lcv >= (select min(lcv) from sd_log))
                        Group by produto, loja, dia)',

Id = 2;
Filho(1) = 4
Atualiza(1) = Delta_sumario = 'SD_PLM_VENDAS'
Delta_sumario_base = 'SD_PLD_VENDAS'
Insere_delta_sumario = ' insert into sd_plm_vendas (produto,loja,mês,qtd,
                receita, contador, lcv)  (select produto,loja,mês,sum(qtd) as
                qtd, sum(receita) as receita, sum(contador) as contador,
                max(lcv) as lcv from sd_pld_vendas a, tempos b
                where a.dia = b.dia and a.lcv > = (select min(lcv)
                from sd_log) group by a.produto,a.loja,b.mês)'

Exclua_cache_sumario_mínimo = ' delete from plm_vendas_min
                    where (produto,loja,mês) in (Select produto,loja,mês
                    From sd_plm_vendas WHERE  lcv >= (select min(lcv)
                            from sd_log) group by produto,loja,mês)'

Insere_cache_sumario_mínimo = ' insert into plm_vendas_min
            (Select g.produto, g.loja, h.mês, min(g.qtd) as qtd,max(g.lcv) as lcv
                    From pld_vendas_min g,  tempos h  Where g.dia = h.dia  and
                    (g.produto, g.loja, h.mês) in (select produto,loja,mês from
                    sd_plm_vendas WHERE lcv >= (select min(lcv) from sd_log))
                    Group by g.produto, g.loja, h.mês)'

Exclua_cache_sumario_maximo = ' delete from plm_vendas_max
                    where (produto,loja,mês) in (Select produto,loja,mês
                    From sd_plm_vendas WHERE  lcv >= (select min(lcv)
                            from sd_log) group by produto,loja,mês)'

Insere_cache_sumario_maximo = ' insert into plm_vendas_max
            (Select g.produto, g.loja, h.mês, max(g.qtd) as qtd,max(g.lcv) as lcv
                    From pld_vendas_max g,  tempos h Where g.dia = h.dia  and
```

```
                (g.produto, g.loja, h.mês) in (select produto,loja,mês from
                sd_plm_vendas WHERE lcv >= (select min(lcv) from sd_log))
                group by g.produto, g.loja, h.mês)',

Id = 4;
Filhos = null
Atualiza(1) = Delta_sumario = 'SD_CATR_VENDAS'
Delta_sumario_base = 'SD_PLM_VENDAS'
Insere_delta_sumario = ' insert into sd_catr_vendas (categoria,regiao,qtd,
                receita,contador, lcv)
                (select c.categoria,b.regiao,sum(qtd) as qtd, sum(receita) as
                receita,sum(contador) as contador, max(lcv) as lcv
                from sd_plm_vendas a, lojas b, produtos c
                 where a.loja = b.loja and a.produto = c.produto and a.lcv > =
                (select min(lcv) from sd_log) group by c.categoria,b.regiao)'

 Exclua_cache_sumario_mínimo = ' delete from catr_vendas_min
                        where (categoria, regiao) in (Select categoria, regiao
                        From sd_catr_vendas WHERE  lcv >= (select min(lcv)
                        from sd_log) group by categoria, regiao)'

Insere_cache_sumario_mínimo = ' insert into catr_vendas_min
                (Select p.categoria, j.regiao, min(g.qtd) as qtd,max(g.lcv) as lcv
                 From plm_vendas_min g,  lojas j, produto p Where g.loja = j.loja
                and g.produto = h.produto  and (p.categoria, j.regiao) in
                (select p.categoria, j.regiao from sd_catr_vendas
                WHERE lcv >= (select min(lcv) from sd_log))
                Group by p.categoria, j.regiao)'

Exclua_cache_sumario_maximo = ' delete from catr_vendas_max
                        where (categoria, regiao) in (Select categoria, regiao
                        From sd_catr_vendas WHERE  lcv >= (select min(lcv)
                        from sd_log)  group by categoria, regiao)'

Insere_cache_sumario_maximo = ' insert into catr_vendas_max
        (Select p.categoria, j.regiao, max(g.qtd) as qtd,max(g.lcv) as lcv
         From plm_vendas_max g,  lojas j, produto p Where g.loja = j.loja
        and g.produto = h.produto  and (p.categoria, j.regiao) in
        (select p.categoria, j.regiao from sd_catr_vendas WHERE lcv >=
                (select min(lcv) from sd_log)) Group by p.categoria, j.regiao)'

Id = 3;
Filhos = null
Atualiza(1) = Delta_sumario = 'SD_PCD_VENDAS'
Delta_sumario_base = 'SD_PLD_VENDAS'
Insere_delta_sumario = ' insert into sd_pcd_vendas (produto,cidade,dia,
                        qtd, receita, contador, lcv)
                        (select produto,b.cidade,dia,sum(qtd) as qtd, sum(receita)
                        as receita,sum(contador) as contador,
                        max(lcv) as lcv from sd_pld_vendas a, lojas b
```

```
                    where a.loja = b.loja and a.lcv >= (select min(lcv)
                        from sd_log) group by a.produto,b.cidade,a.dia)'

 Exclua_cache_sumario_mínimo = ' delete from pcd_vendas_min
                    where (produto,cidade,dia) in (Select produto,cidade,dia
                    From sd_pcd_vendas WHERE  lcv >= (select min(lcv)
                    from sd_log) group by produto,cidade,dia)'

Insere_cache_sumario_mínimo = ' insert into pcd_vendas_min
        (Select a.produto, b.cidade, a.dia, min(a.qtd) as qtd, max(a.lcv) as lcv
        From pld_vendas_min a, lojas b Where a.loja = b.loja and
            (a.produto, b.cidade, a.dia) in (select produto,cidade,dia from
            sd_pcd_vendas WHERE lcv >= (select min(lcv) from sd_log))
                Group by a.product,b.cidade,a.dia)'

Exclua_cache_sumario_maximo = ' delete from pcd_vendas_max
                    where (produto,cidade,dia) in (Select produto,cidade,dia
                    From sd_pcd_vendas WHERE  lcv >= (select min(lcv)
                    from sd_log) group by produto,cidade,dia)'

Insere_cache_sumario_maximo = ' insert into pcd_vendas_max
   (Select a.produto, b.cidade, a.dia, max(a.qtd) as qtd, max(a.lcv) as lcv
            From pld_vendas_max a, lojas b Where a.loja = b.loja and
        (a.produto, b.cidade, a.dia) in (select produto,cidade,dia from
        sd_pcd_vendas WHERE lcv >= (select min(lcv) from sd_log))
            Group by a.product,b.cidade,a.dia)'
```

C.9 Estrutura de dados *nós*_atualiza

```
Id = 1;
Filho(1) = 2
Filho(2) = 3
Tabela_sumario = 'PLD_VENDAS'
Tabela_sumario_exclusão_temp = ' CREATE TABLE PLD_DEL  AS
            (Select * FROM pld_vendas where (produto,loja,dia) IN
                (select produto,loja,dia from sd_pld_vendas))'

Tabela_sumario_inserção_temp = 'CREATE TABLE  PLD_INS AS
            (select produto, loja, dia, qtd, receita, min_qtd, max_qtd, count
                from (select produto,loja,dia,sum(qtd)  as qtd,
                    sum(receita) as receita, min(min_qtd) as min_qtd
                    max(max_qtd) as max_qtd, sum(count) as count
                    from ((select produto,loja,dia, qtd, receita,
                        null as min_qtd, null as max_qtd, count,
                        from PLD_DEL)
                union all
                    (select produto,loja,dia, qtd, receita, min_qtd,
                    max_qtd, count   from sd_pld_vendas))
```

```
                    group by produto,loja,dia)
                    having count>0)'

Tabela_sumario_exclusão = ' DELETE FROM PLD_VENDAS
WHERE (produto,loja,dia) IN (SELECT produto,loja,dia FROM PLD_DEL)'

Tabela_sumario_inserção = 'INSERT INTO PLD_VENDAS
                                (SELECT *  FROM PLD_INS)'

Id = 2;
Filho(1) = 4
Tabela_sumario = 'PLM_VENDAS '
Tabela_sumario_exclusão_temp = ' CREATE TABLE PLM_DEL  AS
            (Select * FROM plm_vendas where (produto,loja,mês) IN
                    (select produto,loja,mês from sd_plm_vendas))'

Tabela_sumario_inserção_temp = 'CREATE TABLE  PLM_INS AS
      (select produto, loja, mês, qtd, receita, min_qtd, max_qtd, count
                    from (select produto,loja,mês,sum(qtd)  as qtd,
                            sum(receita) as receita, min(min_qtd) as min_qtd
                    max(max_qtd) as max_qtd, sum(count) as count
                            from ((select produto,loja,mês, qtd, receita,
                                    null as min_qtd, null as max_qtd, count,
                                    from PLM_DEL)
                    union all
                            (select produto,loja,mês,qtd, receita, min_qtd,
                            max_qtd, count from sd_plm_vendas))
                    group by produto,loja,mês)
                    having sum(count)>0)'

Tabela_sumario_exclusão = ' DELETE FROM PLM_VENDAS   WHERE
      (produto,loja,mês) IN (SELECT produto,loja,mês FROM PLM_DEL)'

Tabela_sumario_inserção = 'INSERT INTO PLM_VENDAS
                                (SELECT *  FROM PLM_INS)'


Id = 4;
Filhos = null
Tabela_sumario = 'CatR_VENDAS'
Tabela_sumario_exclusão_temp = ' CREATE TABLE CATR_DEL  AS
            (Select * FROM catr_vendas where (categoria, região) IN
                    (select categoria, região from sd_catr_vendas))'

Tabela_sumario_inserção_temp = 'CREATE TABLE  CATR_INS AS
      (select categoria, região, qtd, receita, min_qtd, max_qtd, count
            from (select categoria, região,sum(qtd)  as qtd, sum(receita) as
             receita, min(min_qtd) as min_qtd, max(max_qtd) as max_qtd,
            sum(count) as count
                            from ((select categoria, região, qtd, receita,
```

```
                                    null as min_qtd, null as max_qtd, count,
                                        from CATR_DEL)
                        union all
                            (select categoria, região, qtd, receita, min_qtd,
                            max_qtd, count from sd_catr_vendas))
                        group by categoria, região)
                        having sum(count)>0)'

Tabela_sumario_exclusão = ' DELETE FROM CATR_VENDAS WHERE
        (categoria, região) IN (SELECT categoria, região FROM CATR_DEL)'

Tabela_sumario_inserção='INSERT INTO CatR_VENDAS
                                    (SELECT *  FROM CatR_INS)'


Id = 3;
Filhos = null
Tabela_sumario = 'PCD_VENDAS'
Tabela_sumario_exclusão_temp = ' CREATE TABLE PCD_DEL  AS
                (Select * FROM pcd_vendas where (produto,cidade,dia) IN
                        (select produto,cidade,dia from sd_pcd_vendas))'

Tabela_sumario_inserção_temp = '
        CREATE TABLE  PCD_INS AS
        (select produto, cidade, dia, qtd, receita, min_qtd, max_qtd, count
                        from (select produto,cidade,dia,sum(qtd)  as qtd,
                                sum(receita) as receita, min(min_qtd) as min_qtd
                                max(max_qtd) as max_qtd, sum(count) as count,
                                from ((select produto,cidade,dia, qtd, receita,
                                        null as min_qtd, null as max_qtd, count,
                                            from PCD_DEL)
                        union all
                              (select produto,cidade,dia, qtd, receita, min_qtd,
                                max_qtd, count from sd_pcd_vendas))
                        group by produto,cidade,dia)
                        having sum(count)>0)'

Tabela_sumario_exclusão = ' DELETE FROM PCD_VENDAS  WHERE
        (produto,cidade,dia) IN (SELECT produto,cidade,dia FROM PCD_DEL)'

Tabela_sumario_inserção = ' INSERT INTO PCD_VENDAS
                                    (SELECT *  FROM PCD_INS)'
```

C.10 Estrutura de dados *nós_busca*

```
Id = 1;
Filho(1) = 2
Filho(2) = 3
Tabela_delta_sumario = 'SD_PLD_VENDAS_LOCAL'

Tabela_delta_sumario_criação = ' CREATE TABLE
SD_PLD_VENDAS_LOCAL as select a.produto, a.loja, a.dia, a.qtd as qtd,
        a.receita as receita, b.qtd as min_qtd, c.qtd as max_qtd, a.contador
        from (select produto, loja, dia, sum(qtd) as qtd,
        sum(receita) as receita, sum(count) as count
        from sd_pld_Vendas@proxy group by produto,loja,dia) a
 LEFT OUTER JOIN
        PLD_VENDAS_MIN@proxy b
ON a.produto = b.produto and a.loja=b.loja and a.dia=b.dia
LEFT OUTER JOIN
        PLD_VENDAS_MAX@proxy c
ON a.produto = c.produto and a.loja=c.loja and a.dia=c.dia '

Id = 2;
Filho(1) = 4
Tabela_delta_sumario = 'SD_PLM_VENDAS_LOCAL'

Tabela_delta_sumario_criação = ' CREATE TABLE
SD_PLM_VENDAS_LOCAL as  select a.produto, a.loja, a.mês, a.qtd as qtd,
        a.receita as receita, b.qtd as min_qtd, c.qtd as max_qtd, a.contador
        from (select produto, loja, mês, sum(qtd) as qtd,
        sum(receita) as receita, sum(count) as count
        from sd_plm_Vendas@proxy group by produto,loja,mês) a
LEFT OUTER JOIN
        PLM_VENDAS_MIN@proxy b
ON a.produto = b.produto and a.loja=b.loja and a.mês=b.mês
LEFT OUTER JOIN
        PLM_VENDAS_MAX@proxy c
ON a.produto = c.produto and a.loja=c.loja and a.mês=c.mês'


Id = 3;
Filho(1) = null
Tabela_delta_sumario = SD_'PCD_VENDAS_LOCAL'

Tabela_delta_sumario_criação = ' CREATE TABLE
SD_PCD_VENDAS_LOCAL as select a.produto, a.cidade, a.dia, a.qtd as
     qtd, a.receita as receita, b.qtd as min_qtd, c.qtd as max_qtd, a.contador
        from (select produto, cidade, dia, sum(qtd) as qtd,
        sum(receita) as receita, sum(count) as count, max(lcv) as lcv
        from sd_pld_Vendas@proxy group by produto,cidade,dia) a
 LEFT OUTER JOIN
        PCD_VENDAS_MIN@proxy b
```

```
ON a.produto = b.produto and a.cidade=b.cidade and a.dia=b.dia
LEFT OUTER JOIN
        PCD_VENDAS_MAX@proxy c
ON a.produto = c.produto and a.cidade=c.cidade and a.dia=c.dia'


Id = 4;
Filho(1) = null
Tabela_delta_sumario = 'PCD_VENDAS_LOCAL'

Tabela_delta_sumario_criação = ' CREATE TABLE
SD_CatR_VENDAS_LOCAL as select a.categoria, a.região, a.qtd as qtd,
        a.receita as receita, b.qtd as min_qtd, c.qtd as max_qtd, a.contador,
        from (select categoria, região, sum(qtd) as qtd,
        sum(receita) as receita, sum(count) as count
        from sd_catr_Vendas@proxy group by categoria, região) a
LEFT OUTER JOIN
        CatR_VENDAS_MIN@proxy b
ON a.categoria = b.categoria and a.região=b.região
LEFT OUTER JOIN
        CatR_VENDAS_MAX@proxy c
ON a.categoria = c.categoria and a.região=c.região'
```