UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

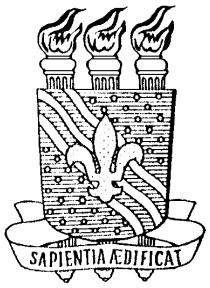
DISSERTAÇÃO DE MESTRADO

Guardando Históricos de Dimensões em Data Warehouses

André Barbosa Rocha

Campina Grande – PB

Fevereiro de 2000



UNIVERSIDADE FEDERAL DA PARAÍBA

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

GUARDANDO HISTÓRICOS DE DIMENSÕES EM DATA WAREHOUSES

André Barbosa Rocha

(MESTRANDO)

Ulrich Schiel

(ORIENTADOR)

Campina Grande – PB

Fevereiro de 2000

ANDRÉ BARBOSA ROCHA

GUARDANDO HISTÓRICOS DE DIMENSÕES EM DATA WAREHOUSES

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

Área de Concentração : **CIÊNCIA DA COMPUTAÇÃO**

Linha de Pesquisa: **SISTEMAS DE INFORMAÇÃO E BANCOS DE DADOS**

Orientador : **PROF. DR. ULRICH SCHIEL**

CAMPINA GRANDE

Fevereiro 2000

Ficha Catalográfica

ROCHA, André Barbosa

R672G

Guardando Históricos de Dimensões em Data Warehouses

Dissertação (Mestrado) – UFPB/CCT/COPIN, Campina Grande, Fevereiro de 2000.

152 p. Il.

Orientador: Ulrich Schiel

1. Bancos de Dados
2. Data Warehouses
3. Bancos de Dados Temporais

CDU: 681.3.07B

GUARDANDO HISTÓRICOS DE DIMENSÕES EM DATA WAREHOUSES

ANDRÉ BARBOSA ROCHA

DISSERTAÇÃO APROVADA EM 29.02.2000

PROF. ULRICH SCHIEL, Ph.D
Orientador

PROF. MARCUS COSTA SAMPAIO, Dr.
Examinador

PROF. DÉCIO FONSECA, Dr.
Examinador

CAMPINA GRANDE – PB

Dedico este trabalho a :

Minha mãe Gleudery. Um dia haveremos de nos encontrar novamente!

Poliana, minha metade.

Minha família, sempre confiante. Em especial à Yolanda, Mãe a quem devo tudo, a Glêryston, Pai e exemplo de vida, a Chagas, meu tio e pai, à Ângela, com muito carinho, e a meus irmãos de coração, Déborah e Danilo.

Todas as pessoas que vêm no conhecimento um caminho para libertação da alma.

“Digo-vos, em verdade, que, se não vos converterdes e tornardes quais crianças, não entrareis no reino dos céus. – Aquele, portanto, que se humilhar e se tornar pequeno como esta criança será o maior no reino dos céus – e aquele que recebe em meu nome a uma criança, tal como acabo de dizer, é a mim mesmo que recebe”. (S. MATEUS, cap. XVIII, vv 1 a 5.)

Agradecimentos

A Deus, pela oportunidade de existir.

A Ulrich Schiel e Marcus C. Sampaio pelo apoio e amizade desde meus tempos de graduação.

Às maravilhosas Josenita Ramos (Lili) e Águeda Cabral (Dadinha).

À Aninha, pelo carinho com que me tratou todos esses anos.

Ao professor Marcelo Barros por me apoiar em várias horas de minha vida.

Aos grandes "mafiosos", amigos de longa data.

Aos grandes amigos do LSI.

A todos os colegas, professores e funcionários do DSC pela amizade e companheirismo.

À CAPES, pela bolsa de dois anos.

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

OBRIGADO !

Resumo

ROCHA, André Barbosa, "Guardando Históricos de Dimensões em Data Warehouses", Dissertação de Mestrado, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Fevereiro 2000.

Um dos mais importantes patrimônios de uma organização é sua informação. Este patrimônio é quase sempre mantido na forma de sistemas transacionais, ou sistemas de produção. Grosso modo, um sistema de produção opera com dados detalhados e atualizados. Seu objetivo principal é apoiar as operações da organização no dia-a-dia, ou sua gerência no nível operacional. As gerências de nível mais alto da organização necessitam de informações sumariadas e históricas, não sendo portanto contempladas pelos sistemas de produção.

A incapacidade dos sistemas de produção de apoiar as atividades de gerência de uma organização em todos os seus níveis tem engendrado intensos esforços de pesquisa e desenvolvimento no sentido de se construir sistemas de apoio à decisão *stricto sensu*, chamados Data Warehouses. Apesar dos enormes progressos neste sentido, muitos requisitos dos sistemas de apoio à decisão carecem ainda de soluções adequadas.

Neste trabalho, tratamos do requisito *tempo* em Data Warehouses, apresentando uma solução formal para o problema do histórico de atributos de dimensões de Esquemas em Estrela. A motivação para o nosso trabalho partiu da constatação de que, embora os bancos de dados para sistemas de apoio à decisão devam ser temporais por definição, o tratamento que tem sido dado ao tempo (estruturas de dados e interfaces temporais) ainda é rudimentar.

Abstract

ROCHA, André Barbosa, "Guardando Históricos de Dimensões em Data Warehouses", Dissertação de Mestrado, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Fevereiro 2000.

One of the most important patrimony of an organization is its information. This patrimony is nearly always kept in the form of operational systems, or production systems. Broadly speaking, a production system works on updated and detailed data. Its main objective is to give support to the operations of the organization in the everyday life, or to its managing in the operational level. The higher levels of management in the organization need summarized and historical information, and are not, thus, covered by production systems.

The incapacity of production systems in supporting managing activities on an organization in all its levels has forced intense research and development efforts towards building decision supports systems *stricto sensu*, called Data Warehouses. In spite of the huge progress in this direction, many of the requirements of these systems sill lack adequate solutions.

In this work, we deal with the requirement *time* in Data Warehouses, presenting a formal solution for the problem of the dimensions attributes history in Star Schemas. The motivation for our work came from the awareness that, although the databases for decision support systems should be temporal by definition, the treatment that has been given to *time* (data structures and temporal interfaces) is still rudimentary.

Lista de Figuras

Sumário

Capítulo 1

Introdução

1.1 Contextualização

No mundo capitalista globalizado em que vivemos hoje, a informação é o patrimônio mais precioso de qualquer organização. Em todos os níveis organizacionais (do Operacional ao Estratégico) a boa manipulação das informações resulta em melhores resultados no desempenho das atividades empresariais. Em tal contexto, a tecnologia, e mais particularmente a Tecnologia da Informação, exerce um papel diferencial de produtividade e competitividade para as empresas.

Ao longo da evolução dos sistemas de informação (SI's), mudanças significativas aconteceram no papel e na abrangência da Informática dentro das organizações. O computador deixou de ser uma máquina de automação de processos produtivos para se tornar uma poderosa ferramenta de análise dos negócios para a alta gerência. Isto devido à crescente necessidade das empresas de agregarem valor às suas informações como uma forma de se manterem competitivas no mercado em que atuam.

Competitividade, no mundo capitalista globalizado em que vivemos hoje, não significa tão somente agilidade, eficiência e rapidez dos processos produtivos, mas principalmente ter capacidade de "prever" as possibilidades futuras, as inovações, as futuras necessidades do mercado e até mesmo a criação de novos mercados. Olhar para e tentar prever o futuro (ou até mesmo construí-lo) é uma questão de sobrevivência para qualquer organização moderna. E o futuro depende essencialmente de uma importantíssima atividade tático-estratégica : o planejamento.

As informações necessárias aos níveis decisórios das organizações são bastante distintas daquelas necessárias aos níveis de produção. A alta gerência necessita de informações sumariadas, agregadas sob diversos ângulos, orientadas a assuntos [Inmon 97]. Desta forma, os sistemas de informação que as manipulam também são essencialmente diferentes.

No atual estágio evolutivo dos sistemas de informação destacamos o Data Warehousing e o processamento OLAP (*On-line Analytical Processing*). Juntas, essas duas tecnologias provêem para o usuário executivo as informações necessárias no nível corporativo para melhores tomadas de decisão. É muito comum encontrarmos na literatura os termos OLAP e Data Warehousing como sinônimos. Podemos assegurar que apesar de muito próximos esses termos se complementam sem, no entanto, se confundirem. Verdadeiramente, o processamento OLAP está inserido no ambiente de Data Warehousing, que, grosso modo, é a infra-estrutura tecnológica de hardware e software para a atividade de análise gerencial.

Ambientes de Data Warehousing e ferramentas OLAP surgiram primeiramente no ambiente comercial, impulsionados pela crescente e diferenciada necessidade por informações no nível corporativo. Talvez por isso essas tecnologias careçam de melhor embasamento teórico-científico em alguns de seus conceitos e aspectos. A academia voltou seu interesse para tais tecnologias quando o mercado já estava por absorvê-las intensamente.

1.2 Motivação

Durante algum tempo a academia seguiu um curso diferente no que concerne à integração de múltiplas bases de dados heterogêneas com vistas à análise gerencial [Widom 95]. O mercado, por sua vez, sempre sedento por resultados imediatos, seguiu seu próprio caminho no desenvolvimento das tecnologias OLAP e Data Warehousing. Desta forma, os conceitos que envolvem essas tecnologias ainda estão fortemente arraigados no âmbito comercial, carecendo da independência e do rigor acadêmico. Certamente por isso, há ainda muitos pontos abertos a pesquisas nesta área.

Acreditamos que para fazer pesquisa nesta área é preciso, antes de tudo, esquecer sua origem comercial. Alguns aspectos já consagrados comercialmente precisam ser contestados para que se possa chegar a resultados independentes. Destacamos aqui o termo independente no sentido de que muito daquilo que é feito comercialmente é tendencioso. Além disso, muitas das idéias que são propostas e funcionam são encaradas como "a solução do problema" sem, no entanto, serem a melhor solução para o problema. Atribuímos isso ao ritmo alucinante do mercado de informática, que não deixa espaço para melhores investigações.

Alguns pontos destas tecnologias necessitam verdadeiramente de um estudo mais aprofundado. E um ponto crucial refere-se às características essencialmente históricas dos bancos de dados voltados para análise. A análise gerencial que é feita sobre repositórios OLAP tem um caráter histórico. Tanto que o tempo é tratado como uma dimensão à parte. Apesar dos data warehouses serem bancos de dados históricos por natureza, o tratamento dado ao tempo ainda é, em certos aspectos, rudimentar. Deve-se isso talvez ao esquecimento (ou falta de conhecimento) por parte dos *experts* em Data Warehousing da ampla pesquisa realizada sobre bancos de dados temporais. Certamente essas duas áreas estão intimamente ligadas, porém, pouco se tem feito para uni-las.

Neste trabalho atacamos um problema básico dos esquemas em estrela (o esquema relacional para data warehouses). Na forma como são concebidos, estes esquemas tratam as tabelas de dimensão como sendo estáticas, imutáveis. Porém, mostramos nesse trabalho que isso não é verdade na prática. Valores de atributos de dimensões podem mudar ao longo do tempo. Tais alterações podem ser de grande interesse para o usuário executivo. Portanto, devemos de alguma forma guardar o histórico de seus valores antigos. Afinal, assim como os bancos de dados temporais, data warehouses são *append-only*. Informações armazenadas em um data warehouse não devem simplesmente ser descartadas, já que elas tiveram seu período de validade.

As soluções propostas por outros autores para permitir alterações em valores de atributos dimensionais não são satisfatórias. Boa parte delas são decisões de projeto que atacam problemas específicos e muitas delas não guardam o histórico

corretamente. Além disso, todas carecem de rigor formal. Por isso resolvemos atacar esse problema.

1.3 Objetivo

O trabalho objetiva fornecer uma solução completa, eficaz e formal para guardar o histórico de dimensões dos esquemas em estrela. A tarefa consiste em manter o histórico de valores de atributos dimensionais de forma correta, concisa, simples e geral. Nossa solução não altera o esquema em estrela tradicional para que suas principais características se mantenham intactas. Ela permite que os históricos de quaisquer atributos (ou conjuntos de atributos) de quaisquer dimensões sejam guardados corretamente.

Acreditamos que a solução para guardar qualquer informação histórica passa pela teoria dos bancos de dados temporais. Por isso fomos buscar nessa teoria os conceitos e técnicas que poderiam ser-nos úteis, adaptando-os para nosso problema específico. Essa adaptação é cercada de algumas restrições, impostas pela natureza do ambiente de DWing que difere substancialmente do ambiente operacional para o qual os BD's temporais ainda são desenvolvidos.

Não restringimos nosso objetivo a desenvolver uma solução apenas no nível de infraestrutura, como as outras propostas de outros autores fizeram. Nosso objetivo é também fornecer uma interface para manipulação da estrutura que propomos (o MET* - Meta-esquema Temporal em Estrela). Porém, além de uma interface gráfica que está sendo construída, definimos uma linguagem de consulta (MET*SQL). Essa necessidade surgiu na medida em que a estrutura proposta possibilitou a execução de novas operações analíticas, diferentes daquelas executadas sobre esquemas em estrela tradicionais.

Preocupamo-nos também em evitar ambigüidades. Por isso optamos por formalizar nossa proposta. A formalização abrange deste a estrutura lógica até a linguagem de consulta para sua manipulação. Com isso, nossa solução pode ser chamada de modelo. Todavia, preferimos continuar chamando-a de esquema já que ela

é uma extensão dos esquemas em estrela e este termo já está consagrado na literatura de DWing.

1.4 Relevância

Acreditamos que a relevância do nosso trabalho será melhor entendida à medida que o leitor analisar os vários exemplos de situações específicas e típicas da análise gerencial que discutimos ao longo do trabalho.

Vale salientar que o problema das dimensões que mudam com tempo foi discutido inicialmente por Ralph Kimball em [Kimball 98] e em alguns artigos técnicos de revistas especializadas [Kimball 98b]. Aparentemente ele foi o primeiro autor a detectar tal problema em projetos de DWing.

Ao longo de nossos trabalhos, porém, pudemos comprovar a importância do tema através de discussões que surgiram na lista de discussão por e-mail dwlist@datawarehousing.com. Esta lista é composta por pessoas de várias partes do mundo ligadas a projetos ou pesquisas em DWing. Um dos integrantes é o próprio Ralph Kimball. Nos anos de 1998, 1999 e início de 2000 surgiram várias discussões sobre o assunto. Boa parte delas foi iniciada por projetistas que estavam enfrentando problemas em atender certas exigências dos usuários de seu ambiente. Essas exigências tinham a ver justamente com o histórico de atributos dimensionais. Nessas discussões pudemos avaliar o quão este tema é importante e o que está sendo feito pelo mundo afora para solucioná-lo.

O mais intrigante é que nenhuma das soluções apresentadas na lista pelo menos cita o termo bancos de dados temporais. Muitas delas são apenas variações das propostas de Kimball. Outras, inconscientemente, passam perto da teoria do bancos de dados temporais. Porém, nenhuma delas efetivamente utiliza os conceitos dessa tecnologia para resolver o problema do histórico de dimensões.

Quando chegávamos à fase final do trabalho, encontramos um artigo acadêmico sobre o assunto [Bliujute et al 98]. Ele veio reforçar ainda mais a relevância de nosso trabalho, até porque a forma como ele ataca o problema já tem um pouco a ver com

BD's temporais. No entanto, a solução proposta deixa muitos pontos em aberto que podem levar a problemas em projetos de esquemas em estrela.

1.5 Organização

O restante do nosso trabalho está estruturado da seguinte forma:

- No Capítulo 2 discutimos brevemente a evolução dos sistemas de apoio à decisão. Em seguida nos aprofundamos nos termos OLAP e Data Warehousing apresentando definições, características e operações efetuadas nesses ambientes;

- No Capítulo 3 detalhamos o esquema em estrela, sua estrutura típica, algumas variações dela e as principais operações realizadas sobre este esquema. Por fim apresentamos o problema do histórico de dimensões através de exemplos;

- O Capítulo 4 contém as principais propostas de alguns autores para resolver o problema. Discutimos também os pontos fracos dessas soluções;

- No Capítulo 5 apresentamos nossa solução: o MET*(Meta-esquema Temporal em Estrela). Iniciamos este capítulo discutindo os principais conceitos dos bancos de dados temporais. Em seguida apresentamos o MET* informalmente. Logo depois formalizamos a estrutura e a álgebra do MET* e por último apresentamos o MET*SQL (uma extensão do SQL 92 baseada na álgebra proposta) que permitirá a manipulação de dados históricos no esquema;

- O Capítulo 6 relata a implementação do MET* no SGBD Oracle8 Versão 8.0.4. Esta implementação utilizou-se da tecnologia objeto-relacional. Aproveitamos para discutir alguns pontos da implementação desta tecnologia neste SGBD e quais as suas limitações para implementação de data warehouses.

- O Capítulo 7 apresenta nossas conclusões e os caminhos que pretendemos seguir no futuro para dar prosseguimento a este trabalho.

Capítulo 2

Sistemas de Informação, OLAP e Data Warehousing

2.1 Introdução

Como em qualquer atividade humana, o planejamento empresarial consiste primeiramente em se entender fatos presentes e passados, entendimento este que depende de informações sobre tais fatos, sejam elas "boas ou más" [Souza 99]. As informações necessárias para um bom planejamento diferem em sua essência das informações necessárias à automação de processos produtivos. Aquelas são sumariadas, históricas, analíticas, agregadas e orientadas a assuntos, enquanto que estas são detalhadas, atualizadas e orientadas a aplicações [Inmon 97]. Desta forma, também os Sistemas de Informação (SI's) que as manipulam são essencialmente diferentes.

As diferenças expostas acima foram um dos principais fatores para que houvesse evolução dos SI's. Partindo dos Sistemas de Automação (conhecidos como EDP's : *Electronic Data Processing*) do nível operacional (execução) os SI's atingiram o nível tático (controle) com a implantação dos MIS's (*Management Information Systems*) e posteriormente chegaram ao nível estratégico (gerência) com o desenvolvimento dos EIS's (*Executive Information Systems*) e dos DSS's (*Decision Support Systems*). Em cada nível, mudou substancialmente o foco e a organização das informações disponibilizadas ao usuário final, além das técnicas e da infra-estrutura para cada tipo de SI.

Na próxima seção discutiremos rapidamente a evolução bem como as principais características de cada tipo de SI. No restante do capítulo enfocaremos um tipo de

processamento específico para EIS e DSS, processamento OLAP, e as técnicas e infra-estrutura por trás dessa tecnologia, consubstanciadas no Data Warehousing.

2.2 Evolução e Contextualização dos SI's

A evolução dos SI's está, obviamente, intimamente ligada à evolução das tecnologias de processamento, armazenamento e transmissão de dados. Entretanto essa relação não fecha um círculo em si mesma. O objetivo final dos SI's é "melhorar o desempenho de funcionários das empresas através da aplicação de tecnologia de informações" [Sprague&Watson 91], ou seja, "tornar disponíveis informações corretas, precisas, atualizadas e concisas, na apresentação adequada e no momento ideal" [Souza 99]. Portanto, a evolução do SI's deve-se verdadeiramente às crescentes e diferenciadas necessidades de informação para melhor desempenho dos funcionários que dependem desses sistemas, necessidades estas que variam enormemente de acordo com a função do funcionário dentro da organização.

Os SI's passaram por três estágios evolutivos distintos. Na seqüência, resumimos os principais aspectos de cada estágio, que sejam conhecidos, de antemão, como Era da Automação, Era do Controle e Era da Produtividade [Souza 99].

No primeiro estágio, chamado de **Era da Automação**, os SI's tinham por principal objetivo a automação de tarefas rotineiras e de baixa complexidade como cálculos, totalizações, comparações, geração de relatórios, etc. Seu foco estava no armazenamento, processamento e fluxo de dados do nível operacional [Souza 99] [Sprague&Watson 91].

Esses sistemas, classificados como EDP (*Electronic Data Processing*) inicialmente consistiam de programas escritos em linguagem de alto nível, geralmente COBOL, que manipulavam arquivos mestres armazenados em fita magnética. Mais tarde, com o advento do armazenamento em disco, como forma de resolver os sufocantes problemas do armazenamento em fita magnética [Inmon 97], surgiram novas técnicas de armazenamento e recuperação de dados. Citamos aqui, os sistemas de arquivos, que usavam as técnicas de armazenamento seqüencial e recuperação indexada; os Bancos de Dados Hierárquicos, como o IMS (*Information Management System*) da IBM; e por fim, os Bancos de Dados em Rede, que seguiam o modelo em rede formulado pelo

CODASYL (*Comittee on Development of Applied Symbolic Languages*) na década de 70. O principal modo de processamento desses sistemas era batch.

Em meados da década de 70, surgiu o processamento on-line. Ao contrário dos sistemas *batch*, os sistemas *on-line* refletem imediatamente no BD qualquer processamento feito sobre os dados, provendo também um "tempo de resposta imediato" ao usuário. Neste estágio surgiram os primeiros SGBD's (Sistemas Gerenciadores de Bancos de Dados) relacionais que implementavam o Modelo Relacional de Codd e utilizavam o conceito de transações serializáveis[1] para o processamento de informações [Korth&Silberschatz 95]. O conceito de Banco de Dados surge como "uma única fonte de dados para todo o processamento" [Inmon 97], tornando possível separar os detalhes tecnológicos e físicos do processamento e armazenamento dos dados, das regras de negócio. Os dados agora são visualizados sob diferentes níveis (nível físico, nível conceitual e nível de visões) pelos diversos tipos de usuários de SI's [Korth&Silberschatz 95] .

No segundo estágio, conhecido como **Era do Controle**, surgiram os primeiros sistemas voltados para gerência : os MIS's (*Management Information Systems,* em português, Sistemas de Informações Gerenciais).

Inicialmente, o MIS objetivou integrar grupos de SI's para prover informações no nível corporativo, abrangendo todos os níveis decisórios da organização [Sprague&Watson 91]. Este novo enfoque gerencial provocou uma grande mudança na forma como os dados eram vistos pela organização. A informática evoluiu de sua função de automação (execução) para se tornar uma ferramenta de planejamento e controle, uma nova maneira de medir, avaliar, analisar e controlar caminhos a serem seguidos para o alcance de metas e objetivos.

O desenvolvimento desses sistemas baseava-se na arquitetura de desenvolvimento espontâneo [Inmon 97] cujo núcleo eram os programas extratores [Inmon 97]. Tais programas selecionavam dados de arquivos ou bancos de dados de acordo com algum critério de seleção e os transportavam para outros arquivos ou

[1] Unidades atômicas que agrupam conjuntos de operações sobre dados e que obedecem às propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

bancos de dados, formando assim bases de dados mais próximas dos aspectos gerenciais particulares de cada ramo do negócio.

A arquitetura de desenvolvimento espontâneo tinha como paradigma simplesmente "deixar acontecer". Tal falta de controle sobre a construção de novas bases levou à explosão de programas extratores dentro das organizações. [Inmon 97] afirma que era comum uma grande empresa fazer até 45 mil extrações diárias. Sem controle, vários problemas começaram a surgir decorrentes desta arquitetura. Dentre eles o mais grave para a gerência foi certamente a "falta de credibilidade dos dados" [Inmon 97]. Eram comuns relatórios diferentes sobre o mesmo assunto apresentarem dados discrepantes, o que, ao invés de apoiar decisões, as atrapalhava. Alguns outros problemas apresentados na literatura [Inmon 97] [Souza 99] limitaram o papel do MIS. Entre outros, citamos : o tipo de processamento utilizado, o transacional clássico, que limitava a tarefa de análise gerencial por não possibilitar uma visão realmente administrativa/gerencial dos dados; e a limitação das tecnologias da época (inicio dos anos 80), como interfaces, arquiteturas de hardware, etc.

Pelos fatores citados acima, o papel dos MIS's limitou-se a orientar decisões operacionais e mais tarde decisões táticas, o objetivo de fornecer subsídios para decisão gerencial no nível corporativo não foi atingido, até porque esta não é uma tarefa trivial [Souza 99].

Em suma, as informações fornecidas pelos MIS's restringiram-se a relatórios gerenciais pré-definidos (totalizações, comparações, percentuais, etc.) disponibilizados em níveis de departamentos e setores (MIS para marketing, MIS para produção, etc.) [Souza 99] [Sprague&Watson 91]. Seu foco centrou-se nas informações direcionadas a gerentes no nível tático.

A continuação do ciclo evolutivo dos SI's se deu com o surgimento de novos sistemas denominados EIS (*Executive Information Systems,* em português, Sistemas de Informações Executivas) e mais tarde dos DSS (*Decision Suport Systems,* em português, Sistemas de Apoio à Decisão). Como os próprios nomes dizem, esses sistemas destinam-se ao nível executivo, corporativo, estratégico das empresas. Estamos na **Era da Produtividade** [Souza 99].

O suporte tecnológico necessário a estes sistemas enfim foi consolidado. Interfaces gráficas, arquitetura cliente-servidor, PC's com grande capacidade de processamento, redes de telecomunicações e corporativas integradas, etc. impulsionaram seu desenvolvimento e sua implantação. Agora o usuário tomador de decisão (do inglês: *decision maker*) consegue ter acesso à informação de sua estação de trabalho na sua própria mesa.

Segundo [Kelly 95], "o principal objetivo do EIS é suportar o aprendizado gerencial sobre a organização ... e sua interação com o ambiente externo". O EIS permite ao usuário executivo localizar problemas e determinar tendências relacionadas aos vários aspectos do negócio [Inmon 97], aspectos que vão desde desempenho em mercados até produtividade de funcionários [Kelly 95]. O foco do EIS está sobre as decisões estratégicas da alta-gerência. Os executivos deste nível possuem visões diferentes dos dados, é a visão corporativa, que em nada se assemelha à visão operacional. À alta-gerência interessam dados resumidos, agregados sob diversos aspectos, orientados a assuntos e históricos. O acesso aos dados deve ser rápido, direto, fácil, intuitivo e sob a perspectiva dos negócios, já que o usuário do EIS não é um *expert* em Ciência da Computação.

Para oferecer o ferramental necessário à alta gerência, os EIS's precisam de dados organizados numa forma essencialmente diferente dos dados operacionais dos EDP's. Como citado no parágrafo anterior, os dados necessários à análise devem ser agregados, históricos, resumidos e orientados a assunto. Portanto, a organização lógica e física da informação necessária ao processamento de EIS (ou seja, de análise) difere sobremaneira daquela fornecida pelos EDP's do nível operacional. Ela precisa refletir a visão empresarial da alta gerência, precisa refletir seus modelos mentais [Kelly 95].

As tradicionais técnicas de modelagem de dados (Entidade-Relacionamento, Orientação a Objetos) mostraram-se ineficientes para construção de repositórios para o processamento de EIS [Kimball 98] [Thomsen 97] [Codd 93] . Essas técnicas não ofereciam os recursos necessários à visualização dos dados agregados sob diferentes aspectos do negócio. Uma nova técnica conhecida como *Modelagem Multidimensional* (MM) [Kenan 95] [Thomsen 97] [Kimball 97] foi proposta para a modelagem de repositórios EIS.

Na MM, os dados operacionais são visualizados sob diversos ângulos administrativos, refletindo os modelos mentais da alta-gerência. Esses ângulos administrativos são chamados **dimensões** do negócio (ex. produto, loja, promoção, região, etc.). Referentes a cada dimensão existem **fatos** comerciais relevantes à gerência (ex. vendas, compras, etc). **Fatos** são valores quantitativos sobre o desempenho comercial de um grupo de dimensões. Assim, por exemplo, podemos visualizar o fato **vendas** por **loja**, por **produto** e por **região**.

Para melhor compreender o descrito acima, observe a Figura 2.1. Esta estrutura é um cubo tridimensional com as dimensões: produto, região e tempo. Note que cada face representa uma dimensão. O cubo em si é preenchido com valores quantitativos que representam o número de unidades vendidas de determinado produto, durante um determinado tempo numa determinada loja. Esta estrutura é conhecida como cubo multidimensional [Kenan 95] [Thomsen 97]. No nosso exemplo, representamos apenas três dimensões pela nossa capacidade de visualização espacial limitar-se a esse número. Todavia, é possível estruturas com N dimensões.

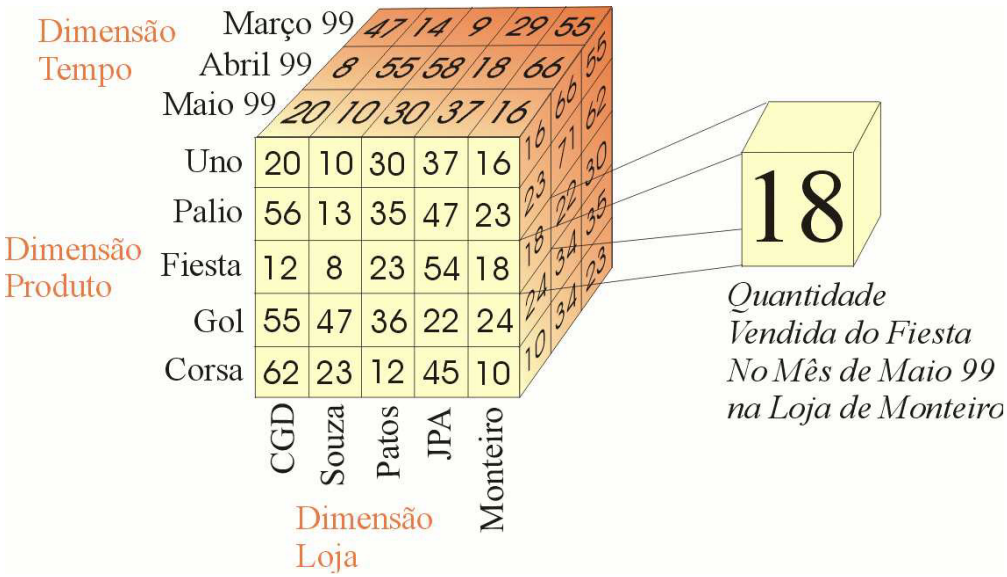


Figura 2.1 - Cubo multidimensional com três dimensões.

O tipo de processamento aplicado sobre estruturas dimensionais difere substancialmente daquele utilizado por aplicações do nível operacional[2]. Surge o processamento analítico denominado OLAP (*On-line Analytical Processing*) [Codd 93] [OLAP Council 95] [OLAP Council 97] [Pendse 98] [Thomsen 97], com a necessária flexibilidade esperada pelos usuários de EIS. O processamento OLAP tornou-se então a alma do EIS. A próxima seção deste capítulo é dedicada a OLAP.

No mundo dos sistemas ditos gerenciais encontrados mais uma sigla: DSS (*Decision Support Systems*). Ela tem sido freqüentemente usada como sinônimo de EIS e até mesmo de MIS por alguns autores e por diversos fabricantes de software, devido talvez ao impacto causado pela sua nomenclatura, Apoio à Decisão, que torna o marketing dos fabricantes mais apelativo.

Alguns autores consideram os DSS's como uma evolução dos MIS's [Inmon 97], outros como um sinônimo de EIS e OLAP [Power 99] [POE et al 98] e outros ainda os associam com *Data Mining* [Souza 99]. Não há um consenso sobre o uso desse termo para caracterizar uma gama de sistemas voltados para uma atividade específica e bem definida. Uma melhor e mais detalhada discussão sobre esse tema pode ser encontrada em [Souza 99]. Concordamos com esse autor quando afirma que os DSS's não são uma evolução dos MIS's por diversos fatores, sendo os mais importantes deles o tipo de processamento envolvido (OLTP no MIS e OLSP [Souza 99] no DSS), a modelagem (E-R versus Dimensional) e a forma de armazenamento dos dados. Concordamos também no fato de que os DSS envolvem a aplicação de heurísticas para reconhecimento de padrões e descoberta de tendências, e que o processamento usado nos DSS's difere em alguns pontos do processamento OLAP. Porém, fica claro também nesta referência que os EIS's e os DSS's são muito próximos em algumas de suas características.

[2] O Processamento Transacional Clássico utilizando por grande parte das aplicações do nível operacional através dos SGBD's Relacionais é chamado de *On-Line Transaction Processing* (OLTP).

2.3 On-Line Analytical Processing (OLAP)

O mercado atual está cada vez mais competitivo, o que obriga organizações a se tornarem cada vez mais ágeis no âmbito do seu planejamento estratégico. Essa agilidade, por sua vez, depende cada vez mais da sofisticação e rapidez dos Sistemas de Informação e da capacidade de se analisar e sintetizar informações a partir deles [Codd 93].

Tal necessidade crescente por informações úteis à análise impulsionou - juntamente com o desenvolvimento de interfaces gráficas cada vez mais intuitivas, o surgimento de novas arquiteturas como a cliente-servidor, o aumento da capacidade de processamento dos micro-computadores, etc.- a utilização de dados operacionais por parte de analistas de alto nível. Através de ferramentas como planilhas eletrônicas, os dados armazenados nos sistemas operacionais[3] passaram a ser usados como base para análise [Thomsen 97]. Entretanto, a atividade de análise necessitava de melhores ferramentas, capazes de efetuar cálculos mais complexos, de fornecer a visão corporativa dos dados sob diferentes ângulos, com interfaces mais amigáveis e capazes de efetuar operações específicas da análise.

2.3.1 Definição

O termo OLAP foi inicialmente proposto por Codd em [Codd 93] e vem sendo utilizado para caracterizar uma nova gama de aplicações voltadas para análise. Essas aplicações executam o processamento OLAP cujo objetivo é prover ao usuário a visualização dos dados sob diferentes ângulos gerenciais e comportar todas as necessidades da atividade de análise [Souza 99]. Ferramentas OLAP possuem capacidades analíticas para visualizar os dados sob diferentes ângulos de forma fácil e rápida [Alalouf 97].

---

[3] O termo Sistemas Operacionais (*Operational Systems*) é muito utilizado na literatura para caracterizar os sistemas de automação do nível operacional, os EDP's, porém não deve ser confundido com o mesmo utilizado para caracterizar a camada de software básico utilizada para gerência de recursos de hardware e software nos computadores. Esta confusão deve-se à tradução dos dois termos em inglês : *Operational Systems* e *Operating Systems*, recaírem no mesmo termo em português: Sistemas Operacionais.

Segundo [Thonsem 97], "OLAP é o processo de criar e gerenciar dados corporativos multidimensionais para visualização e análise do usuário que deseja enxergar o que realmente os dados dizem". OLAP conduz analistas, gerentes e executivos a "descobrirem conhecimentos" nos dados através de acesso rápido, interativo e consistente a uma variedade de visões possíveis da informação [OLAP Council 97].

Note-se a referência a termos como "diferentes ângulos", "dados multidimensionais" e "variedade de visões possíveis". Todos eles referem-se à análise multidimensional, que é o coração do OLAP. Esses diferentes ângulos representam os diferentes ramos, áreas, aspectos ou componentes do negócio: as dimensões do negócio (produto, loja, fornecedor, promoção, etc.). Referentes às dimensões existem fatos quantificáveis sobre o desempenho dessas em atividades do negócio (fato vendas, fato compras, etc.). Dados esses conceitos que refletem os dois pontos importantes na análise multidimensional, trazemo-los para o mundo dos SI's. Os atributos dos dados operacionais podem ser classificados no processamento OLAP, segundo sua natureza, em:

- Atributos de Fatos : atributos quantitativos sobre o desempenho do negócio em determinado(s) ramo(s). Exemplo : sobre o fato vendas, a quantidade vendida, o preço da unidade, a margem de lucro, etc;

- Atributos de Dimensão : atributos qualitativos que caracterizam os ramos do negócio envolvidos na medida de desempenho de determinado fato. Exemplo : sobre a dimensão produto, descrição, embalagem, preço, etc.;

Os conceitos de fatos e dimensões permeiam todo o universo da análise multidimensional. Esses dois conceitos são suficientes para prover a visão gerencial dos dados. As dimensões, como representantes de áreas, aspectos, ramos ou componentes do negócio provêm a visão dos dados que realmente importa aos tomadores de decisão. Os fatos, por sua vez, expressam a medida de desempenho dos componentes das dimensões. São os fatores quantitativos que permitem a introspecção nos dados para descoberta de tendências e reconhecimento de padrões.

Voltando à Figura 2.1, podemos enxergar melhor o discutido acima. Note-se que cada dimensão diz respeito a determinado setor específico da empresa. A gerência regional expressa pela dimensão loja, a gerência de produto expressa pela dimensão produto e a gerência financeira expressa pela dimensão tempo. Cada gerência pode isolar facilmente os fatos de seu interesse no cubo. Por exemplo, para a gerência regional são de interesse as vendas da cidade de Monteiro. Esses valores podem ser recuperados "fatiando" um "pedaço" específico do cubo, como mostra a Figura 2.2.

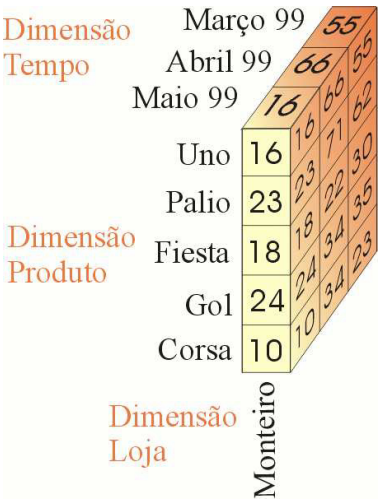


Figura 2.2 - Corte do cubo multidimensional para a Gerência Regional.

2.3.2 Requisitos para Aplicações OLAP

Seguindo a tendência das doze regras definidas para BD's Relacionais (BDR) em 1985, E. F. Codd definiu em 1993 outras doze regras para avaliação de sistemas OLAP. Estas regras podem ser encontradas em [Codd 93]. Entretanto, ao contrário das regras para BDR, as novas regras definidas por Codd não atingiram o sucesso esperado. Kimball afirma em [Kimball 98] que "os doze critérios OLAP são muito vagos para serem usados como diretrizes de avaliação de um sistema... A filosofia OLAP necessita de critérios mais específicos". Segundo [Pendse 98], "as regras de Codd tornaram-se indesejáveis para detectar a 'obediência' OLAP". Segundo também [Thomsen 97], "a credibilidade do artigo foi manchada quando se descobriu que fora patrocinado por uma companhia particular, Arbor Software, '...' e propôs mostrar que apenas o produto da Arbor Software satisfazia uma porcentagem significativa das Doze Regras de

Codd." Mais tarde, em 1995, outras 6 regras foram acrescentas às 12 perfazendo um total de 18 regras.

Abaixo, compilamos alguns requisitos para sistemas OLAP de algumas fontes diferentes (inclusive das regras de Codd) que, por aparecerem freqüentemente na literatura [MicroStrategy 95] [Thomsen 97] [Pendse 98] [Alalouf 97] [OLAP Council 97] [Codd 93] [Souza 99] [Next Action 98], consideramos os mais importantes.

- **Sistemas OLAP devem fornecer Visão Multidimensional dos Dados**: este é um consenso sobre uma característica desejável das aplicações OLAP. Afinal, análise multidimensional é a base para OLAP;

- **O acesso aos dados deve ser rápido**: este é outro consenso sobre o tema. O tempo de resposta de aplicações OLAP deve ser da ordem de segundos ou poucos minutos, isso porque geralmente um usuário final considera que um processo falhou depois de 30 segundos de espera. Outro fator de influência é a forma pela qual se dá a atividade de análise: o conjunto resposta de uma consulta geralmente condiciona à formulação de uma nova questão que resulta numa outra consulta. Este processo é executado seqüencialmente pelo analista. Se o tempo de resposta é lento, esta seqüência é quebrada trazendo prejuízos para a análise.

- **Sistemas OLAP devem suportar análises complexas**: Totalizações, comparações e agrupamentos não são as únicas operações de análise. Algumas operações envolvem cálculos mais complexos. Um sistema OLAP deve suportar bem consultas e cálculos ad-hoc sobre grandes quantidades de dados. Algumas dessas consultas incluem análise de séries temporais, alertas de exceções, cálculos estatísticos e de ranking, análise histórica, etc. Muitas técnicas são utilizadas para melhorar o desempenho de consultas complexas sobre grandes conjuntos de dados, incluindo a pré-computação de agregados [Kimball 98].

- **Acesso aos dados fácil e intuitivo**: para isso é necessário o uso interfaces gráficas. É comum o uso de software de planilhas, de gráficos (de vários tipos) e geradores de relatórios interativos.

- **Suporte Temporal**: o tempo é um componente integral de qualquer tarefa de análise. O tempo é uma dimensão intrínseca a qualquer negócio. Portanto, uma aplicação OLAP deve fornecer suporte à análise temporal.

- **Suporte a grandes conjuntos de dados**: aplicações OLAP devem suportar consultas e cálculos ad-hoc sobre grandes conjuntos de dados sem prejudicar a performance. Freqüentemente repositórios para OLAP armazenam dados no menor nível de granularidade[4] possível, ou seja, dados atômicos. Desta forma é comum encontrarmos repositórios de tamanho da ordem de terabytes.

- **Sistemas OLAP devem ser flexíveis**: Segundo [Thomsen 97], sistemas OLAP devem ser flexíveis sob diversas óticas : visão flexível dos dados, análise flexível, definições flexíveis e interfaces flexíveis. Destacamos um ponto importante na flexibilidade : definições flexíveis. Isto se refere ao sistema aceitar mudanças no modelo lógico do repositório OLAP sem grandes alterações em sua estrutura física. Mais claramente, deve-se permitir a adição, remoção, alteração de dimensões sem grande impacto na estrutura dimensional. Um cubo dimensional, por exemplo, precisa ser totalmente reestruturado na adição de uma dimensão.

- **Suporte Multi-usuário**: É comum em ambientes de análise, que dezenas, centenas e até milhares de usuários estejam usando simultaneamente um sistema OLAP, através de LAN's ou mesmo WAN's. Deve-se prover a segurança, confiabilidade e robustez necessárias a um ambiente como este. Além disso, na atividade de análise há muita cooperação entre usuários, portanto, cooperação á mais uma característica desejável de um sistema OLAP.

Como afirmando anteriormente, os requisitos citados acima não são um guia completo para avaliação de uma aplicação OLAP. Eles foram os comumente encontrados na literatura pesquisada. Não há um consenso sobre todas as características desejáveis para sistemas OLAP. Até mesmo as dezoito regras de Codd são alvos de críticas e controvérsias.

2.3.3 Modelos para Repositórios OLAP

Está claro até este ponto que as necessidades ditas gerenciais diferem muito das necessidades ditas operacionais. Os sistemas operacionais geralmente se baseiam em processamento transacional clássico, onde um grande número de transações é executado num pequeno intervalo de tempo, com alto grau de concorrência. As atividades dos sistemas operacionais tendem a acontecer numa taxa relativamente constante. Os dados são detalhados, as atualizações são tão freqüentes quanto as leituras, os bancos de dados refletem o estado atual da informação, as consultas geralmente ocorrem sobre um pequeno conjunto de dados e são conhecidas de antemão [Thomsen 97] e a estrutura dos dados é fixa enquanto que o conteúdo é variável. O foco dos sistemas operacionais está no armazenamento, na confiabilidade, na performance e na disponibilidade dos dados. Eles mantêm a empresa funcionando [Kimball 98].

No outro extremo, os sistemas projetados para análise procuram facilitar consultas e navegação na estrutura dimensional e assim prover informação realmente gerencial. Os dados para aplicações OLAP são resumidos, históricos e baseados em assuntos. As consultas OLAP geralmente *ad-hoc* e acessam grandes conjuntos de dados. Não há atualizações como as que executamos no nível operacional, essas geralmente são feitas em **batch** e por causa disso não há controle de concorrência. Os dados não necessariamente seguem os princípios da normalização [Korth&Silberschatz 95] já que o foco do OLAP está na recuperação e visualização e não no armazenamento. A estrutura juntamente com o conteúdo é flexível.

[4] A granularidade expressa o nível de detalhe em que os dados são armazenados nos repositórios OLAP.

Pelas diferenças discutidas acima, a modelagem para repositórios OLAP distancia-se sobremaneira das técnicas usadas para BD's operacionais. Como exposto na Seção 2.2, a técnica utilizada para construção de repositórios OLAP é a modelagem multidimensional [Chuck et al 98] [Kimball 98].

Existem duas grandes vertentes para modelagem multidimensional : MOLAP e ROLAP, as quais descrevemos nas subseções seguintes.

2.3.3.1 MOLAP (Multidimensional OLAP)

Também chamada de MDOLAP (*Multi Dimensional OLAP*) [MicroStrategy 95] e MDDB (*MultiDimensional Database*), a abordagem MOLAP baseia-se nos Bancos de Dados Multidimensionais [Kenan 95]. A estrutura por trás dessa abordagem é o cubo multidimensional[5] (ou *Multidimensional Array*, em português, Vetor Multidimensional) como o da Figura 2.1. Esses cubos geralmente são implementados sob a forma de matrizes esparsas[6]. Esquemas especiais de compressão e indexação são usados para contornar os problemas dos espaços vazios.

Em um cubo multidimensional, cada aresta representa uma dimensão do negócio. Cada linha ou cada coluna de cada aresta representa um elemento de uma dimensão (ex. CGD, Monteiro, etc. para a dimensão Loja e Uno, Palio e Fiesta para a dimensão Produto). As células do cubo são preenchidas com valores quantitativos que são as medidas ou fatos do negócio.

Por uma questão de simplicidade, geralmente os exemplos de cubos multidimensionais restringem-se a três dimensões. Entretanto, em sistemas reais, podem haver N dimensões. A Figura 2.3 mostra visualmente a inclusão de uma nova dimensão no cubo da Figura 2.1. Para facilitar essa visualização, a dimensão tempo foi transferida para fora da estrutura quadridimensional e a dimensão Promoção foi incluída em seu lugar. Agora temos um cubo com quatro dimensões.

---

[5] Aliás, um abuso de linguagem, porque matematicamente um cubo tem três, e apenas três dimensões.

[6] Matrizes com muitos espaços vazios em suas células. Espaços vazios ocorrem pela "ausência de valores que atendam ao critério de interseção entre dimensões, por exemplo, em um sistema de vendas nem todo produto é vendido em todos os departamentos ou durante todo um período de tempo." [Souza 99].

Soluções baseadas em cubos multidimensionais são proprietárias. As tecnologias empregadas para processamento e armazenamento dependem dos fabricantes, muitas vezes sendo incompatíveis entre si. Alguns líderes de mercado são Arbor Software com o Essbase, Kenan Technologies com o Acumate Enterprise, Oracle/IRI com o Express e D&B/Pilot Software com o LightShip [Firestone 97].

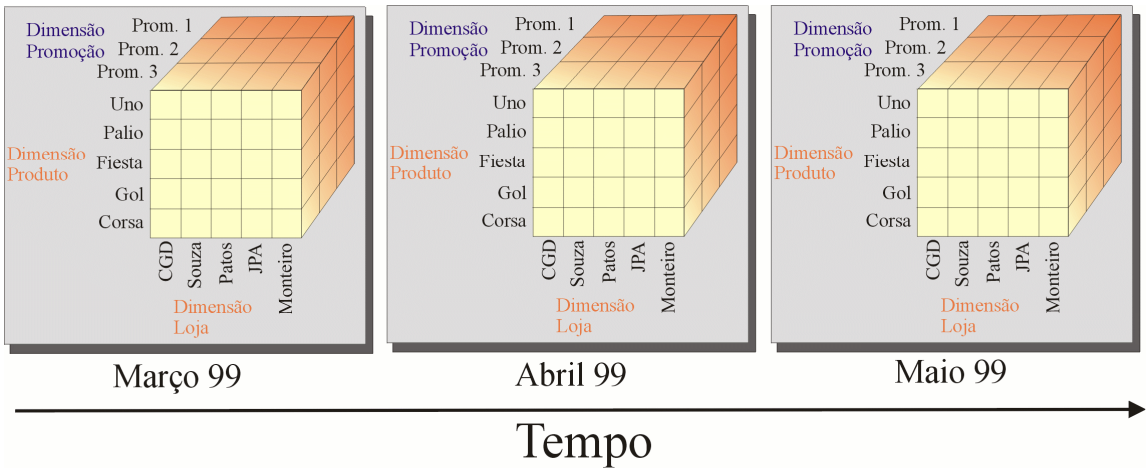


Figura 2.3 - Cubo quadridimensional.

2.3.3.2 ROLAP (Relational OLAP)

Para enfrentar as tentadoras facilidades oferecidas pelos BD's Multidimensionais, a "indústria relacional" logo propôs sua alternativa. O ROLAP [Alalouf 97] [Chuck et al 98] [Kimball 98] [Kimball et al 98] [Microstrategy 95] [Next Action 98] [POE et al 98] [Souza 99] utiliza-se de tabelas relacionais para simular o cubo multidimensional.

O esquema proposto concentra numa tabela as medidas ou fatos do negócio. Esta tabela é conhecida como tabela de fatos (do inglês : *fact table*). Os componentes das dimensões são armazenados em outras tabelas, chamadas tabelas de dimensão (do inglês: *dimension tables*). As tabelas de dimensão compõem com a tabela de fatos um relacionamento de chave estrangeira 1-N (um para N). Uma tabela de fatos está ligada a várias tabelas de dimensão, formando uma estrutura semelhante a uma estrela, fato esse que lhe deu o nome de Esquema em Estrela (do inglês *Star Schema*) [Red Brick 95].

A Figura 2.4 ilustra a organização básica de um esquema em estrela. Uma tabela central, a tabela de fatos, "rodeada" de tabelas secundárias, as tabelas de dimensão, uma para cada dimensão. Note-se neste esquema o uso das chaves estrangeiras das dimensões para compor a chave primária da tabela de fatos. Um fato numa tabela de fatos pode ser representado por diferentes medidas através de vários "atributos de fatos" (ex. unidades vendidas, valor em Real, etc.). As tabelas de dimensãos contêm atributos qualitativos que guardam características que descrevem cada componente da dimensão sob a perspectiva do negócio, ou seja, nas dimensões são armazenadas as características importantes para cada ramo do negócio (ex. para a dimensão produto : modelo, marca, embalagem, depto, etc.).

O próximo capítulo será dedicado inteiramente ao aprofundamento dos esquemas em estrela.

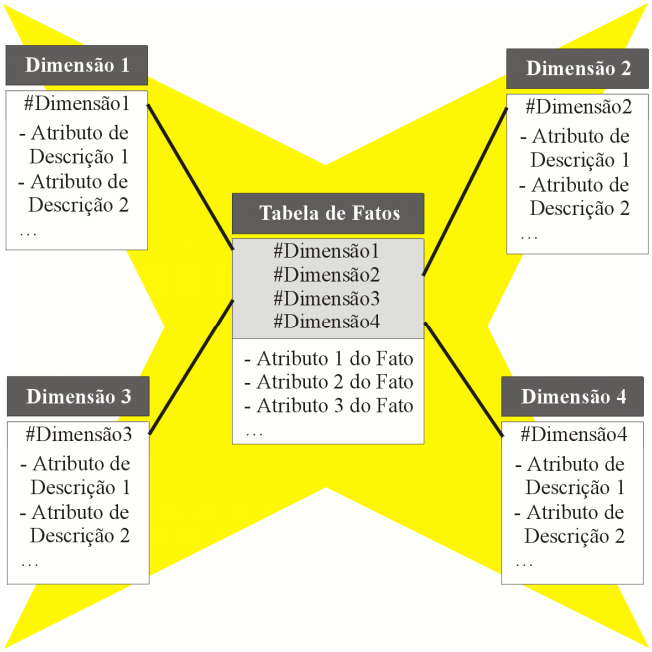


Figura 2.4 - Esquema em estrela [Souza 99].

Outras abordagens menos conhecidas para modelagem multidimensional são o HOLAP (*Hybrid* OLAP) [Alalouf 97], que junta características dos dois mundos, e o VTOLAP (*Vertical* OLAP), uma solução da Sybase IQ [Firestone 97].

2.3.3.3 Comparativo entre MOLAP e ROLAP

Escolher uma das duas estruturas para implementar um sistema OLAP depende da avaliação de vários parâmetros. Em [Firestone 97] vários parâmetros são descritos para avaliação de classes de produtos OLAP. Em [MicroStrategy 95] encontra-se um estudo sobre as vantagens de se usar ROLAP em vez de MOLAP. Este artigo também se propõe a desfazer alguns "mitos" sobre as limitações da tecnologia ROLAP.

Entretanto, uma rápida comparação entre as duas tecnologias mostra duas diferenças fundamentais. A primeira delas é que soluções baseadas em MOLAP são proprietárias, ficando o usuário restrito às alternativas oferecidas pelo fabricante, enquanto que soluções baseadas em ROLAP podem ser implementadas usando SGBD's relacionais, o que possibilita o uso de SQL padrão. A segunda, enquanto que ROLAP é adequado para a manipulação de grandes quantidades de dados, produtos MOLAP possuem essa capacidade limitada. A Tabela 2.1 resume mais algumas diferenças entre as duas tecnologias.

Soluções ROLAP são indicadas para sistemas que manipulam grandes conjuntos de dados e necessitam de alta escalabilidade. Para isso o desempenho é um pouco afetado. Isso, porém, é aceitável. Já soluções baseadas em MOLAP manipulam quantidades limitadas, mas possuem excelente performance.

Parâmetro	MOLAP	ROLAP
Volume de Dados	Limitado.	Admite grandes quantidades.
Flexibilidade estrutural	Modificações na estrutura levam à reconstrução do cubo e dos índices.	Tabelas são alteradas sem maiores problemas.
Desempenho das consultas	Rápido, baseado em índices.	Geralmente mais lento que MOLAP.
Universo da consulta	Limitado aos índices do modelo.	SQL permite ilimitadas possibilidades de consultas.
Tecnologia	Proprietária.	Independente de fabricante.
Padronização	Em andamento.	Relacional.

Tabela 2.1 - Principais pontos de divergência entre MOLAP e ROLAP [Souza 99].

2.3.4 Operações OLAP

A maioria das consultas OLAP é ad-hoc e algumas operações dessas consultas sobre as estruturas dimensionais são típicas e merecem atenção especial. Elas são chamadas de operações OLAP e são divididas em dois grandes grupos [Souza 99]:

- Análise Prospectiva : consiste em analisar partindo de um conjunto de dados chegar a conjuntos mais detalhados, menos detalhados ou outros conjuntos de dados [Inmon 97] [Souza 99].

  Operações :

  - Drill-Down : descer no nível de detalhes de uma dimensão;
  - Roll-Up : contrário de Drill-Down, ou seja, caminhar para a visão de dados mais resumidos;
  - Drill-Across : significa caminhar a partir de uma dimensão para outra dimensão, combinando-as para mudar o enfoque da análise.

- Análise Seletiva : objetiva selecionar pedaços do conjunto de dados para análise [Souza 99].

  Operações:

  - Slice&Dice : Em português, cortar e fatiar, seleciona pedaços transversais do cubo e em seguida aplica critérios de seleção sobre este pedaço.
  - Pivoting : Rotação. Muda a orientação dimensional para um relatório.

Adiante, veremos alguns exemplos :

Drill-Down :

É comum se falar que Drill-Down é executado em um conjunto hierárquico de atributos de dimensão, ou seja, numa hierarquia[7] [Kenan 95]. Entretanto, concordamos com [Kimball 98] que afirma que "Drill-Down em um data warehouse nada mais é que adicionar cabeçalhos de linha de tabelas de dimensão" em relatórios. Adicionar um cabeçalho a um relatório significa adicionar detalhes. Vejamos o exemplo da Tabela 2.2. Nesta tabela temos um relatório de unidades vendidas por loja para o mês de maio, segundo o cubo multidimensional da Figura 2.1. Estaremos fazendo Drill-Down se adicionarmos o cabeçalho Produto à Tabela 2.2. Esta tabela toma então a forma da Tabela 2.3.

A operação de Roll-Up neste caso consistiria simplesmente em partir da Tabela 2.3 para chegar à Tabela 2.2. Ou seja, subir para um menor nível de detalhe.

Mês	Loja	Unidades Vendidas
Maio 99	CGD	205
Maio 99	Souza	101
Maio 99	Patos	136
Maio 99	JPA	205
Maio 99	Monteiro	91

Tabela 2.2 - Relatório de unidades vendidas no mês de maio de 99 para o cubo dimensional da Figura 2.1.

Mês	Loja	Produto				
		Uno	Palio	Fiesta	Gol	Corsa
Maio 99	CGD	20	56	12	55	62
Maio 99	Souza	10	13	8	47	23
Maio 99	Patos	30	35	23	36	12
Maio 99	JPA	37	47	54	22	45
Maio 99	Monteiro	16	23	18	24	10

Tabela 2.3 – Adicionando o cabeçalho Produto à Tabela 2.2 descemos para um nível maior de detalhe.

Slice&Dice:

[7] Hierarquias serão detalhadas no Capítulo 3. Para agora basta-nos um exemplo de hierarquia dimensional : na dimensão produto é comum encontrarmos a hierarquia: chave de produto → tamanho da embalagem → marca → subcategoria → categoria → departamento [Kimball 98].

A operação Slice seleciona de um cubo multidimensional um corte transversal para análise. Vimos uma operação desse tipo na Figura 2.2, onde a gerência regional cortou o cubo para a análise da região de Monteiro. Agora, se fatiarmos esse "pedaço" do cubo para analisarmos apenas os modelos da marca Fiat (Uno e Palio), estaremos executando uma operação Dice (fatiar). Observe a Figura 2.5.

Pivoting:

A operação de Pivoting rotaciona os dados para que o usuário os enxergue sob outro ângulo. No cubo multidimensional esta operação consiste simplesmente de uma rotação de 90º na visão da estrutura. Um cubo de três dimensões, por exemplo, pode ser visto sob seis ângulos diferentes. Não há necessidade de nenhuma reestruturação dos dados. [Kenan 95].

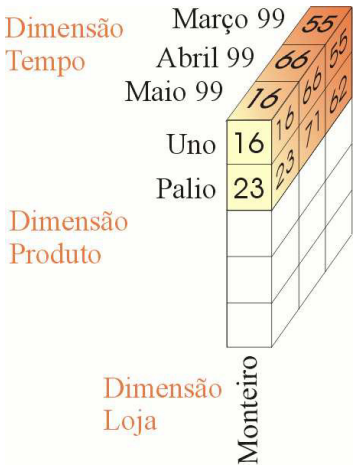


Figura 2.5 - Slice&Dice no cubo multidimensional da Figura 2.1.

Para a estrutura ROLAP, Kimball afirma em [Kimball 98] que rotação é a "reorganização de linhas e colunas de um relatório depois dele ter sido recuperado do DBMS". Ele também confirma a utilidade dessa operação por permitir um outro ângulo de visão sobre os dados. Aplicando uma rotação sobre a Tabela 2.3 obtemos a Tabela 2.4.

Mês	Produto	Loja				
		CGD	Souza	Patos	JPA	Monteiro
Maio 99	Uno	20	10	30	37	16
Maio 99	Palio	56	13	35	47	23
Maio 99	Fiesta	12	8	23	54	18
Maio 99	Gol	55	47	36	22	24
Maio 99	Corsa	62	23	12	45	10

Tabela 2.4 - Pivoting.

2.4 Data Warehousing

A atividade de análise está intimamente ligada ao problema de integração de múltiplos bancos de dados heterogêneos e distribuídos e outras fontes de dados [Widom 95]. Neste ponto, academia e indústria seguiram caminhos divergentes durante certo tempo. O problema da integração foi atacado inicialmente pela academia através da abordagem de *mediadores* [Widom 95]. Nesta abordagem, conhecida como *abordagem sob demanda*, a informação é extraída e integrada das diversas fontes somente quando a consulta é submetida. O problema desta abordagem é a performance[8]. O tempo de resposta das consultas é seriamente comprometido pelo processamento requerido para desmembrar a consulta em subconsultas, direcioná-las às respectivas fontes de dados, recolher os dados, integrá-los, filtrá-los e montar o conjunto resposta final. Num ambiente como este, consultas ad-hoc para atividade de análise tornam-se impraticáveis.

A alternativa da indústria é a *abordagem antecipada*. Nela a informação é extraída das fontes, integrada, filtrada e armazenada num repositório central, separado (lógica e fisicamente) dos sistemas operacionais e especialmente projetado para as necessidades de análise. Quando uma consulta é submetida, é executada diretamente sobre este repositório, sem necessidade de acesso a outras bases. Este repositório central é conhecido como ambiente projetado de data warehouse [Inmon 97] e esta abordagem como Data Warehousing [Widom 95].

[8] Um outro problema nesta abordagem é a impossibilidade de se fazer análise histórica. Isto porque as fontes operacionais geralmente só guardam o estado atual dos dados.

2.4.1 DWing e DW

Na literatura são encontradas muitas definições para o termo data warehouse (DW). Segundo [Inmon 97], "o data warehouse é um conjunto de dados baseados em assuntos, integrados, não-volátil, e variável em relação ao tempo, de apoio às decisões gerenciais". Esta definição é tida como clássica pela maioria dos estudiosos do assunto. [Wu&Buchmann 97] afirma que "um DW é um repositório de informações integradas de sistemas operacionais e sistemas legados[9] que provê os dados para o processamento analítico e para a tomada de decisão". Em [Widom&Yang 97], um DW "é um repositório para consultas e análises eficientes de informações integradas de uma extensa variedade de fontes". Para [POE et al 98] um DW "é um banco de dados analítico *read-only* que é usado como base para os sistemas de apoio à decisão".

Já Data Warehousing para [Chaudhuri&Dayal 97] "é uma coleção de tecnologias de suporte à decisão disposta a capacitar o gerente a tomar as melhores decisões de forma rápida". Segundo [Gardner 98], "Data Warehousing é um processo, não um produto, para montar e gerenciar dados vindos de várias fontes com o objetivo de prover uma visão simples, detalhada sobre parte de ou todo negócio".

Podemos notar que nenhuma das definições acima é exata e precisa. Isto porque uma definição completa requer a discussão de muitos pontos-chave dos ambientes de DWing [Gupta 97]. Entretanto, há uma concordância entre pesquisadores, vendedores e desenvolvedores sobre o data warehouse ser de interesse do 'apoio a decisões executivas' e conter dados históricos, sumariados e consolidados de várias fontes operacionais [Sen&Jacob 98].

Portanto, preferimos ao invés de tentar definir estes termos, discutirmos seus pontos-chave através do estudo das arquiteturas para DWing e do fluxo dos dados neste ambiente.

[9] O termo legado é usado para caracterizar os sistemas arquivados e em batch da Era da Automação.

2.4.2 Arquiteturas para DWing

Grosso modo, uma arquitetura define uma série de regras e estruturas que definem uma organização para o projeto de um sistema [POE et al 98], ou seja, ela descreve as funções e responsabilidades de cada parte integrante do sistema [Souza 99].

A literatura fornece uma variedade de arquiteturas para DWing. [Firestone 98] discute a evolução arquitetural dos DW's. Entretanto, a maioria das arquiteturas propostas para DWing aborda diretamente os aspectos físicos e estruturais deste tipo de sistema. Para o nosso propósito preferimos partir do estudo de uma arquitetura em camadas proposta em [Wu&Buchmann 97], para depois analisarmos as arquiteturas físicas mais comuns para DWing.

2.4.2.1 Arquitetura em Camadas para DWing

Segundo [Wu&Buchmann 97] a arquitetura de um Dwing depende de quatro fatores: volume de dados a serem armazenados, características dos dados, arranjo dos dados e o uso que será feito dos dados. Cada um desses fatores define uma camada para a arquitetura em camadas para DWing:

- Volume de dados → Camada de Administração de Dados : define formas eficientes para armazenamento e recuperação dos dados, construção de índices, *data clustering*, etc.

- Características dos Dados → Camada de Armazenamento de Dados : define as estruturas lógicas para armazenamento dos dados, executa a transformação entre os modelos de dados externos e a estrutura lógica do DW , executa o "tratamento" desses dados, é responsável pelo processamento e otimização de consultas, pela construção de agregados, etc.

- Arranjo dos Dados → Camada de Interface de Aplicação : executa o arranjo conceitual dos dados de acordo com as necessidades das

aplicações, converte o modelo lógico do DW para o modelo conceitual (ROLAP para MOLAP, por exemplo) requerido pela aplicação.

- Uso dos Dados → Camada de Apresentação : é a camada de acesso do usuário que inclui ferramentas OLAP, planilhas, relatórios, etc. Geralmente está presente no lado cliente.

Na Figura 2.6 encontra-se a representação gráfica da arquitetura descrita acima. note-se que, como em qualquer outra arquitetura em camadas, ela proporciona que alterações em qualquer camada sejam feitas sem que outras tenham que ser re-adaptadas, e todas as camadas fornecem serviços para as camadas imediatamente superior e inferior.

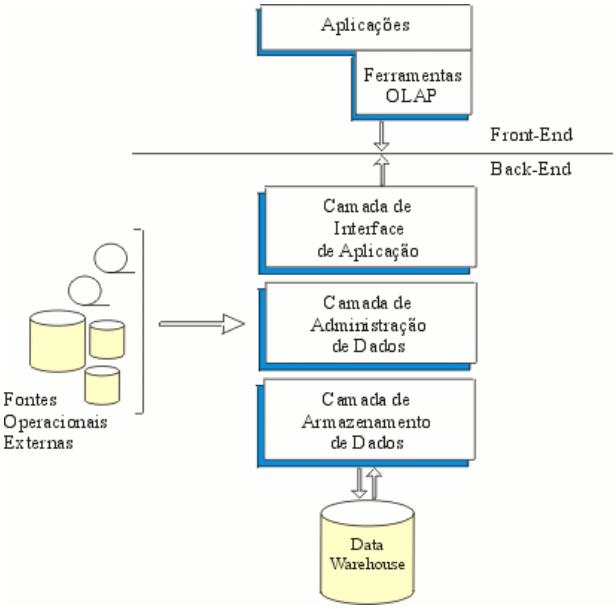


Figura 2.6 – Arquitetura em camadas para Data Warehousing [Wu&Buchmann 97].

Uma arquitetura em camadas semelhante à da Figura 2.6 foi proposta em [MicroStrategy 95]. Esta arquitetura reúne as Camadas de Administração e Armazenamento de Dados em uma única camada: Camada de Bancos de Dados, que agrega as funcionalidades das outras duas.

2.4.2.2 Arquitetura Funcional Básica

Uma arquitetura em camadas como a descrita acima tem muita utilidade na visualização das funcionalidades de cada camada do sistema. Entretanto, falta nesta arquitetura um melhor detalhamento dos componentes de um data warehouse bem como do ciclo de vida dos dados neste tipo de sistema. Através do detalhamento desses dois aspectos poderemos entender melhor o funcionamento geral de um ambiente de DWing. Por essas razões apresentamos na Figura 2.7 uma Arquitetura Funcional Básica para DWing.

Encontramos nesta arquitetura os principais componentes desses ambientes inseridos em retângulos sombreados. As setas indicam o fluxo dos dados entre os componentes.

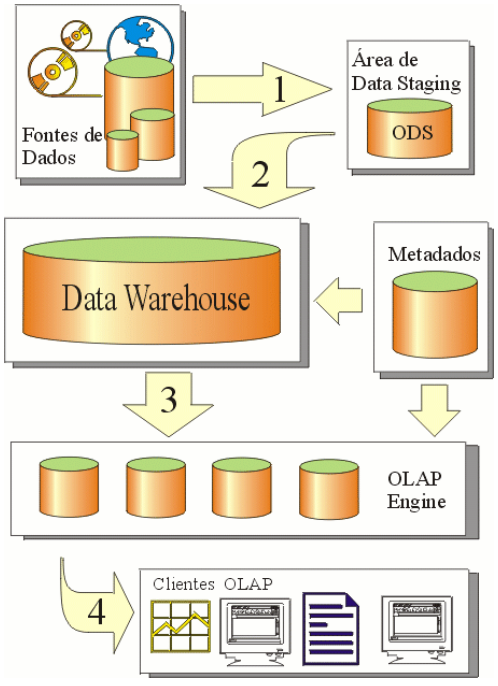


Figura 2.7 – Arquitetura funcional básica de um ambiente de DWing.

Componentes

Fontes de dados

É com os dados dessas fontes que o data warehouse é povoado. Diferentemente dos sistemas operacionais, onde as entradas de dados geralmente são feitas

diretamente pelo usuário, a entrada de dados do DW é feita a partir de dados dos sistemas operacionais e de fontes externas. As fontes de dados podem ser divididas em dois grupos :

- Fontes de Dados Internas : são os sistemas de automação do nível operacional. Os dados são provenientes de sistemas OLTP ou dos chamados *legacy systems* (em português : sistemas legados ou herdados), Os sistemas legados são os sistemas em batch ou arquivados da Era da Automação. Esses sistemas não utilizam processamento transacional e nem controle de concorrência[10].

- Fontes de Dados Externas : são fontes que não fazem parte do SI's da empresa. Os dados dessas fontes geralmente chegam através de redes públicas de comunicação. A Internet, por exemplo, é uma grande fonte de dados externa.

Área de Data Staging

É uma área de armazenamento intermediária entre as Fontes de Dados e o DW. Nesta área são feitos o tratamento, a integração e o controle de qualidade sobre os dados para que eles possam ser transferidos para o DW. Segundo [Kimball et al 98], os dados nesta área geralmente (mas não necessariamente) estão em terceira forma normal (3FN) e são não-históricos. Isto porque esta área não tem nenhum propósito de fornecer serviços de consulta e apresentação dos dados para análise. O repositório desta área também é conhecido como ODS (*Operational Data Store*) [Kimball et al 98] [Souza 99].

Não necessariamente existe apenas um repositório para a Área de Data Staging, porém, os vários que venham a existir devem ser integrados e homogêneos.

Data Warehouses

[10] Apesar de alguns autores como [Inmon 97] e [POE et al 98] não fazerem distinção entre *legacy systems* e sistemas transacionais, concordamos com [Souza 99] que afirma que *legacy systems* são tipicamente os velhos sistemas *batch* ou arquivados que não implementavam o conceito de transações ACID.

É o repositório central, integrado, histórico e orientado a assuntos. Dependendo da arquitetura física adotada para o ambiente de DWing, existem variações de modelos para construção do DW. Ele pode, por exemplo, ser construído em um MDDB com cubos multidimensionais, ou então em um SGBDR com esquemas em estrela, ou ainda num SGBDR em 3FN [Inmon 97] [Gardner 98].

OLAP Engine

Este componente está presente numa arquitetura baseada em ROLAP. Nela, dados do DW são extraídos para repositórios menores chamados Data Marts, que são DW's departamentais, menores em tamanho e orientados aos assuntos específicos do departamento. Geralmente são baseados em MDDB (podendo também ser baseados em esquemas em estrela). São hospedados em servidores de apresentação [Kimball et al 98], de onde os dados são diretamente manipulados por aplicações OLAP.

Metadados

Os Metadados têm um papel fundamental nos data warehouses. Os Metadados basicamente são informações sobre o que são e onde estão as coisas no DW [Inmon 97]. Há basicamente três classes de Metadados [Sen&Jacob 98] [Chaudhuri&Dayal 97] :

- Metadados Operacionais: que guardam informações estruturais sobre os dados no banco de dados, bem como mantêm informações estatísticas e de auditoria;

- Metadados Administrativos: mantêm informações necessárias ao correto uso do DW. Alguns aspectos abrangidos são: descrições sobre as fontes, dimensões, hierarquias, agregados, ferramentas de back-end e front-end, segurança, controle de acesso, consultas pré-definidas, histórico da extração, carga, transformação e controle de qualidade, o modelo de dados, etc.

- Metadados Corporativos : faz a ponte entre os conceitos lógicos e físicos do DW e os conceitos do negócio, facilitando a vida do usuário executivo.

Clientes OLAP

Generalizamos o termo clientes OLAP para caracterizar todas as aplicações que acessam os dados multidimensionais para análise. São as ferramentas utilizadas pelo corpo administrativo para análise gerencial. Estas aplicações incluem desde ferramentas OLAP complexas e poderosas até planilhas eletrônicas e geradores de relatórios. Entretanto, como essas últimas precisam "ter" a visão multidimensional da informação, elas precisam ter funcionalidades de ferramentas OLAP puras.

Fluxo de Dados

Num ambiente de DWing os dados seguem um fluxo determinado. Inicialmente esses dados precisam ser (1) integrados de varias fontes para posteriormente serem (2) estruturados, armazenados e (3) distribuídos para (4) análise.

Cada seta direcionada na arquitetura da Figura 2.7 é um estágio do fluxo de dados no DWing que agrega diversas atividades:

1 – Extração das várias fontes, armazenamento e integração na Área de Data Staging.

Uma vez na Área de Data Staging, os dados passam por diversas etapas de transformação as quais incluem [Kimball 98] [Kimball et al 98]:

- Integração e geração de novas chaves ;
- Adição do tempo;
- Checagem da integridade referencial para tabelas de fato e tabelas de dimensão;
- Desnormalização ou normalização;
- Conversão de tipos de atributos;
- Cálculos, derivações e alocações;
- Construção de Agregados;
- Tratamento de Valores Nulos.

2 – Carga dos dados no data warehouse.

Uma vez feita toda a transformação na Área de Data Staging os dados são carregados no DW. As principais atividades do processo de carga são:

- Particionamento de tabelas;
- Reconstrução de Índices;

- Controle de Qualidade;
- Publicação dos Dados.

3 - A partir do DW Corporativo os dados são carregados nos Data Marts.

O uso de Data Marts melhora o desempenho do sistema à medida que as consultas analíticas são divididas entre vários DW's departamentais, evitando assim a sobrecarga de consultas sobre o DW corporativo.

As principais atividades envolvidas nesta carga são:

- Separação dos dados por áreas funcionais;
- Transformação dos dados do modelo lógico corporativo para o modelo lógico departamental.

4 – Neste ponto o fluxo de dados é basicamente composto de conjuntos-resposta para os Clientes OLAP. A atividades incluem :

- Processamento e otimização de consultas;
- Geração de relatórios;
- Controle de acesso e segurança;

A manutenção de um DWing também inclui o processo de *refresh*, que consiste em propagar para o DW as atualizações feitas nos dados operacionais [Chaudhuri&Dayal 97]. Geralmente o processo de *refresh* é feito em *batch*, quando o sistema é desativado para consulta por algumas horas, e é periódico. Essa periodicidade define a granularidade, ou seja, o nível de detalhe em que os dados são armazenados no DW[11] [Kimball 98] [POE et al 98]. É recomendável que ele tenha granularidade mais fina possível, o nível atômico dos dados.

2.4.2.3 Arquiteturas Físicas

Há várias arquiteturas físicas propostas para DWing [Souza 99] [POE et al 98]. Discutiremos aqui as duas principais vertentes : arquitetura em duas camadas (*two-tier architecture)* e arquitetura em três camadas (*three-tier architecture*).

[11] É possível manter vários níveis de granularidade no ambiente de DWing [Inmon 97].

Arquitetura em Duas Camadas

- **OLAP multidimensional [MicroStrategy 95] [Wu&Buchmann 97] :**

Nessa arquitetura o MDDB agrega as funcionalidades de todas as camadas lógicas definidas na Seção 2.4.2.1. As consultas analíticas são executadas diretamente no MDDB. Sumários para todas as dimensões são computados para ganho de performance. Os dados no MDDB são carregados no menor nível de granularidade. Os principais problemas dessa arquitetura são o limite do volume de dados que o MDDB pode armazenar e a escalabilidade. A adição de novas dimensões ao cubo multidimensional acarreta, invariavelmente, a reconstrução total da estrutura física e dos índices.

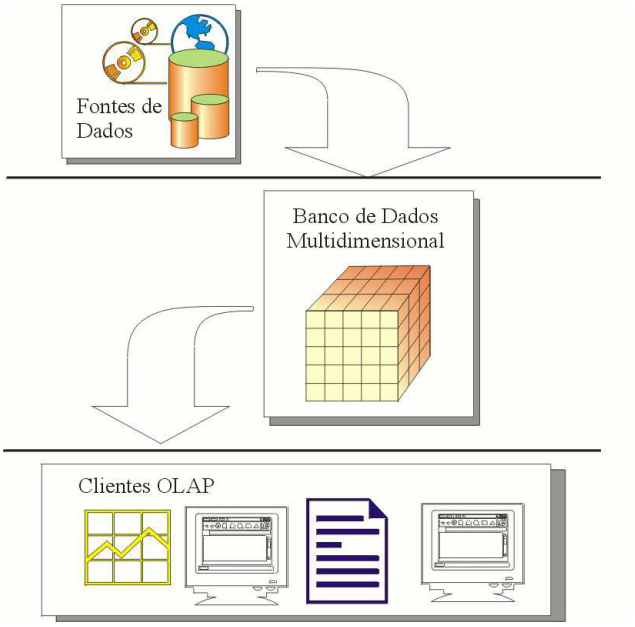


Figura 2.8 - Arquitetura OLAPmultidimensional.

- **Data Warehouse Federativo [Souza 99] [Kimball et al 98]**

"O data warehouse nada mais é do que a união de todos os Data Marts constituintes" [Kimball et al 98]. Esta arquitetura tem a enorme vantagem de possibilitar o desenvolvimento incremental (*bottom-up*) [Firestone 98] possibilitando

resultados práticos em menor tempo de desenvolvimento. O ponto-chave no desenvolvimento seguindo esta arquitetura é a identificação e modelagem das chamadas *Conformed Dimensions*[12] [Kimball et al 98]. Essas dimensões são comuns entre duas ou mais tabelas de fatos, portanto, podendo ser comum a dois ou mais Data Marts. Esquecer-se de considerar as *Conformed Dimensions* no projeto dos Data Marts pode gerar redundâncias que afetarão negativamente o projeto maior que é o do data warehouse Corporativo.

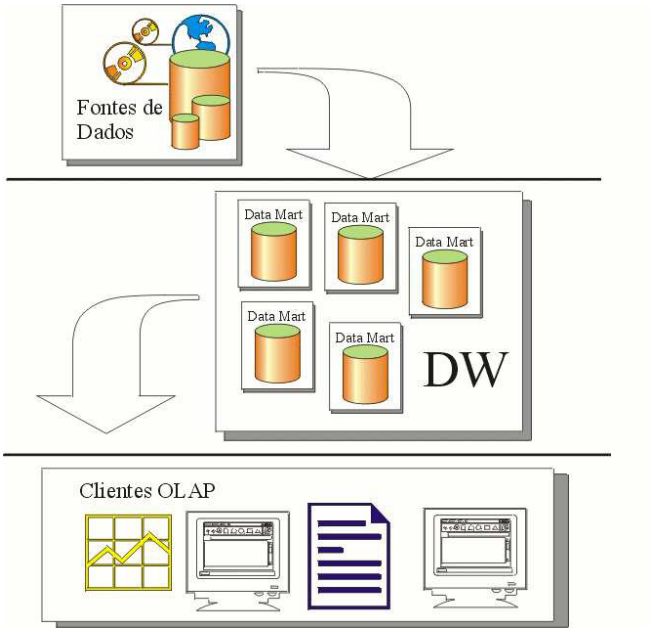


Figura 2.9 - Arquitetura de data warehouses federativos.

Arquitetura em Três Camadas

O objetivo principal dessa arquitetura é evitar a sobrecarga das atividades de análise sobre o DW corporativo. Essa responsabilidade é deixada para um nível intermediário: os *Data Marts*.

Esta arquitetura é equivalente à arquitetura funcional da Seção 2.4.2.2. Alguns autores defendem que o DW corporativo seja projetado em 3FN com a adição de atributos temporais para manutenção do histórico [Inmon 97] [Souza 99] [Gardner 98]. Os Data Marts seriam então projetados com estruturas multidimensionais como o cubo multidimensional ou o esquema em estrela.

[12] Por não encontrarmos uma boa tradução para o termo, novo por sinal, preferimos

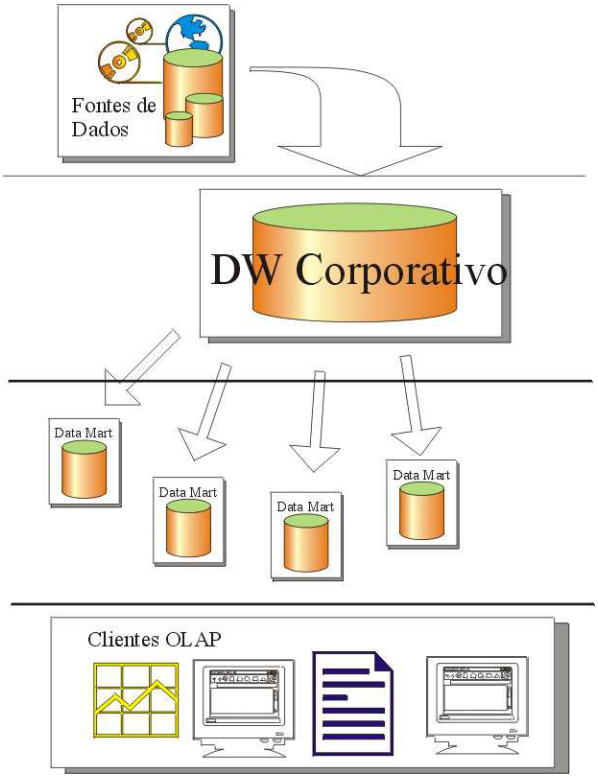


Figura 2.10 – Arquitetura em três camadas.

mantê-lo no idioma original.

Capítulo 3

Esquema em Estrela : Características e Limitações

3.1 Introdução

Apesar das muitas abordagens sobre arquiteturas e modelos para DW's, dois consensos existem na bibliografia consultada :

1 – O modelo de dados para análise gerencial deve ser multidimensional;

2 - Um DW é, em sua própria essência, um banco de dados histórico.

A modelagem multidimensional reflete o pensamento administrativo sobre os negócios. O desempenho de uma determinada atividade comercial é medido pelo cruzamento de informações dos vários setores ou componentes que juntos desenvolvem tal atividade. Os dois grandes conceitos envolvidos nesta perspectiva são os fatos e as dimensões. Sendo as dimensões as entidades lógicas representativas das áreas organizacionais envolvidas numa atividade. Entidades únicas de um ramo do negócio geralmente são fragmentadas em várias entidades lógicas num modelo E-R. O preço pago por essa fragmentação é que os sistemas de produção (OLTP) geralmente são difíceis de se consultar [Kimball et al 98]. Temos que considerar ainda o fato de que esses sistemas estão distribuídos entre vários departamentos que os implementam cada um com seu esquema e em sua plataforma específica. A modelagem multidimensional proporciona a visão corporativa integrando esses sistemas e modelando os dados segundo uma perspectiva da gerência e não do departamento de SI.

Além disso, ao contrário do processamento operacional, a análise gerencial necessita não só dos dados atuais da empresa, mas também de dados históricos. Gerentes analisam seus negócios comparando desempenhos atuais com desempenhos

em meses anteriores, ou em trimestres anteriores ou em anos anteriores, etc. [Kimball 98]. Bancos de dados OLTP geralmente não implementam nenhum tipo de suporte temporal, salvo poucas exceções. Eles refletem o estado atual do negócio. Entretanto, DW's são inerentemente históricos, isto é, guardam informações atuais e do passado. As tabelas de fatos são séries temporais de medições do negócio [Kimball 98] [Kimball et al 98] [Poe et al 98]. Informações armazenadas neste contexto não podem ser alteradas pura e simplesmente como fazemos nos sistemas de produção. O DW deve ser um banco de dados *append-only*, ou seja, informações são apenas inseridas, sendo alterações e exclusões consideradas somente sob algumas regras rígidas pré-determinadas. Num DW, vale o princípio de que todo fato ou objeto existe ou existiu em algum tempo passado, sendo assim, as informações sobre essa existência devem ser de alguma forma armazenadas. Desta feita, não podemos simplesmente excluí-las como se elas nunca tivessem existido.

O foco deste trabalho está justamente nos dois aspectos discutidos acima. Nosso estudo sobre esquemas em estrela para DW's revelou que, apesar dessa estrutura ser no geral adequada, ela falha em alguns aspectos concernentes ao histórico. Um esquema em estrela efetivamente guarda o histórico de fatos do negócio nas tabelas de fatos. Os fatos são guardados sob a forma de instantâneos (*snapshots*) medidos de acordo com a granularidade do DW [Kimball 98] [Kimbal et al 98] [Inmon 97]. Entretanto, casos reais mostram que o fator "histórico" num DW não se resume aos fatos, pois também as dimensões possuem um histórico intrínseco que precisa ser guardado.

Neste capítulo estudaremos mais detalhadamente esquemas em estrela analisando algumas de suas particularidades bem como variações em sua estrutura típica. Por fim, apresentaremos o problema principal concernente ao histórico que encontramos em esquemas em estrela e que nos propomos a resolver neste trabalho.

3.2 Esquema em Estrela em Detalhes

Entidades[13] do mundo real num modelo E-R comumente são fragmentadas em várias entidades lógicas e mapeadas em várias tabelas relacionais. Essa fragmentação é

[13] Ou objetos do mundo real.

resultado justamente do objetivo de se eliminar redundâncias. Isso, contudo, transforma um esquema E-R corporativo numa estrutura com centenas de entidades, difícil de memorizar pelo usuário e difícil também de se consultar, pela comum ocorrência de várias ligações possíveis (relacionamentos) entre entidades distantes [Kimball et al 98]. Além disso, entidades no modelo E-R são pares idênticos, ou seja, um mesmo conceito da modelagem é utilizado para modelar objetos de naturezas diferentes, sem nenhuma diferenciação semântica entre eles. Por exemplo, no modelo E-R, Vendas é modelado identicamente a Produto, a Fornecedor, a Vendedor, etc. Segundo [Inmon 97], "por uma série de razões, as entidades do mundo de Data Warehousing são qualquer coisa menos pares idênticos".

Um esquema em estrela é um modelo assimétrico. A tabela central, de fatos, geralmente possui tamanho muitas vezes superior às tabelas secundárias, de dimensão. As entidades centrais são medidas de fatos do negócio. As entidades secundárias são descrições dos ramos do negócio envolvidos nos fatos. Um esquema em estrela típico para o setor varejista de supermercados pode ser visto na Figura 3.1.

Para entendermos essa figura, precisamos de algumas convenções:

1 – Cada tabela é representada por um retângulo escuro, que contém o nome da tabela, e um retângulo sombreado, que contém uma lista de atributos que compõe as colunas da tabela.

2 – As chaves de cada tabela são codificadas com um '#' no final. Ex. Produto#, Loja#, etc.

3 – Os relacionamentos de chave estrangeira entre tabelas são representados por linhas.

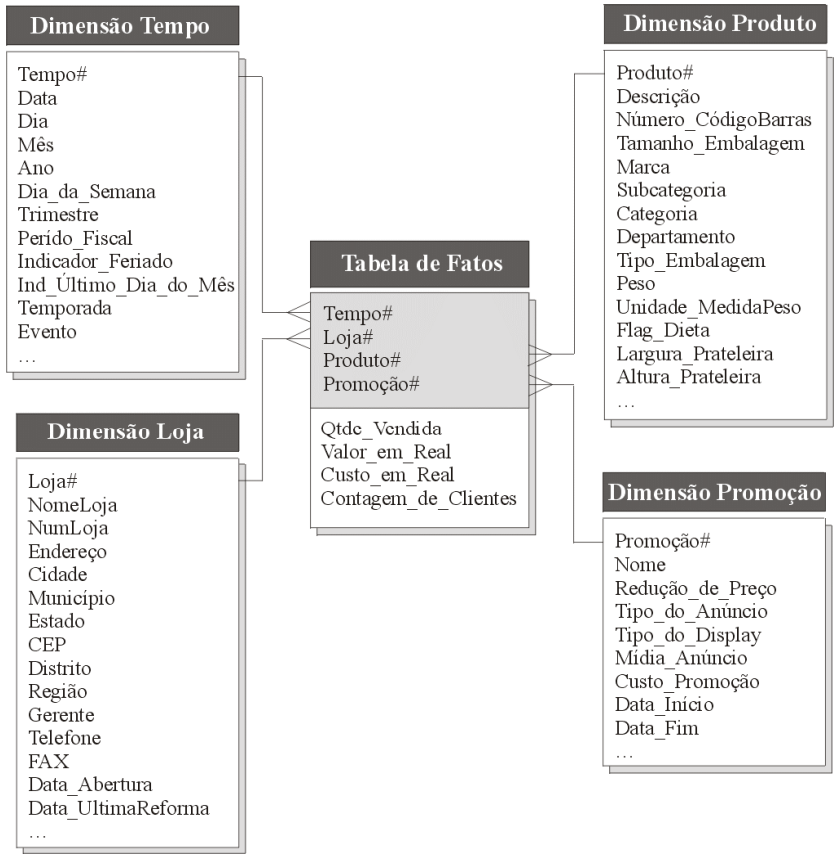


Figura 3.1 – Esquema em Estrela para Vendas no Varejo [Kimball 98].

A seguir analisaremos alguns detalhes importantes dos esquemas em estrela tomando o exemplo da Figura 3.1 como ilustração.

3.2.1 Tabela de Fatos

A tabela central, chamada tabela de fatos (do inglês *fact table*) contém os fatos ou medições do negócio. Seus atributos não-chaves geralmente são valores numéricos aditivos, medidos pelo cruzamento das dimensões do negócio. Quando esquemas em estrela são modelados a partir de modelos E-R do nível operacional [Golfarelli et al 98], fatos podem ser identificados em entidades que são muito atualizadas ou relacionamentos N-N. No exemplo acima, temos três atributos aditivos: *Qtde_Vendida*, o número de unidades vendidas de determinado **produto,** durante um determinado período (**tempo),** em uma determinada **loja** sob determinada oferta (**promoção**); *Valor_em_Real*, a soma em dinheiro desta venda; *Custo_em_Real*, o custo desta quantidade de produtos; e Contagem_de_Clientes, que analisaremos mais adiante. A chave da tabela de fatos é composta pelo conjunto de chaves das dimensões que

participam da atividade. A chave da tabela de fatos Vendas é composta por: Tempo#, Produto#, Loja# e Promoção#.

Note-se que a tabela de fatos na verdade é uma seqüência de instantâneos do negócio. Os instantâneos são criados a partir da ocorrência de algum evento. Num ambiente de negócios, um evento pode ser uma venda, uma compra, a entrega de um pedido, um pagamento, etc., ou seja, um evento é uma atividade discreta de qualquer natureza que merece algum tipo de registro (o instantâneo). Mas um evento também pode ser a passagem do tempo. Por exemplo, o final do dia, o final da semana, um feriado, etc. [Inmon 97]. Sob essa perspectiva, a principal atividade de um DWing é o registro de eventos do negócio.

Eventos podem ser registrados através de atributos aditivos ou não. Kimball [Kimball 98] classifica os fatos de um negócio em aditivos, semi-aditivos e não-aditivos. Os fatos aditivos podem ser somados em relação a qualquer dimensão. Já os fatos semi-aditivos podem ser somados em relação a algumas dimensões, porém, não todas. E os fatos não-aditivos não podem ser somados de jeito nenhum, são usados apenas para o registro de eventos. Um exemplo de fato semi-aditivo é o atributo Contagem_de_Clientes da tabela de fatos da Figura 3.1. Este atributo armazena a quantidade de Clientes que comprou um determinado Produto, num determinado Dia, em uma Loja sob determinada Promoção. Este atributo não pode ser somado em relação à dimensão Produto. Imagine que em um dia 20 Clientes compraram o Produto A e 30 compraram o Produto B. Não há como sabermos a quantidade exata de Clientes que comprou um e a quantidade que comprou o outro.

O registro de instantâneos em um DW pode ser feito desde o nível mais detalhado da informação até o nível mais resumido. Um certo nível de detalhe é conhecido como **granularidade** do DW [Inmon 97] [Kimball 98] [Kimball et al 98] [Poe et al 98]. No exemplo da Figura 3.1, a granularidade da tabela de fatos é o dia. Isso significa dizer que estamos registrando as vendas diárias de determinado produto. Um nível mais detalhado seria, por exemplo, o registro de cada transação de venda efetuada no PDV (Ponto de Venda, caixa). Um nível mais resumido poderia ser o registro das vendas semanais. O nível de granularidade do DW depende de vários fatores [Inmon 97], entretanto é recomendável que o DW tenha dados “expressos no nível de ‘menor grão’ de cada dimensão” porque as consultas de análise “precisam se

aprofundar no banco de dados de maneira muito precisa" [Kimball 98]. Porém, é possível, e recomendável, manter-se vários níveis de granularidade [Inmon 97]. Isso freqüentemente é feito através da computação de agregados [Kimball 98].

3.2.2 Tabelas de Dimensões

As dimensões[14] contêm atributos que caracterizam cada ramo do negócio envolvidos em determinado fato. É importante para a perfeita compreensão do usuário analista (tomador de decisão) que os nomes e os valores dos atributos sejam tão claros quanto possível, sem abreviações e sem nenhum tipo de codificação, comumente usada nos sistemas operacionais [Kimball 98].

As chaves primárias das dimensões são formadas por um único atributo. Kimball [Kimball 98a] [Kimball 98b] recomenda que essas chaves sejam artificiais (*surrogate keys*), uma generalização das chaves naturais dos sistemas de produção. Ao contrário das chaves codificadas dos sistemas de produção, essas chaves não devem possuir nenhuma semântica porque a codificação de chaves de nada serve ao usuário executivo[15]. Em verdade, a única função das chaves primárias é a junção de tabelas de dimensão com tabelas de fatos[16]. Outras razões para o uso de chaves generalizadas são a economia de espaço e a rapidez para o processamento de junção. Uma chave comum dos sistemas operacionais em média tem tamanho entre 12 e 20 bytes. Num DW, uma chave composta de um inteiro de 4 bytes é capaz de representar uma dimensão de até 2^{32} linhas, ou seja, mais de 2 bilhões de linhas. Portanto, inteiros de 4 bytes são uma excelente escolha para chaves de dimensão.

Kimball também afirma que as dimensões são ortogonais [Kimball 98], ou seja, não há nenhum tipo de relacionamento entre elas. Realmente a maioria das dimensões não tem nenhum tipo de vínculo ou dependência. Porém, veremos mais adiante, que algumas dimensões têm uma relação de dependência com a dimensão Tempo. Em verdade, a maioria das dimensões tem esse vínculo.

---

[14] Usaremos a partir deste ponto no texto os termos Dimensão e Tabelas de Dimensão como sinônimos.

[15] A codificação de chaves primárias para conter semântica deveria ser evitada até nos sistemas operacionais, já que valores de chaves deveriam ser invisíveis aos usuários.

[16] Apesar de serem usadas para guardar histórico de dimensões como veremos no Capítulo 4.

3.2.2.1 A Dimensão Tempo

Uma dimensão particular merece atenção especial no esquema em estrela da Figura 3.1: Tempo. A inserção da temporalidade nos dados de um DW é feita através da dimensão Tempo. O registro dos eventos na tabela de fatos tem um marcador de tempo que indica a ocorrência da tal evento no mundo real : a chave da dimensão Tempo. Isso certamente poderia ser feito através da simples inserção de um atributo "tempo" na tabela de fatos. Por exemplo, em vez de se criar a dimensão Tempo e inserir sua chave na tabela de fatos, poderíamos incluir um atributo tempo nesta tabela com o domínio DATE [Bliujute et al 98]. Esta é uma solução válida se as consultas analíticas precisam que os eventos sejam registrados apenas com relação a dias, meses e anos. Porém, muitos usuários executivos analisam os dados dividindo-os em períodos fiscais, dias úteis, fins-de-semana, temporadas, feriados, etc. [Kimball 98]. A solução simplista de incluir um atributo tipo DATA na tabela de fatos não oferece este tipo de semântica.

Pelas razões acima, a maioria dos DW's mantém uma (ou mais) dimensão(ões) Tempo. Uma típica dimensão Tempo aparece na Figura 3.1. Esta dimensão, assim como as outras, possui uma chave e atributos descritores. A chave geralmente é um inteiro incrementado de forma seqüencial. Cada dia na dimensão tem sua chave que é o inteiro imediatamente posterior à chave do último dia registrado. Cada linha da tabela representa um "átomo" de tempo, ou seja, o menor nível de granularidade em que os eventos são registrados no DW.

3.2.2.2 Hierarquias

É comum encontrarmos em dimensões atributos que mantêm entre si um relacionamento 1-N. Esses atributos formam as chamadas hierarquias de dimensões [Kimball 98] [Kimball et al 98] [Poe et al 98] [Thomsen 97]. Dois exemplos de hierarquias podem ser vistos na Figura 3.2.

Na Figura 3.2 (a) temos uma hierarquia de produtos. Nesta hierarquia fica claro que um departamento contém várias categorias, que contém várias subcategorias, que contém várias marcas e assim por diante.

Figura 3.2 – (a) Hierarquia de Produto. (b) Hierarquia de Tempo.

Hierarquias são particularmente úteis na análise dos dados em diferentes níveis de detalhe. Usuários executivos freqüentemente analisam dados sob diferentes perspectivas, caminhando na hierarquia, executando *drill-down* e *roll-up*. Alguns autores [Poe et al 98] [Thonsem 97] consideram as operações de *drill-down* e *roll-up* como a atividade de descer e subir em detalhes na hierarquia, respectivamente. Entretanto, concordamos com Kimball [Kimball 98] que afirma que tais operações OLAP não são executadas necessariamente apenas em hierarquias, já que um cabeçalho (coluna) a ser adicionado a um relatório não precisa necessariamente pertencer a uma.

3.2.3 Snowflake Schemas

Freqüentemente preocupados em economizar espaço de armazenamento, nós, como projetistas, tendemos a estranhar esquemas em estrela como o da Figura 3.1. Note-se que neste esquema, a única tabela realmente normalizada é a tabela de fatos [Kimball 98]. As dimensões possuem um alto nível de redundância em seus registros, principalmente pela presença das hierarquias.

A tendência da maioria dos projetistas é normalizar as dimensões para diminuir a redundância. Ao normalizar um esquema em estrela estaremos criando uma variação dessa estrutura conhecida como s*nowflake schema* (em português conhecido como Esquema em Flocos de Neve). A normalização desmembra dimensões em várias

tabelas ligadas por chaves estrangeiras. A dimensão produto da Figura 3.1 pode ser desmembrada na dimensão da Figura 3.3.

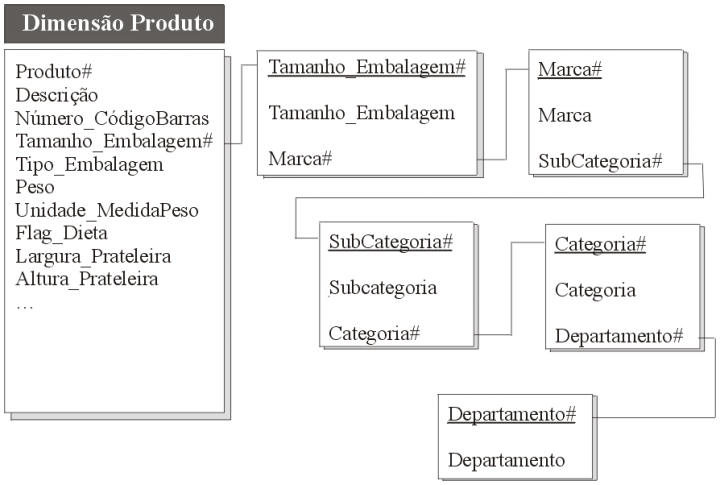


Figura 3.3 – Dimensão Produto desmembrada.

Há uma grande rejeição à normalização de dimensões, apesar de existirem casos em que é necessária. Certamente essa normalização afetará o desempenho de pesquisas, pois aí haverá a necessidade de junções. O custo-benefício do desmembramento de dimensões certamente não encoraja essa atividade. Segundo [Kimball 98], "qualquer estimativa realista do espaço em disco necessário para o data warehouse pode efetivamente ignorar as tabelas de dimensão". Isso porque proporcionalmente as tabelas de fatos são muitas vezes maiores do que qualquer dimensão. Em geral, o desmembramento de dimensões produz uma economia de espaço de menos de 1% (um por cento) [Kimball 98], o que definitivamente não justifica a perda de desempenho em consultas que isso acarreta.

Alguns autores defendem o uso de normalização em DWing com arquitetura em três camadas [Souza 99] [Gardner 98]. Segundo eles, o DW corporativo deveria ser construído no modelo relacional em 3FN com a adição do fator tempo aos dados. As estruturas dimensionais (esquema em estrela e cubo multidimensional) deveriam ser utilizadas apenas no projeto de Data Marts. Alguns desses autores baseiam-se em [Inmon 97]. Entretanto, Inmon afirma nesta referência que "a estrutura de projeto necessária para gerenciar grandes quantidades de dados residentes em uma entidade contida no data warehouse é denominada *star join*", e mais adiante, "a modelagem de dados clássica se aplica às tabelas de dimensão (ou seja, às entidades pouco povoadas)

e o projeto de *star join* se aplica às tabelas de fatos (ou seja, às entidades muito povoadas)". Podemos perceber por essas afirmativas que Inmon defende apenas que sejam usados s*nowflake schemas* para o projeto do DW Corporativo, e não o uso de modelos relacionais em 3FN. Afirmar que um DW deve ser construído em 3FN não necessariamente significa que não devemos construí-lo com esquemas em estrela, já que um *snowflake schema* pode ser totalmente normalizado e mesmo assim continuará sendo uma estrutura dimensional.

3.2.4 Browsing

Uma seção típica de análise geralmente se inicia com o usuário executivo navegando em tabelas de dimensões para conhecer os relacionamentos entre seus atributos e desta forma descobrir cabeçalhos e restrições para seus relatórios [Kimball 98] [Kimball 98 et al]. Essa navegação é chamada de *browsing*.

Numa consulta *browsing* são aplicadas restrições em atributos de uma dimensão para observação dos valores de outros atributos na presença dessas restrições. Comumente são feitas navegações em hierarquias, porém não só hierarquias são válidas para a aplicação das restrições. Consultas *browsing* geralmente são construídas com o comando SQL DISTINCT que seleciona valores distintos de um conjunto resposta.

Uma típica seção de *browsing* é mostrada nas Tabelas 3.1, 3.2 e 3.3. Obviamente que essas consultas devem ser feitas através de interfaces gráficas adequadas ao usuário executivo, portanto imaginemos que as tabelas estão sendo mostradas em um *browser*. Inicialmente o usuário executivo escolhe os atributos que quer consultar e monta seus cabeçalhos. Agora ele seleciona os departamentos da dimensão produto (SELECT DISTINCT Departamento FROM Produto;) (Tabela 3.1). Posteriormente o usuário executivo seleciona (restringe) com um clique do mouse o Departamento Laticínios (SELECT DISTINCT Marca FROM PRODUTO WHERE Departamento='Laticínios') (Tabela 3.2). A tabela 3.2 mostra a atual configuração do *browser*. Em seguida o usuário clica na marca Nestlé . Todos os produtos Nestlé do departamento de laticínios são mostrados na coluna Descrição (SELECT DISTINCT Descrição FROM Produto WHERE MARCA='Nestlé') (Tabela 3.3). Neste ponto o, usuário executivo já deve ter despertado curiosidade em alguns desses produtos. Ele

agora pode construir seu relatório final arrastando campos de outras dimensões e da tabela de fatos para analisar medidas de desempenho.

Departamento	Marca	Produto
Laticínios Padaria Congelados Limpeza Higiene Pessoal ...		

Tabela 3.1 – SELECT DISTINCT Subcategoria FROM Produto;

Departamento	Marca	Produto
Laticínios	*Nestlé* Parmalat Batavo Danone ...	

Tabela 3.2 - SELECT DISTINCT Marca FROM PRODUTO WHERE Subcategoria='Laticínios';

Departamento	Marca	Produto
Laticínios	*Nestlé*	Iogurte Morango Requeijão Cremoso Queijo Tipo Parmesão Iogurte Cenoura-Laranja ...

Tabela 3.3 - SELECT DISTINCT Descrição FROM PRODUTO WHERE Marca='Nestlé';

Conhecendo agora a atividade de *browsing*, entendemos melhor porque a normalização de tabelas de dimensão afeta diretamente o desempenho do sistema. As consultas acima foram realizadas sem nenhum tipo de junção. As junções serão necessárias apenas quando forem efetuadas análises com a tabela de fatos. Acontece que consultas *browsing* representam 80% (oitenta por cento) da atividade de análise [Kimball 98]. Portanto, estamos evitando com a desnormalização uma perda de desempenho em 80% das atividades de análise executadas num ambiente de DWing.

3.3 Dimensões que Mudam com o Tempo

Os únicos relacionamentos existentes num esquema em estrela são aqueles entre as tabelas de fatos e as tabelas de dimensão [Red Brick 95] [Kimball 98]. Sendo assim, nós podemos pensar que as dimensões são ortogonais entre si [Kimball 98] (prova disso é a disposição das dimensões no cubo multidimensional), em outras palavras, não há nenhum tipo de relacionamento entre elas. Por outro lado, notamos na maioria dos DW's a presença da dimensão Tempo, porque esses repositórios armazenam grandes quantidades de dados históricos [Widom 95]. E se as dimensões são ortogonais entre si, não havendo nenhum tipo de relacionamento entre nenhumas delas, somos levados a acreditar que seus dados são estáticos, imutáveis desde o momento de sua carga, o que significa dizer que as dimensões não têm nenhum tipo de vínculo com o tempo. Este definitivamente não é o caso. As dimensões evoluem ao longo do tempo. Tanto sua estrutura (novas colunas) quanto seus dados passam por modificações. A tarefa de *refresh* [Wu&Buchmann 97] [Widom 95] é, além de atualizar as tabelas de fatos, refletir no DW essas modificações, afinal os dados desses repositórios são derivados dos sistemas operacionais que são constantemente atualizados.

Vejamos alguns exemplos :

Exemplo 1:

A dimensão Loja da Figura 3.1. Podemos perceber que alguns dados nesta dimensão são praticamente imutáveis. Por exemplo, Data_Abertura, Estado, Município, Cidade, Distrito, etc. Outros, todavia, podem sofrer alterações nos sistemas operacionais que precisam ser refletidas no DW. Por exemplo, o Telefone e o FAX podem mudar, o Gerente pode ser substituído, o NomeLoja pode ser alterado, uma loja pode sofrer reformas, etc.

Exemplo 2:

Alguns DW's mantêm uma dimensão Cliente. Uma dimensão Cliente típica está representada na Figura 3.4. Note-se nesta dimensão uma grande quantidade de atributos pré-dispostos a alterações : Profissão, Estado_Civil, Escolaridade, Nível_de_Renda, Num_Dependentes, etc. São comuns dimensões Cliente em alguns

ramos de negócio com milhões de linhas. Portanto, essas dimensões são altamente voláteis.

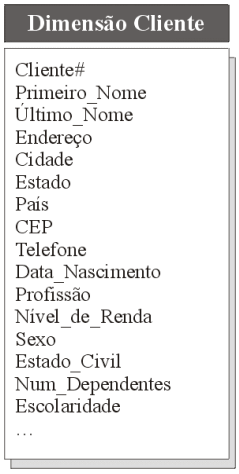


Figura 3.4 – Dimensão Cliente típica.

Exemplo 3:

Na dimensão Produto são comuns mudanças em hierarquias de produtos. Um produto pode "saltar" de uma hierarquia à outra com muita freqüência [Pimenta 98].

Inúmeros outros exemplos de dimensões que mudam com o tempo podem ser encontrados na prática. DW's para bancos mantêm dimensões para Contas que são altamente voláteis, DW's para seguros mantêm dimensões para Segurados e Reclamações que são também bastante voláteis, etc. [Kimball 98].

Os exemplos acima provam que grande parte das dimensões (senão todas) tem um relacionamento direto com a dimensão Tempo. Se não todos os atributos de uma dimensão são variáveis no tempo, então pelo menos um sub-conjunto desses atributos é. Além disso, como já falamos anteriormente, "uma das principais responsabilidades do data warehouse é representar de forma correta o histórico passado" [Kimball 98]. No entanto, o único histórico a ser representado pelo DW não é o histórico dos fatos, pois *as dimensões de um negócio possuem às vezes um histórico intrínseco que precisa ser guardado*, ou seja, *as tabelas de dimensão de um DW podem ser históricas.*

Os atributos de uma dimensão podem ser classificados da seguinte forma:

- Atributos não-voláteis : não são alterados nunca. Exemplo: Cliente : Nome, Sexo, Data_Nascimento, etc.

- Atributos voláteis : sofrem alguma alteração no decorrer do tempo. Podem ser :

    - Não-históricos : sua alteração não representa nenhuma influência nas atividades do negócio e, portanto, seu histórico não precisa ser guardado. Exemplos : Dimensão Loja : Telefone, FAX, Endereço, CEP, etc.

    - **Históricos** : sua alteração pode representar impactos em atividades do negócio, sendo assim, devemos guardar seu histórico exato de alguma maneira. Exemplos : Dimensão Loja : Gerente; Dimensão Cliente : Estado_Civil, Nível_de_Renda, Escolaridade, etc.

Alterações em valores de determinados atributos podem refletir mudanças em fatos do negócio. Por exemplo, a mudança de um gerente numa loja pode afetar as vendas, um gerente pode ter um perfil mais dinâmico ou agressivo do que seu antecessor e com isso aumentar as vendas de uma loja; clientes têm hábitos consumistas diferentes dependendo de seu estado civil; mudanças em embalagens de produtos podem afetar suas vendas, etc. Todas essas mudanças são de interesse do usuário executivo e precisam ser refletidas no DW de alguma forma.

O esquema em estrela em sua forma clássica não fornece nenhum ferramental para guardar valores de atributos históricos. Se um gerente é substituído por outro numa loja há apenas uma alteração simples no atributo, perdendo-se seu valor antigo. Perdemos também, nesse caso, a possibilidade de rastrearmos as vendas dos gerentes anteriores para compararmos com as vendas do atual e assim analisarmos se foi válida essa troca. Se não guardarmos convenientemente valores de atributos históricos de clientes, não poderemos traçar perfis de comportamento de grupos de clientes ao longo do tempo. Se não guardarmos os valores dos atributos históricos de produtos, não poderemos analisar os impactos causados por determinadas mudanças.

Guardar históricos de dimensões abre espaço para um novo tipo de análise : análise histórica. O histórico de atributos dimensionais deve ser rastreado de forma exata, consistente e de fácil entendimento para o usuário executivo. Além disso, não deve causar nenhum impacto significativo no desempenho do ambiente de DWing. Considere-se aí a capacidade de armazenamento e o processamento de consultas.

O histórico das dimensões contém, assim como os fatos, semântica de interesse do usuário executivo. É preciso que a temporalidade dos atributos dimensionais seja tratada de forma correta e precisa. Esta não é apenas uma decisão do projetista, mas sim um aspecto que tem relevância na análise multidimensional como um todo. Particularmente nos esquemas em estrela esse ponto não tem sido tratado com a importância que tem. A maioria das soluções propostas é decisão de projeto que atende a casos particulares. Uma solução geral, completa, satisfatória e formal não foi encontrada em toda a bibliografia pesquisada. Este é o objetivo que nos propomos atingir neste trabalho.

Capítulo 4

Trabalhos Correlatos

4.1 Introdução

O problema do histórico de dimensões não tem sido satisfatoriamente abordado na literatura apesar de sua relevância. Como vimos no Capítulo 3, o histórico dos fatos não é o único que deve ser guardado no esquema em estrela. Atributos dimensionais históricos podem ter grande importância na análise dimensional. Esses atributos representam o porquê de certas mudanças e tendências em fatos do negócio. Não guardar o histórico significa perder a capacidade de descobrir fatores influentes no desempenho de determinadas atividades. Porém, esse tema tem despertado pouco interesse entre os pesquisadores do assunto.

Outros aspectos temporais dos DW's têm despertado crescentemente o interesse de pesquisadores [Widom 95] [Devlin 97]. O foco dos trabalhos acadêmicos tem sido a manutenção de visões materializadas com aspectos temporais [Musicant&Dabu 97] [Widom&Yang 97]. Por outro lado, projetistas de DW têm enfrentado várias situações em que o histórico de atributos de dimensões têm que ser mantido de alguma forma. Usuários de DW's exigem em várias ocasiões o mapeamento histórico desses atributos para realização de consultas temporais, principalmente comparações entre dados referentes aos valores antigos e dados referentes aos valores atuais. A maioria dos trabalhos sobre esse tema tem um enfoque prático, são alternativas de projetos para viabilizar situações específicas que surgem a partir de exigências dos usuários analistas.

Ralph Kimball foi o pioneiro neste tema [Kimball 98] [Kimball et al 98]. Ele classifica as dimensões que possuem atributos históricos como Dimensões de Modificação Lenta (do inglês : *Slowly Changing Dimensions*) e propõe várias técnicas

para resolução desse problema, as quais discutiremos na Seção 4.2.1. Muitos projetistas de DW e até algumas ferramentas comerciais implementam suas idéias [Netz 99]. Porém, reconhecendo as limitações de sua própria abordagem para determinados tipos de dimensões, Kimball propôs outras alternativas: minidimensões demográficas e *timestamping* de dimensões muito grandes [Kimball 98] [Kimball et al 98].

Criticando as idéias de Kimball encontramos apenas um trabalho acadêmico: [Bliujute et al 98]. Neste trabalho é proposto um *Temporal Star Schema* que difere do esquema em estrela tradicional em alguns aspectos. Na Seção 4.3 discutiremos com mais detalhes essa abordagem.

A relevância do tema *histórico de dimensões* pôde ser melhor compreendida através da lista de discussão por e-mails dwlist@datawarehousing.com[17]. Nesta lista, projetistas de DW do mundo inteiro discutem os mais variados temas sobre Data Warehousing. Discute-se desde aspectos conceituais e arquiteturais até aspectos de implementação e ferramentas disponíveis. Nos anos de 1998, 1999 e 2000 surgiram várias discussões sobre o tema *dimensões que mudam com o tempo*. Nessas discussões foi possível analisar as técnicas propostas e utilizadas por diversos autores para resolver o problema. Compilamos na Seção 4.4 algumas dessas alternativas que julgamos serem as mais viáveis.

Nas seções seguintes deste capítulo analisaremos as técnicas para guardar o histórico de dimensões compiladas das fontes citadas. Essa análise se dará através da aplicação da técnica proposta em exemplos de dimensões. Os dois exemplos que usaremos serão a dimensão Loja da Figura 3.1 e a dimensão Cliente da Figura 3.4.

Nossa análise tem o objetivo de mostrar como os modelos propostos até agora falham nos requisitos que consideramos fundamentais para guardar o histórico de atributos de dimensões. Tais requisitos são:

1. A história de cada atributo histórico deve ser guardada de forma consistente e exata, ou seja, não devemos aceitar perda de informação

[17] Para inscrever-se nesta lista mande um e-mail para o endereço dwlist-request@datawarehousing.com com o comando subscribe no corpo da mensagem.

temporal e nem lapsos de tempo entre o que aconteceu realmente e o que está registrado no DW;

2. Devemos evitar redundâncias e conseqüentemente desperdício de espaço de armazenamento[18];

3. O histórico completo de um valor de um atributo (ou valores de um conjunto de atributos correlatos: entidade semântica (ver Seção 5.3.2)) deve ser representado em uma única entidade lógica, no nosso caso, um registro;

4. O esquema utilizado deve ser facilmente entendido também pelo usuário executivo;

5. A solução deve permitir análise histórica (inclusive browsing temporal) com bom desempenho;

6. Todas as características e variações do esquema em estrela original devem ser mantidas intactas bem como as operações típicas realizadas sobre esse esquema (browsing, drill-down, roll-up, etc.), ou seja, queremos apenas uma extensão do esquema original, facilmente adaptável aos esquemas já existentes.

4.2 Propostas de Ralph Kimball

4.2.1 Dimensões de Modificação Lenta (Slowly Changing Dimensions (SCD's))

Kimball considera que "a maioria das dimensões é constante ao longo do tempo e que podemos preservar uma estrutura dimensional independente" [Kimball 98]. Para ele, mesmo havendo um relacionamento entre a dimensão Tempo e as outras dimensões, o caminho para a solução não passa por colocar tudo dependente do

[18] Apesar da maioria das dimensões ser muito menor do que as tabelas de fatos, algumas dimensões, como a dimensão Cliente de uma telefônica, por exemplo, podem chegar facilmente a milhões de registros. Nestes casos devemos certamente evitar dados redundantes.

tempo, pois isso levaria à criação de modelos complexos como os tradicionais modelos Entidade-Relacionamento.

Ele propõe três alternativas para "capturar o aspecto temporal das modificações". São as chamadas Dimensões de Modificação Lenta (SCD's) Tipo 1, Tipo 2 e Tipo 3. Examinaremos cada uma delas utilizando como exemplo a dimensão Loja. O objetivo é guardar o histórico do atributo Gerente. Uma vez alterado o valor desse atributo no sistema operacional, o novo valor deve ser refletido no DW e os valores antigos desse atributo devem ser guardados.

4.2.1.1 SCD Tipo 1

Nas SCD's do tipo 1, a alteração do atributo Gerente da dimensão Loja é direta e simples como uma alteração comum nos sistemas OLTP. Observe a Figura 4.1.

Loja#	Nome_Loja	Endereço	...	Gerente	...
1	CGD	R. 13 de Maio		Jonas	
2	JPA	R. 21 de Abril		João	
3	NAT	R. 11 de Outubro		David	
…	…	…	...	...	...

Loja#	Nome_Loja	Endereço	...	Gerente	...
1	CGD	R. 13 de Maio		Carlos	
2	JPA	R. 21 de Abril		João	
3	NAT	R. 11 de Outubro		David	
…	…	…	...	...	...

Figura 4.1 – SCD Tipo 1 : alteração simples do atributo gerente da Loja 1.

Esse tipo de alteração é válido para atributos dimensionais não-históricos cujos valores antigos não precisam ser guardados, já que sua mudança não causa nenhum impacto significativo no negócio (ex: telefone, endereço, CEP, etc.). Esse também é o caso para alterações decorrentes de entradas de dados incorretas. Neste caso corrige-se o valor do atributo normalmente. Note-se que aqui não se guarda nenhum histórico, portanto, não serve ao nosso propósito.

4.2.1.2 SCD Tipo 2

Para efetivamente guardar o histórico dos atributos dimensionais, Kimball propõe uma técnica, que aplicada sobre a dimensão Loja funciona da seguinte forma:

- Inclui-se na tabela de dimensão um novo registro com o novo valor do atributo histórico Gerente, os valores antigos de todos os outros atributos da dimensão e uma nova chave, neste caso, generalizada com dígitos de versão.

- A partir deste momento, todos os registros incluídos na tabela de fatos referentes às vendas da Loja alterada devem conter a chave do novo registro da Loja.

Observe a Figura 4.2. Note-se que a partir do tempo 5 a chave da Loja 1 passa a ser 1b. Esta abordagem permite a coexistência de várias versões de um mesmo atributo. Numa consulta, se fizermos uma restrição sobre o atributo Gerente, teremos no conjunto resposta exatamente um registro com a versão da Loja que nos interessa, e se a restrição for aplicada a qualquer outro atributo teremos como resposta vários registros com todas as versões da Loja [Kimball et al 98]. Isso é útil em casos como a coexistência de um mesmo produto com duas embalagens diferentes numa mesma Loja, por exemplo. Os dois continuam a serem vendidos e os fatos são registrados para cada versão do produto [Kimball 98].

Analisando mais profundamente esta técnica, podemos notar que verdadeiramente ela particiona *os fatos* referentes a cada Gerente, e não o histórico do atributo em si. Para rastrearmos o histórico do atributo é preciso uma junção com a tabela de fatos para detecção da data de tal alteração, ou seja, o histórico do atributo está registrado na tabela de fatos. Isso gera alguns problemas. Primeiramente não concordamos que a tabela de fatos deva ser utilizada para registrar histórico de atributos dimensionais (pois sua função é registrar fatos). Estas tabelas são várias vezes maiores que as tabelas de dimensão, o que prejudica o *desempenho de consultas* para rastreamento do histórico. Segundo, *os valores antigos são perfeitamente armazenados, mas não o histórico stricto-sensu*. Um exemplo: um cliente casou. Certamente este fato é relevante aos negócios. Logo que foi registrado o casamento, o atributo Estado_Civil

foi alterado no sistema operacional e esta alteração foi propagada para o DW, gerando a inclusão de um novo registro com a chave generalizada. Todavia, um registro na tabela de fatos só vai ser incluso para este cliente quando ele efetuar uma transação qualquer. E se depois de casar-se ele passar três meses, um ano, um ano e meio sem efetuar nenhuma transação com a companhia? A data registrada na tabela de fatos não refletirá a data exata (e nem aproximada) de seu casamento. A tabela de fatos é esparsa! O leitor pode afirmar: "Mas essa data é de interesse do usuário executivo?". Pode ser! O usuário analista pode, por exemplo, querer descobrir porque as vendas de determinados produtos caem em determinado mês todos os anos e a resposta pode estar na quantidade de casamentos que se realizam sempre neste mês. Ele pode também querer traçar o perfil do comportamento de clientes logo após seus casamentos.

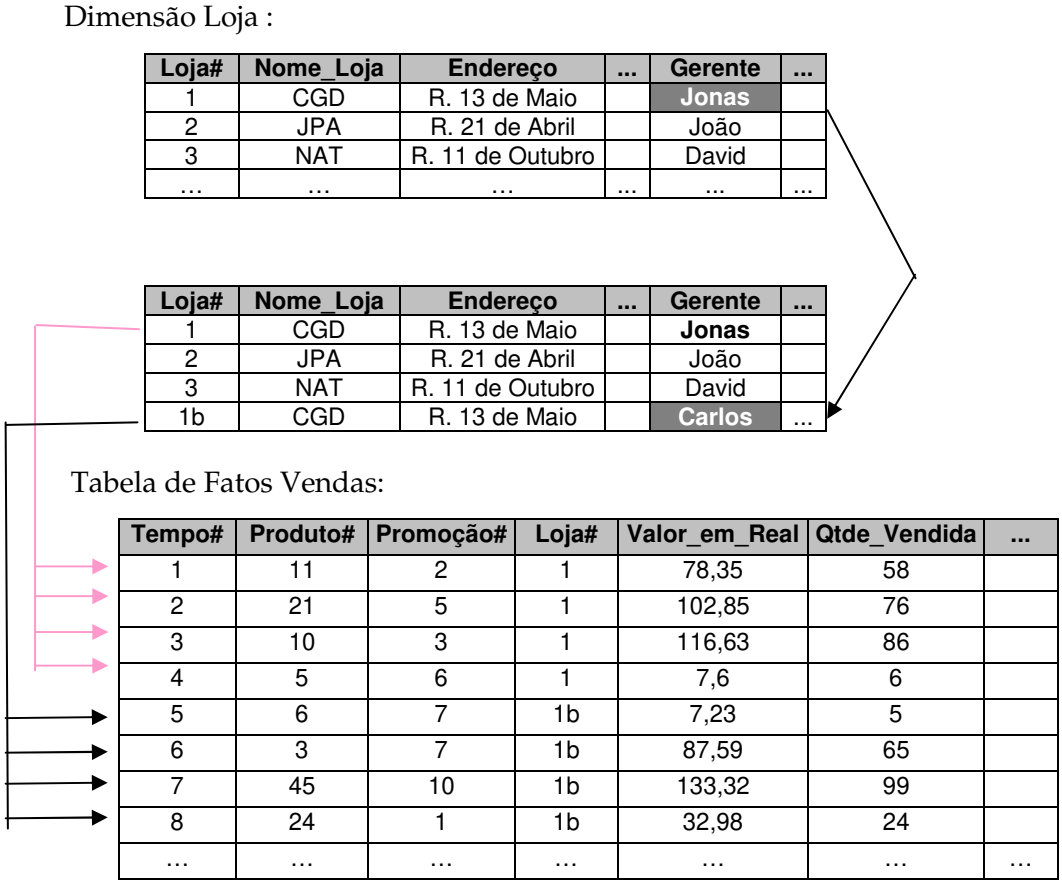


Figura 4.2 – SCD Tipo 2 : Um novo registro com uma chave generalizada com dígito de versão é incluído na dimensão.

Em [Kimball 98], Kimball afirma que "não há necessidade de incluir uma data de efetivação e absolutamente não há necessidade de criar restrições de tempo para a

tabela dimensional para obter uma resposta certa". Porém, já em [Kimball et al 98] ele admite o uso de "uma data de efetivação e uma data de fim no registro". O uso destas datas resolve os problemas citados acima. Elas são inclusas como dois atributos na dimensão: Data_Efetivação e Data_Fim, e indicam o período de tempo em que o registro foi ou continua sendo válido.

Dois outros problemas surgem com o uso de SCD's Tipo 2. Eles se manifestam quando esta técnica é utilizada com dimensões muito grandes e não tão constantes ao longo do tempo. O primeiro deles é a inclusão de um novo registro na dimensão sempre que houver alterações em atributos. Uma dimensão Cliente típica tem entre 50 e 100 atributos e em média 10 milhões de registros [Kimball et al 98]. Se 30% dos atributos sofre algum tipo de alteração pelo menos três vezes em um ano, teremos nesse período um acréscimo de 9 milhões de registros, ou seja, a tabela praticamente dobra de tamanho. Neste caso, SCD's Tipo 2 acarretam *um desperdício muito alto de espaço de armazenamento*.

O segundo problema é o uso de versionamento de chaves para dimensões. O versionamento é feito com a inclusão de um ou dois dígitos de versão na chave. Sempre que uma alteração for efetuada ou uma consulta sobre dados atuais for submetida é preciso antes descobrir qual a última versão do registro. No pior caso será preciso efetuar consultas ao longo de todas as versões na dimensão. No melhor caso a manutenção de chaves generalizadas pode ser feita através de metadados, o que *sobrecarrega* o sistema de qualquer maneira.

4.2.1.3 SCD Tipo 3

SCD's Tipo 3 são usadas em situações diferentes das SCD's Tipo 2. Kimball afirma que elas são usadas para registrar "mudanças leves" [Kimball et al 98]. Por exemplo, mudança de categoria de um produto, mudança de definição de distrito de uma Loja, etc. Servem para situações em que é preciso rastrear valores antigos e novos de atributos. Para isso inclui-se na tabela de dimensão um novo atributo para conter o valor original do atributo modificado e uma data de efetivação da alteração. Observe a Figura 4.3. Note-se que neste esquema não é necessário incluir um novo registro com uma nova versão da chave. A data de efetivação serve para particionar os fatos de cada

gerente. Fatos anteriores a 11/09/1998 são referentes a Jonas, posteriores a esta data são referentes a Carlos.

Loja#	Nome_Loja	...	Endereço	Gerente_Original	Gerente	Data_Efetivação
1	CGD		R. 13 de Maio		Jonas	
2	JPA		R. 21 de Abril		João	
3	NAT		R. 11 de Outubro		David	
…	…	…	…	...	...	...

Loja#	Nome_Loja	...	Endereço	Gerente_Original	Gerente	Data_Efetivação
1	CGD		R. 13 de Maio	Jonas	Carlos	11/09/1998
2	JPA		R. 21 de Abril		João	
3	NAT		R. 11 de Outubro		David	
…	…	…	…	...	...	...

Figura 4.3 – SCD Tipo 3 : Dois valores para Gerente : antigo e atual; e uma data de efetivação da alteração.

No caso de ocorrer mais uma alteração nesse atributo, o valor original é mantido, o valor atual é alterado e a data de efetivação é atualizada. No nosso exemplo perderíamos o valor Carlos. Isso significa dizer que podemos rastrear apenas dois níveis do histórico, o valor original e o valor mais atual [Kimball 98] [Kimball et al 98], o que é extremamente ineficiente para a maioria dos casos de atributos históricos.

Outro problema que surge com essa técnica é que, caso a dimensão possua muitos atributos suscetíveis a "mudanças leves", seremos obrigados a manter vários atributos antigos e atuais. Isso é extremamente deselegante, além de ser difícil de manter e compreender! Por exemplo, para a dimensão Produto poderíamos ter: Categoria_Original e Categoria_Atual, Sub-Categoria_Original, Sub-Categoria_Atual, Tamanho_Embalagem_Original, Tamanho_Embalagem_Atual, etc.

4.2.2 Dimensões Demográficas

Admitindo o problema da redundância causada pela SCD Tipo 2 na manutenção do histórico de dimensões muito grandes, Kimball propõe uma outra técnica: dimensões demográficas [Kimball 98] [Kimball et al 98]. O nome não caracteriza a técnica em si, mas sua aplicação em uma dimensão particular: Cliente. Como esta técnica foi pioneiramente aplicada nesta dimensão, este nome passou a caracterizar o caso geral.

Na dimensão Cliente alguns atributos servem principalmente a estudos estatísticos (comportamento, perfil, etc.) sobre a coletividade dos clientes, são os chamados atributos demográficos. Esses atributos possuem uma volatilidade relativamente alta, seus valores são atualizados com certa freqüência. A técnica SCD Tipo 2 aplicada a essa dimensão gera muita redundância.

Atributos demográficos são reunidos numa única dimensão, chamada dimensão demográfica. A Figura 4.4 mostra a dimensão Cliente da Figura 3.4 desmembrada em duas dimensões, sendo uma delas a dimensão demográfica. O histórico dos atributos demográficos para cada cliente pode ser rastreado associando-se as duas dimensões através da tabela de fatos. Para cada novo registro da tabela de fatos inclui-se a chave do cliente e a chave de seu perfil demográfico atual.

Uma dimensão demográfica é "povoada" com a combinação de todos os valores possíveis dos seus atributos. Entretanto, o domínio de alguns atributos é muito vasto, o que nos obriga a restringi-los a faixas de valores. Por exemplo, Renda e Crédito são atributos cujos domínios são contínuos. Agrupando valores em faixas (por exemplo: entre 0 e 1000, entre 1001 e 2000, etc.) estaremos construindo um domínio discreto. A dimensão pode então ser "povoada" com a combinação de todos os valores discretos. Para nosso exemplo, suponhamos que o domínio de Profissão tenha 30 valores, Nível_de_Renda: 10 faixas de valores, Estado_Civil: 4 valores, Num_Dendendentes: 5 valores, Escolaridade: 10 valores e Faixa_de_Crédito: 10 faixas de valores. A dimensão demográfica terá 30x10x4x5x10x10 = 600.000 registros.

Essa abordagem apresenta alguns problemas:

- O uso de faixas de valores restringe consultas a níveis mais resumidos de dados. A individualidade de cada cliente, no nosso exemplo, é perdida;
- Uma vez definidas, as faixas não podem ser redefinidas;
- Algumas dimensões demográficas são enormes e precisam ser divididas em duas ou mais mini-dimensões demográficas;

- Consultas *browsing* que relacionam dados das dimensões e dados demográficos têm que fazer junção das duas tabelas através da tabela de fatos. Resultado: *péssimo desempenho*;

O histórico, assim como nas SCD's tipo 2, é rastreado através da tabela de fatos. Apesar de eliminarem o problema do desperdício de espaço de armazenamento, as dimensões demográficas também podem conter históricos inconsistentes por causa da esparsidade da tabela de fatos. A mudança no valor de algum atributo demográfico de algum cliente só é registrada na tabela de fatos se o cliente efetuar alguma transação que deva ser registrada nesta tabela. Kimball [Kimball et al 98] propõe que a alteração seja registrada inserindo-se na tabela de fatos um registro com valores de fatos nulos. Funciona, mas é deselegantíssimo ! Além de causar erros em consultas que utilizam o operador SQL COUNT.

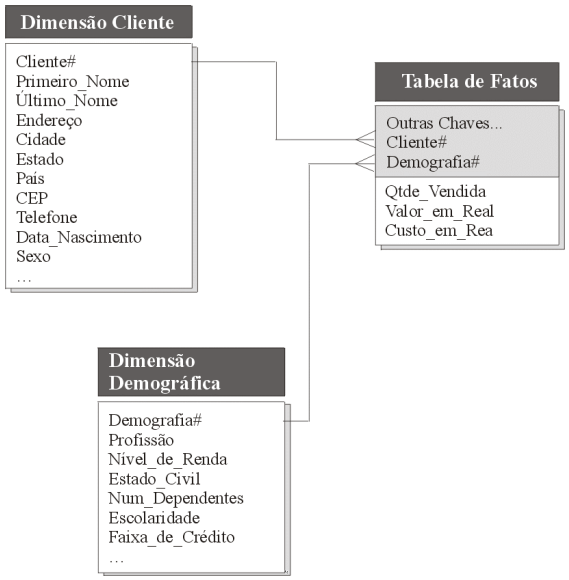


Figura 4.4 – Dimensões Demográficas.

Para contornar o problema do *browsing*, Kimball [Kimball 98] [Kimball et al 98] sugere que seja posto um atributo chave estrangeira na tabela de dimensão com o valor da chave primária correspondente da dimensão demográfica, criando assim um *link* entre a duas tabelas. Porém, o histórico continua sendo rastreado pela tabela de fatos, pois o link aponta apenas para os dados demográficos mais atuais do cliente.

4.3 Temporal Star Schema

O único artigo técnico que encontramos sobre o tema [Bliujute et al 98] propõe um *Temporal Star Schema* (TSS). Segundo este artigo, os dados no DW podem ser orientados a estados ou orientados a eventos. A representação correta das mudanças de estados é a responsável principal pelo tamanho do DW e pelos problemas de performance de consultas.

Segundo [Bliujute et al 98], dados orientados a estados podem ser representados num esquema em estrela sob duas formas: séries temporais [Klein et al 99] ou eventos. O principal problema das séries temporais é encontrar um granularidade correta que seja um meio termo entre a possibilidade de manter informações incompletas e manter altos níveis de redundância. Tomemos por exemplo o saldo de uma conta bancária. Se decidirmos por uma granularidade de 1 (uma) hora poderemos estar guardando informações incompletas, pode haver muitas transações neste espaço de tempo. Se decidirmos pela granularidade de 1 (um) minuto, estaremos possivelmente guardando muita informação redundante. A representação por eventos resolve este problema porque registramos o estado apenas quando o balanço da conta é alterado. Entretanto, representação por eventos apresenta muitos problemas de performance em consultas. Estes problemas são discutidos mais profundamente em [Devlin 98]. [Bliujute et al 98] afirma que dados orientados a estados são melhor representados por intervalos. O intervalo indica o período de tempo em que o estado foi válido.

O TSS difere dos esquemas em estrela tradicionais em seu tratamento do tempo. Ele propõe que o tempo não seja tratado como uma dimensão à parte. Ao invés de manter uma dimensão em separado é melhor incluir *timestamps* (ver Seção 6.2.2) em todas as tabelas de esquema. Assim, teríamos *timestamps* de intervalos para as tabelas cujos dados são orientados a estados, como as dimensões e algumas tabelas de fatos; e *timestamps* de instantes para tabelas cujos dados são orientados a eventos, que é a maioria das tabelas de fatos. Um exemplo dessas tabelas pode ser visto na Figura 4.5.

Dimensão Loja

Nome_Loja	...	Endereço	...	Gerente	Inicio	Fim
CGD		R. 13 de Maio		Jonas	12/09/96	10/10/20
JPA		R. 21 de Abril		João	13/04/95	25/08/98
JPA		R. 21 de Abril		Carlos	26/08/98	10/10/20
NAT		R. 11 de Outubro		David	12/06/92	25/08/95
NAT		R. 11 de Outubro		João	26/08/98	24/10/99
NAT		R. 11 de Outubro		Manoel	25/10/99	10/10/20
…	…	…	...	...	...	

Fatos Vendas

Data	Produto#	Promoção#	Valor_em_Real	Qtde_Vendida	...
12/06/92	11	2	78,35	58	…
13/04/95	21	5	102,85	76	…
25/08/95	10	3	116,63	86	…
25/08/95	5	6	7,6	6	…
12/09/96	6	7	7,23	5	…
12/09/96	3	7	87,59	65	…
…	…	…	…	…	…

Fatos Saldo_de_Contas

Inicio	Fim	Cliente	Valor
12/06/92	10/10/00	1	1.000
13/04/95	25/08/98	1	1.500
25/08/95	10/10/00	2	2.300
25/08/95	25/08/95	3	4.000
12/09/96	24/10/99	4	1.000
12/09/96	10/10/00	5	800
…	…	…	...

Figura 4.5 – Exemplos de tabelas no TSS.

A idéia geral desta solução é interessante, porém identificamos alguns problemas:

- Segundo [Bliujute et al 98], podemos eliminar a dimensão tempo e deixar a complexidade de manipulação do tempo para a linguagem de consulta. Discordamos dessa proposta por dois motivos:

  1. Isto significa que teremos que construir nosso DW em algum SGBD temporal. Porém, não há nenhum SGBD temporal comercial confiável o suficiente para se construir uma aplicação tão crítica quanto DWing;

2. A manipulação dos dados temporais do DW estará restrita às funcionalidades oferecidas pelo modelo e pela linguagem adotados pelo SGBD Temporal;

3. As possibilidades de se modelar tempos especiais (feriados, períodos fiscais, etc) na dimensão tempo desaparecem;

Em outras palavras o segundo e o terceiro motivo querem dizer que perdemos a flexibilidade oferecida pela manutenção da dimensão Tempo. Manter esta dimensão significa manter um calendário totalmente flexível para o DW (ver Seções 5.4 e 5.4.2.1).

- Para as dimensões que mudam com o tempo, esta solução apresenta os mesmos problemas de redundância das SCD's Tipo 2, ou seja, ela não é aplicável em dimensões muito grandes.

Alguns pontos-chave dos TSS's não são discutidos no artigo, como a manutenção das chaves do esquema e as funcionalidades temporais requeridas pela linguagem de consulta.

4.4 Outras Abordagens

Na lista de discussões por correio eletrônico dwlist@datawarehousing.com surgiram várias discussões sobre o tema do nosso trabalho nos anos de 1998, 1999 e início de 2000. Esta lista é composta por projetistas, administradores e pesquisadores de DWing. Os temas discutidos são os mais variados, vão desde plataformas de hardware e software até problemas conceituais. Dimensões que mudam com o tempo foi um dos temas mais debatidos nos últimos dois anos. Muitos participantes recorreram à lista solicitando soluções para o problema do histórico de atributos de dimensões. Alguns sugeriram alternativas e esperaram as críticas de outros participantes para incrementar suas propostas. Um dos participantes dessa lista é o próprio Ralph Kimball.

Nas discussões sobre o tema em questão, quase uma dezena de técnicas alternativas foi proposta por diversos autores. Grande parte delas são variações das

SCD's de Kimball. Outras, no entanto, mostraram-se bastante originais e atrativas. Nas próximas seções discutiremos em detalhes algumas dessas alternativas.

Apesar de não ser uma fonte bibliográfica puramente técnica ou científica, destacamos a importância dessa lista de discussões para o nosso trabalho. Ela serviu sobremaneira para compararmos nossa solução com o que mais está sendo feito por outros profissionais da área. Também serviu para mostrar que o histórico de dimensões é um problema importante, não-trivial, e que não deve ser tratado apenas como uma alternativa de projeto porque ele está intimamente ligado à natureza histórica do DW.

4.4.1 Extensão à SCD Tipo 2

[Kanfonas 98] propõe uma variação das SCD's Tipo 2 de Kimball. Nesta proposta, a chave da dimensão é composta de dois atributos : VID (*version identifier*, identificador de versão) e TID (*token identifier*, identificador de chave). Também dois atributos do tipo DATE são inclusos para identificar o período de validade do registro : Início e Fim. Veja a Figura 4.6. As duas chaves da dimensão são incluídas na tabela de fatos. Assim, consultas referentes à determinada versão do registro da Loja são feitas usando a comparação Loja.VID = Vendas.VID AND Loja.TID = Vendas.TID. Já consultas envolvendo todas as versões da Loja usam a comparação Loja.TID = Vendas.TID, com o período podendo ser restringido através das datas Início e Fim.

Note-se que esta técnica é apenas um melhoramento das SCD's Tipo 2. A proposta continua usando versionamento, só que neste caso a versão é marcada num atributo separado da chave. O mesmo problema de redundância se mantém. Porém, o rastreamento do histórico fica independente da tabela de fatos pela inclusão das datas que indicam o período de validade da versão do registro.

Dimensão Loja

TID	VID	Nome_Loja	...	Endereço	...	Gerente	Inicio	Fim
1	1	CGD		R. 13 de Maio		Jonas	12/09/96	10/10/00
2	1	JPA		R. 21 de Abril		João	13/04/95	25/08/98
2	2	JPA		R. 21 de Abril		Carlos	26/08/98	10/10/00
3	1	NAT		R. 11 de Outubro		David	12/06/92	25/08/95
3	2	NAT		R. 11 de Outubro		João	26/08/98	24/10/99
3	3	NAT		R. 11 de Outubro		Manoel	25/10/99	10/10/00
…		…	…	…	...	...	...	

Fatos Vendas

Tempo#	Produto#	Promoção#	TID	VID	Valor_em_Real	Qtde_Vendida	...
1	11	2	1	1	78,35	58	…
2	21	5	1	1	102,85	76	…
3	10	3	2	1	116,63	86	…
4	5	6	2	2	7,6	6	…
5	6	7	2	2	7,23	5	…
6	3	7	3	1	87,59	65	…
…	…	…	…	…	…	…	…

Figura 4.6 – Dimensão com uma chave de registro e uma chave de versão.

4.4.2 Tabela de Fatos Histórica

[Sanders 98] afirma que uma mudança no valor de um atributo é um tipo de fato. Tal fato deve ser registrado numa tabela de fatos sem fatos [Kimball 98] como a da Figura 4.7. Neste caso, a tabela de fatos registra o evento: *alteração de valor de um atributo dimensional*. Note que esta tabela guarda as alterações de todos os atributos de todas as dimensões. O atributo Dimensão# (chave da dimensão) identifica o registro que sofreu alteração, Variável identifica qual atributo do registro sofreu alteração e Data_Início e Data_Fim identificam o período em que o valor contido em Valor foi válido para o atributo contido em Variável. Quando uma alteração ocorre, o valor antigo é registrado na tabela de fatos sem fatos e o novo valor é substituído no atributo correspondente da dimensão.

Dimensão#	Variável	Valor	Data_Início	Data_Fim
112	Gerente	David	11/04/1992	10/09/1999
112	Gerente	Carlos	11/09/1999	10/10/2000
90	Estado_Civil	Solteiro	15/02/1975	12/08/1998
90	Estado_Civil	Casado	13/08/1998	12/12/1999
234	Nível_Renda	2.000	23/04/1998	14/10/1999
234	Nível_Renda	3.500	23/10/1999	10/10/2000
…	…	…	…	…

Figura 4.7 – Tabela de Fatos sem Fatos Histórica.

O principal problema que restringe o uso dessa solução é o fato de que o domínio do atributo valor não pode ser variável, pois domínios variáveis não existem. Neste caso é impossível implementar a tabela acima porque Valor contém tanto valores textuais quanto numéricos. Para usar essa solução, todos os atributos históricos das dimensões teriam que necessariamente ter o mesmo domínio. Isso também vale para as chaves das dimensões. Esta é uma restrição inaceitável no projeto de DW's.

Outro problema que surge nesta proposta é a execução de *browsing*. O *browsing* nesta tabela é diferente do tradicional. A tarefa de *browsing* é basicamente executar projeções sobre atributos de dimensões, enquanto que o *browsing* sobre esta tabela consiste apenas em executar seleções. Isto pode gerar problemas na execução de *Star Joins* [Red Brick 95] porque o *browsing* geralmente é sua atividade antecessora.

Capítulo 5

Meta-esquema Temporal em Estrela (MET*)

5.1 Introdução

No Capítulo 4 analisamos técnicas propostas por alguns autores para guardar o histórico das dimensões e pudemos comprovar que elas constituem, em grande parte, alternativas de projeto aplicadas a situações específicas, visando suprir necessidades dos usuários de seus data warehouses. Essas necessidades geralmente surgem depois que o DW já foi construído.

Kimball afirma que a solução não é pôr tudo dependente do tempo. Ele prefere não criar nenhum relacionamento direto entre as dimensões e a dimensão Tempo. Por isso suas soluções geralmente particionam o histórico através da tabela de fatos, exceto as SCD's Tipo 3 que, no entanto, só guardam um nível de histórico. Mostramos também no Capítulo 4 que o uso da tabela de fatos para rastrear o histórico não é uma boa alternativa devido principalmente a dois motivos: possibilidade de lapsos de tempo no rastreamento do histórico e desempenho ruim em análises históricas por causa do tamanho das tabelas de fatos.

O *Temporal Star Schema* de [Bliujute et al 98] mostra-se uma alternativa interessante. Entretanto, discordamos que seja eliminada a dimensão tempo do esquema em estrela original e sentimos a falta de um melhor embasamento teórico para a manipulação desse esquema. A referência apenas cita que toda a complexidade de manipulação desse esquema deve ser embutida na linguagem de consulta, mas não cita e nem discute quais as operações necessárias à manipulação de um esquema desse tipo.

As tabelas de fatos históricas de [Sanders 98] também são uma alternativa interessante, porém, elas impõem restrições no projeto de esquemas em estrela além de dificultarem tarefas básicas (operações OLAP) comumente executadas num DW, por exemplo, *browsing*.

Neste capítulo, mais precisamente na Seção 5.3, apresentaremos uma solução para guardar o histórico de dimensões em esquemas em estrela: o Meta-esquema Temporal em Estrela (**MET***). O desenvolvimento de nossa proposta baseou-se nos requisitos da Seção 4.1. Nossos principais objetivos foram:

1. Estender os esquemas em estrela tradicionais para suportar a mudança de valores de atributos de dimensões;

2. Desenvolver uma solução geral que se aplique a qualquer caso independente da natureza e o do tamanho da dimensão;

3. Guardar o histórico de forma clara, exata e de fácil compreensão para todos os usuários de DW;

4. Oferecer uma solução completa e formal do ponto de vista prático e teórico.

5. Evitar perda de desempenho e redundâncias por conta do histórico de atributos dimensionais.

O objetivo 4 é particularmente importante no contexto do projeto, da manutenção e da utilização do DW. Sob o ponto de vista do projeto, nosso objetivo foi construir um modelo teórico formal que pudesse ser usado por projetistas de DW para guardar o histórico de dimensões de qualquer natureza e tamanho. Para a manutenção, nosso objetivo foi oferecer o suporte necessário às operações de atualização do DW referentes ao histórico. Entretanto, nossa preocupação não está na tarefa de extração dos dados das fontes operacionais e sim nas tarefas de atualização das dimensões quando as alterações já estiverem sendo gravadas. E sob o ponto de vista de utilização do DW, nosso objetivo foi fornecer uma interface de consulta simples que suportasse análise histórica.

O fator tempo em sistemas de informação é uma das áreas que mais tem despertado interesse entre pesquisadores [Schiel 96]. A modelagem de aspectos temporais do mundo real é uma prática essencial na tecnologia de bancos de dados. Já afirmamos várias vezes neste documento que um DW é um banco de dados histórico voltado para análise. Entretanto, o tratamento dado a alguns aspectos temporais da modelagem de DW's ainda é tosco, prova disso é o histórico de dimensões. Sendo assim, acreditamos que as soluções para os problemas de modelagem do fator tempo em DW's estão na Teoria dos bancos de dados temporais [Tansel et al 93] [Etzion et al 98]. Alguns autores também já despertaram para este fato [Widom 95] [Widom&Yang 97] [Musicant&Dabu 97] [Bliujute et al 98] [Devlin 97].

Primeiramente construímos um modelo formal (o MET*) estendendo o esquema em estrela tradicional para suportar as alterações em atributos dimensionais. Além da construção do modelo em si, estendemos a álgebra relacional padrão com novas funções e operadores para suportar consultas analíticas históricas. A implementação desses novos operadores possibilitou a extensão do SQL92 para o MET*SQL, tornando o código SQL para consultas históricas simples e de fácil entendimento.

O restante deste capítulo está organizado da seguinte forma: na Seção 5.2 introduzimos os bancos de dados temporais discutindo os principais conceitos dessa tecnologia. Na Seção 5.3 apresentamos de maneira informal através de exemplos o Meta-esquema Temporal em Estrela (MET*). Na Seção 5.4 formalizamos o MET* e introduzimos os operadores temporais que suportam análise histórica. Na Seção 5.5 apresentamos o MET*SQL.

5.2 Bancos de Dados Temporais

Bancos de dados temporais têm sido uma intensa área de pesquisa nos últimos anos. Estima-se que pelos menos 800 trabalhos científicos já tenham sido publicados nesta área [Pissinou 94] ([McKenzie 86] [Stam&Snodgrass 88] [Soo 91] [Kline 93] [Tsotras&Kumar 96] são bibliografias sobre o assunto).

Os bancos de dados convencionais mantêm informações atualizadas. Os dados nesses sistemas são atualizados sem nenhuma restrição. Dessa forma, os valores antigos são perdidos. Esta é uma solução plausível para muitas aplicações, porém há

muitas que precisam de alguma forma manter informações presentes, passadas e/ou futuras (data warehouses, por exemplo) [Tansel et al 93]. Bancos de dados convencionais guardam o *snapshot* mais atual das informações. Um *snapshot* de um banco de dados é a visão que se tem dos dados em determinado instante do tempo.

Um banco de dados temporal é um banco de dados que guarda dados passados, presentes e futuros [Tansel et al 93]. Ele tem o fator tempo associado a seus dados como uma dimensão à parte. Nele, os dados não são fisicamente alterados nem excluídos, ou seja, um BD temporal é *append-only*. Um BD relacional é formado por um conjunto de n relações que são estruturas bi-dimensionais de tuplas e atributos. Uma relação pode ser estendida para suportar o tempo como uma dimensão à parte. Ela se torna uma estrutura tri-dimensional como a da Figura 5.1. Para cada ponto do eixo de tempo a relação possui um *snapshot* diferente, ou seja, seus atributos possuem valores diferentes. Um banco de dados relacional temporal é então um conjunto de relações temporais.

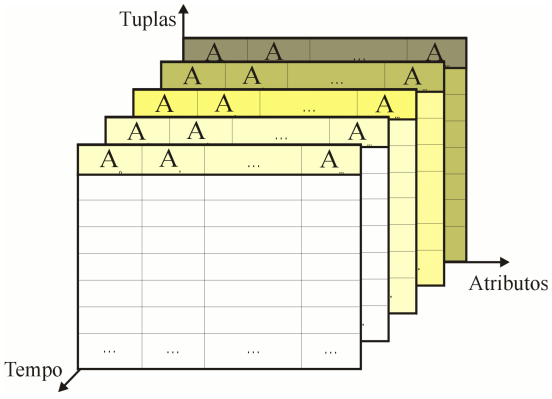


Figura 5.1 – Relação tri-dimensional temporal.

5.2.1 O eixo do tempo

O tempo da forma como o percebemos é contínuo por natureza, porém, indefinível [Schiel 96]. Santo Agostinho, em sua obra Confissões questiona: “Que é, pois, o tempo? Quem poderá explicá-lo clara e brevemente ? Quem o poderá apreender, mesmo só com o pensamento, para depois nos traduzir por palavras o seu conceito?” e afirma “Se ninguém mo perguntar, eu sei; se o quiser explicar a quem me

fizer a pergunta, já não sei". Mas se não somos capazes de defini-lo, pelo menos podemos representá-lo através de modelos.

Na modelagem dos bancos de dados temporais há duas grandes vertentes quanto à densidade do eixo temporal: tempo contínuo e tempo discreto. O primeiro é considerado isomórfico aos números reais, já o segundo é isomórfico aos números naturais. Em qualquer dos casos há uma ordenação linear [Tansel et al 93]. Há ainda modelagens que usam o tempo denso, considerado isomórfico aos números racionais [Snodgrass 95].

Quanto à sua extensão, o eixo do tempo pode ser limitado em seus dois extremos, ilimitado ou limitado em apenas um extremo. BD's temporais geralmente modelam tempo discreto e limitado nos dois extremos, considerando o primeiro extremo (o início de eixo temporal, o marco zero) alguma data importante como, por exemplo, o nascimento de Jesus Cristo, e o último extremo alguma data num futuro distante. A definição de um marco zero define o conceito de tempo absoluto, onde o tempo dos eventos é contado em relação a este marco. Por exemplo, Maria casou em 15/09/1999 do calendário Gregoriano. Outra visão considera o marco zero o instante atual, o agora, que é variável. A contagem do tempo dos eventos é feita relativamente ao tempo atual. Por exemplo, há duas semanas, há um mês, daqui a quinze dias, etc. Este é o tempo relativo [Jensen et al 94].

5.2.2 Conceitos Fundamentais

O elemento fundamental do eixo temporal é o instante que é um ponto neste eixo. Para o tempo discreto, isomórfico aos números naturais, entre dois instantes adjacentes não há outro instante. Todavia, no tempo contínuo, entre dois instantes sempre há um instante. Dois instantes quaisquer definem um intervalo no eixo do tempo. Um intervalo é o tempo entre dois instantes. Um intervalo de duração mínima, de tamanho fixo e indivisível é chamado *chronon*[19] [Jensen et al 94] [Tansel et al 93].

[19] Preferimos por não traduzir alguns termos da teoria dos bancos de dados temporais consagrados na língua inglesa.

Um *span* [Jensen et al 94] [Tansel et al 93] é uma duração do tempo de tamanho conhecido. Por exemplo, uma semana, um mês, um minuto, etc. Um *span* define uma medida de tempo, sem, no entanto, definir sua posição no eixo. Ele pode ser variável ou fixo. É fixo se sua duração independe do contexto. Por exemplo, um minuto, um dia, etc. Se sua duração depende de um contexto, ele é variável. Por exemplo, um mês, uma hora-aula, etc. O *lifespan* [Tansel et al 93] [Jensen et al 94] [Snodgrass 95] de um objeto (atributo, tupla ou relação) é o tempo em que o objeto é definido no banco de dados. O *lifespan* é definido por um ou mais *timestamps*. Normalmente um *timestamp* é um instante ou um intervalo.

5.2.2.1 Taxonomia do Tempo

Os tipos de tempo suportados por um banco de dados temporal podem ser :

- Tempo válido : o tempo em que os fatos aconteceram no mundo real. Tempo real em [Navathe&Ahmed 87] [Navathe&Ahmed 93] ;

- Tempo de transação : o tempo em que os dados são correntes no banco de dados. Os dados são correntes desde o momento em que eles foram registrados até o momento em que eles são corrigidos. O tempo de transação registra o ciclo de vida de um objeto no banco de dados [Lomet&Salzberg 93].

Um outro tipo de tempo é o tempo definido pelo usuário. Ele não é suportado pelo modelo nem pela linguagem de consulta do banco de dados temporal, faz parte de um domínio à parte, fora do eixo do temporal definido para o BD. Sua manipulação é de inteira responsabilidade do usuário. Geralmente este tempo é implementado através de atributos do tipo DATE.

5.2.3 Taxonomia dos Bancos de Dados

Os bancos de dados temporais suportam algum aspecto do tempo, sem contar, no entanto, o tempo definido pelo usuário [Jensen et al 94] [Tansel et al 93]. Um banco de dados temporal mantém a história de objetos [Tansel et al 97] baseado num modelo

temporal que define uma infraestrutura, uma linguagem para manipulação e as restrições de integridade sobre esses objetos.

De acordo com o tipo de tempo que os bancos de dados suportam, temos :

- Bancos de Dados de Tempo Válido (do inglês: *valid-time* databases): suportam tempo válido. São também conhecidos como bancos de dados históricos [Tansel et al 93].

- Bancos de Dados de Tempo de Transação (do inglês: *transaction-time* databases): são bancos de dados que suportam tempo de transação [Lomet&Salzberg 93]. Também conhecidos como bancos de dados *rollback*.

- Bancos de Dados *Snapshot*: são os bancos de dados convencionais, sem nenhum suporte temporal. O único tempo suportado é o tempo definido pelo usuário.

- Bancos de Dados Bitemporais : suportam tempo válido e tempo de transação.

Alguns autores nomeiam os bancos de dados que suportam tempo válido e tempo de transação simplesmente de bancos de dados temporais [Tansel et al 93].

5.2.4 Modelos Temporais

Vários modelos para bancos de dados temporais como extensões do modelo relacional foram propostos. Esses modelos podem ser divididos em dois grandes grupos:

Modelos Agrupados

São modelos que associam tempo aos atributos individuais. Cada valor de um atributo tem um ou mais *timestamps* associados. Esses modelos são baseados em relações N1NF (*Non-First Normal Form* : Não-Primeira Forma Normal). O *lifespan* de um atributo é representado em uma única tupla através de elementos temporais. Um elemento temporal é uma união finita de intervalos disjuntos [Gadia&Nair 93] [Jensen

et al 94]. Uma relação temporal típica de um modelo agrupado tem o aspecto da Figura 5.2. Ela não está em primeira forma normal, seus atributos são multivalorados. O *lifespan* do atributo empregado é definido em apenas uma tupla.

Empregado	Gerente
[1, 4] ∪ [10, NOW] Jonas	[1,2] Tom [2,4] Manoel [10, NOW] David
[2,3] ∪ [6,9] João	[2,3] Manoel [6,9] Luciano
[1, NOW] Sérgio	[1,2] Tom [2,4] Manoel [6,9] Luciano [10, NOW] David

Figura 5.2 – Relação temporal Empregado-Gerente num modelo agrupado.

Os modelos agrupados podem ser classificados em homogêneos e heterogêneos. Nos homogêneos todos os atributos de uma tupla são definidos em um mesmo período de tempo [Tansel 97]. Por exemplo, os atributos da primeira tupla da relação da Figura 5.2 estão definidos nos intervalos [1, 4] e [10, NOW]. Modelos heterogêneos não fazem esta restrição.

Como exemplo de modelos temporais agrupados podemos citar o HRDM [Clifford&Croker 93], o TRDM [Tansel 97] e o TempSQL [Gadia&Nair 93].

Modelos desagrupados

Modelos desagrupados associam tempo a tuplas individuais. Cada tupla tem um ou mais *timestamps* associados. As relações dos modelos desagrupados estão em 1NF (*First Normal Form* : primeira forma normal). A relação da Figura 5.2 é equivalente à relação da Figura 5.3 num modelo desagrupado.

Empregado	Gerente	*Timestamp*
Jonas	Tom	[1,2]
Jonas	Manoel	[2,4]
Jonas	David	[10, NOW]
João	Manoel	[2,3]
João	Luciano	[6,9]
Sérgio	Tom	[1,2]
Sérgio	Manoel	[2,4]
Sérgio	Luciano	[6,9]
Sérgio	David	[10, NOW]

Figura 5.3 – Relação temporal de um modelo desagrupado.

Em tais modelos o *lifespan* de um atributo pode estar dividido entre várias tuplas como podemos observar na Figura 5.3. Note que a chave de uma relação temporal não pode ser apenas um atributo invariável no tempo. A chave deve ser composta de um atributo invariável mais um dos instantes inicial ou final do intervalo.

Alguns modelos implementam os *timestamps* como propriedades implícitas das relações. Um exemplo é o TSQL2 [Snodgrass et al 94] [Snodgrass 95]. Outros os implementam explicitamente como um atributo comum da relação. Exemplo : TRM [Navathe&Ahmed 93] e o IXRM [Lorentzos 93].

5.3 Meta-esquema Temporal em Estrela (MET*)

Um esquema baseado no MET* tem o aspecto geral da Figura 5.4. Neste esquema, o histórico de um atributo é guardado numa tabela separada chamada tabela de histórico. Um tabela de histórico pode guardar valores antigos de um atributo ou de um conjunto de atributos (veja HistóricoAtributo2-n). Uma tabela de histórico típica possui as seguintes colunas: #Dimensão, contém a chave que identifica à qual linha da dimensão pertence o atributo histórico (esse é um relacionamento de chave estrangeira como outro qualquer); AtributoHistórico, que guarda os valores antigos do atributo (ou conjunto de atributos) dimensional que sofre alterações ao longo do tempo; De e Até : que indica o período de tempo (intervalo) em que o valor contido em AtributoHistórico foi válido para o atributo dimensional.

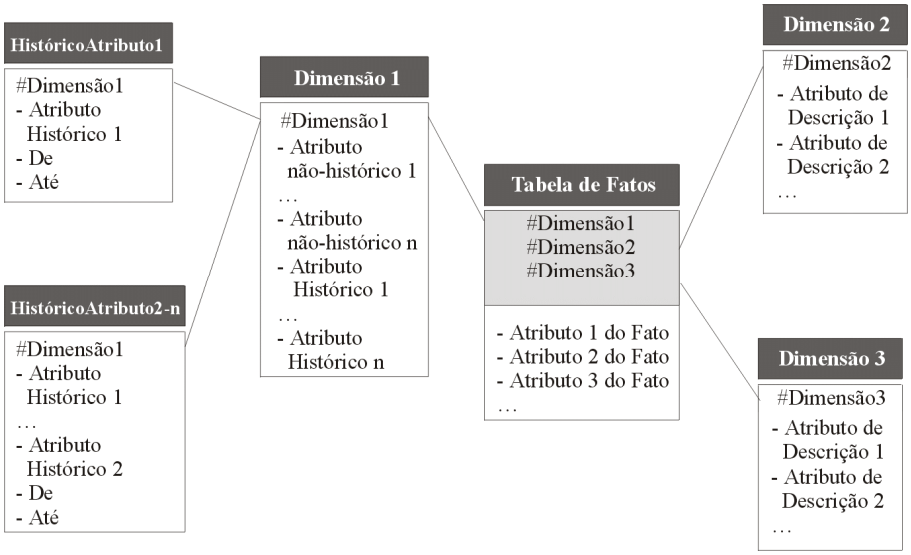


Figura 5.4 – Meta-esquema Temporal em Estrela.

Tomemos como exemplo o esquema em estrela da Figura 2.1. Para guardar os históricos do atributo Gerente da dimensão Loja e dos atributos de hierarquia de produto da dimensão Produto, estendemos tal esquema para o da Figura 5.5. A funcionalidade do MET*, tomando como exemplo a dimensão Loja, é a seguinte:

1. Para cada atributo (ou conjunto de atributos) de uma dimensão para o qual se deseja guardar o histórico (Gerente da tabela Loja, Figura 5.5) constrói-se uma tabela histórica (HistóricoGerente, Figura 5.5) com as seguintes colunas: *chave da dimensão* do(s) atributo(s) (Loja#, Figura 5.5), o(s) *atributo(s) histórico(s)* (Gerente, Figura 5.5) e os atributos *De*, o instante do tempo em que o registro se torna válido, e *Até*, o instante onde se encerra o ciclo de vida do registro;

2. Na dimensão (Loja, Figura 5.5) mantém-se o valor atual do atributo histórico (Gerente, Figura 2);

3. O valor atual também é mantido na tabela histórica. Neste caso Até contém a palavra-chave NOW para indicar que tal valor continua válido no instante atual, que é variável;

4. As chaves das tabelas de histórico são compostas pela chave da dimensão mais o atributo De. Os domínios dos atributos *De* e *Até* são o domínio da chave da dimensão Tempo (domínio de Tempo#, Figura 5.5). Note-se o relacionamento de chave estrangeira entre as tabelas históricas e a dimensão Tempo;

5. Cada vez que um atributo histórico for alterado, o atributo Até do valor antigo é atualizado para o tempo atual e o valor atual é apropriadamente incluído na tabela de histórico da dimensão correspondente; na tabela de dimensão, é mantido o novo valor atual.

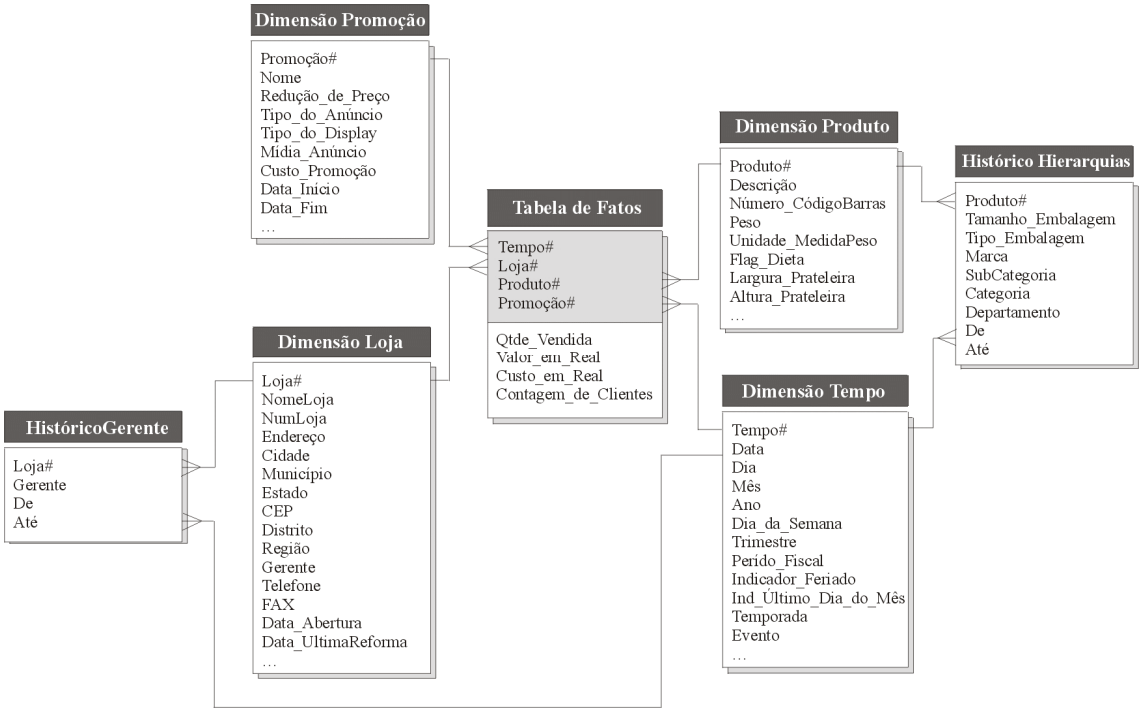


Figura 5.5 – MET* para vendas no varejo.

As vantagens do esquema MET* são:

v1 - O *Star Squema* original é preservado, portanto todas as operações OLAP não temporais continuam aplicáveis;

v2 – É possível manter o histórico integral dos atributos de interesse;

v3 - Os domínios dos atributos *De* e *Até*, sendo compatíveis com o domínio da chave da dimensão Tempo, permitem:

- fazer junção direta de tabelas históricas com a tabela de fatos (junção de HistóricoGerente e Vendas, com De θ Tempo# e/ou Até θ Tempo#). Isso é particularmente relevante porque as junções mais onerosas no esquema em estrela são aquelas feitas com as tabelas de fatos [Kimball 98], que geralmente são enormes;

- tornar iguais as granularidades de tempo das tabelas históricas e a granularidade de tempo da tabela de fatos (ou seja, a granularidade do DW continua sendo regida pela semântica da dimensão Tempo);

v4 - as tabelas históricas, como projetadas, são compatíveis com aquelas de modelos relacionais temporais formalmente definidos [Navathe&Ahmed 87] [Natvathe&Ahmed 93] [Lorentzos 93];

v5 - a transformação de um DW não temporal em um DW com um esquema MET* é bastante simples;

v6 - as versões de registros de dimensões são restritas aos atributos que mudam com o tempo, evitamos assim dados redundantes como encontrado em outras propostas.

A Figura 5.6 contém um exemplo de tabela de histórico para guardar o histórico do atributo Gerente da dimensão Loja.

5.3.1 Browsing Temporal

Esquemas MET* propiciam um novo tipo de *browsing*, *"browsing temporal"*, operado em tabelas históricas. Neste tipo de browsing o usuário executivo analisa os valores antigos de atributos históricos a fim de utilizá-los em consultas *star join*[20] [Red Brick 95] [Kimball 98].

Dimensão Loja

Loja#	Nome_Loja	Num_Loja	Endereço	...	Gerente	...
1	CGD	1	R. 13 de Maio		Jonas	
2	JPA	2	R. 21 de Abril		João	
3	NAT	3	R. 11 de Outubro		David	
…	…	…	…	...	...	...

HistóricoGerente

Loja#	Gerente	De	Até
1	Jonas	10	Now
1	João	3	9
1	David	1	2
2	João	10	Now
2	Sérgio	1	9
3	David	7	Now
3	João	2	3
3	Luis	1	1

Figura 5.6 – Tabela de histórico para o Gerente da dimensão Loja.

[20] Consultas que fazem junções entre dimensões e tabelas de fatos.

Uma operação de *browsing* típica sobre a tabela HistóricoGerente da Figura 5.6 pode ser : *select distinct Gerente, De, Até from HistóricoGerente where Loja# = 3*. O resultado desta consulta está na Figura 5.7.

Gerente	De	Até
David	7	Now
João	2	3
Luis	1	1

Figura 5.7 – Resultado do browsing : *select distinct Gerente, De, Até from HistóricoGerente where Loja# = 3*;

Note-se, porém, dois problemas nessa consulta. Primeiro, a chave da Loja não tem nenhum significado para o usuário executivo. Portanto, não devemos obrigá-lo a usá-la como critério de seleção. Segundo, o período de tempo em que cada valor de Gerente foi válido (De, Até) está expresso no conjunto resposta sob a forma de valores de chaves da dimensão Tempo, que também não têm nenhum significado para o usuário.

Para efetuar o *browsing* temporal em uma tabela de histórico torna-se então necessário fazer junção da dimensão correspondente com a dimensão Tempo. Isto certamente afetará o desempenho em casos de dimensões e tabelas históricas muito grandes. Contornamos este problema incluindo nas tabelas de histórico alguns atributos das tabelas de dimensão que lhes são pertinentes, para melhor descrever a semântica dos atributos históricos. É, no entanto, importante que as redundâncias sejam somente aquelas imprescindíveis ao bom entendimento do conteúdo de uma tabela histórica. Veja a Figura 5.8.

HistóricoGerente

Loja#	Nome_Loja	Gerente	De	Até
1	CGD	Jonas	10	Now
1	CGD	João	3	9
1	CGD	David	1	2
2	JPA	João	10	Now
2	JPA	Sérgio	1	9
3	NAT	David	7	Now
3	NAT	João	2	3
3	NAT	Luis	1	1

Figura 5.8 – Tabela HistóricoGerente adaptada ao *browsing* temporal.

Também poderíamos incluir nesta tabela mais dois atributos : DataInício e DataFim com as data correspondentes aos atributos De e Até de cada registro. Porém, como a dimensão Tempo geralmente não ultrapassa milhares de registro, optamos por utilizar a função KEYTODATE (ver Seção 5.5.1) que recupera nessa dimensão a data correspondente a uma determinada chave. Assim teríamos: *select distinct gerente, KEYTODATE(De), KEYTODATE(Até) from HistóricoGerente where Nome_Loja ="CGD"*.

Na maioria dos casos só é necessário incluir nas tabelas de histórico o atributo que caracteriza a entidade relativa àquele histórico. Por exemplo, no caso do histórico de gerente, o atributo Nome_loja, no caso do histórico de hierarquias, o atributo Descrição do produto, no caso de histórico de demografia, o Nome do cliente, etc.

É importante salientar aqui que não confrontamos Kimball que afirma que a solução não é colocar tudo dependente do tempo [Kimball 98]. Verdadeiramente nossa solução cria uma dependência apenas para os atributos que realmente são dependentes do tempo, logo, apenas as tabelas históricas são dependentes do tempo. As dimensões continuam ortogonais entre si.

5.3.2 A Semântica do Histórico

O MET* é flexível na quantidade de atributos históricos que podem ser reunidos numa só tabela. Esta, portanto, é uma decisão de projeto. Porém, é importante notar que dependendo de quais atributos sejam reunidos numa única tabela, a semântica do histórico pode ser diferente. Vejamos esta questão através de um exemplo.

Suponha que na dimensão Loja da Figura 5.5 ao invés de guardarmos apenas a data da última reforma que a Loja sofreu, decidíssemos guardar um nome para a reforma, o orçamento, o nome da empresa responsável e uma descrição sucinta. Obviamente uma reforma tem muito impacto sobre as vendas da Loja. O usuário executivo certamente terá interesse em rastrear as vendas da Loja antes e depois da reforma, bem como de reformas anteriores. Percebemos que esse é um atributo histórico.

Decidimos então guardar o histórico das Reformas. Todavia, ao invés de criarmos uma tabela de histórico apenas para Reformas, decidimos guardar seu histórico junto com o histórico de Gerente. Construímos então a tabela da Figura 5.9.

HistóricoGerenteReforma

Loja#	NomeLoja	Gerente	Reforma	Descrição	Orçamento	Empresa	De	Até
1	CGD	Jonas	Reforma1	...	...	...	10	Now
1	CGD	João	Reforma1	...	...	...	6	9
1	CGD	João	Reforma0	...	...	...	3	5
1	CGD	David	Reforma0	...	...	...	1	2
2	JPA	João	Reforma4	...	...	...	10	Now
2	JPA	Sérgio	Reforma4	...	...	...	7	9
2	JPA	Sérgio	Reforma3	...	...	...	1	6
3	NAT	David	Reforma5	...	...	...	7	Now
3	NAT	João	Reforma5	…	...	...	2	3
3	NAT	Luis	Reforma5	…	...	...	1	1

Figura 5.9 – Tabela HistóricoGerenteReforma.

Note-se nesta tabela registros com nomes de Gerentes repetidos para uma mesma Loja (registros 2 e 3 para João e 6 e 7 para Sérgio) e também nomes de Reformas repetidos para uma mesma Loja (registros 1 e 2 para Reforma1, 3 e 4 para Reforma0, 5 e 6 para Reforma4 e 8,9 e 10 para Reforma5).

A semântica do histórico que estamos guardando nesta tabela é : *um gerente que realizou uma determinada reforma em uma certa loja, durante um certo período*. Um registro desta tabela não guarda o histórico nem de Gerente e nem de Reforma. O histórico do gerente João, por exemplo, está particionado entre os registros 2 e 3 e da Reforma5 está particionado entre os registros 8, 9 e 10. Isto acontece porque os atributos Gerente e Reforma são ditos temporalmente assíncronos [Navathe&Ahmed 93].

Grosso modo, dois atributos são síncronos se seus valores são alterados sempre no mesmo instante, caso contrário eles são assíncronos. Atributos assíncronos são ditos temporalmente dependentes entre si (ver Seção 5.4.2.2).

Atributos temporalmente assíncronos representam entidades diferentes do mundo real. Nosso exemplo é claro, Gerente e Reforma são entidades que se relacionam neste modelo apenas pelo *período em que um participou do outro*. A principal funcionalidade do MET* é prover junção entre entidades lógicas (tabelas) que se relacionam através do tempo (ex. Gerente e Vendas, Reforma e Vendas, Gerente e Reforma, etc.). Neste caso, não aconselhamos unir numa só tabela entidades que não se

relacionam semanticamente, caso contrário, a semântica do histórico pode tornar-se obscura.

Entretanto, não podemos ser tão rigorosos com o fator sincronismo temporal para a construção de tabelas históricas. Se assim o formos, veremos a proliferação dessas tabelas tornar qualquer esquema MET* uma "teia de aranha" difícil de entender e de manter. Por isso definimos como critério de construção o *sincronismo temporal semântico*. Dois atributos são semanticamente síncronos se seus valores são alterados em instantes diferentes, mas eles pertencem a uma mesma entidade semântica do mundo real. Por exemplo, os atributos de hierarquias de produto são semanticamente síncronos. Seus valores mudam em instantes diferentes, mas eles pertencem à mesma entidade semântica: hierarquia de produto.

Uma dimensão representa um ramo, um departamento, um componente do negócio. Podemos assim, sem perda de generalidade, afirmar que uma dimensão modela uma entidade do negócio. Porém, podemos facilmente comprovar que uma entidade do negócio é formada pela composição de várias outras, digamos assim, sub-entidades, que preferimos chamar de *entidades semânticas*. Por exemplo, a dimensão Loja é composta de um *Corpo Gerencial*, atributos de *Geografia*, atributos de *Descrição* e *Reformas*. A dimensão Cliente é composta de *Dados Pessoais*, atributos de *Geografia* e atributos de *Demografia*. Veja a Figura 5.10. Cada uma dessas sub-entidades é geralmente representada por um conjunto de atributos. Neste caso, não nos interessa guardar o histórico de *um* atributo (a menos que ele sozinho represente uma entidade semântica) e sim de uma entidade completa.

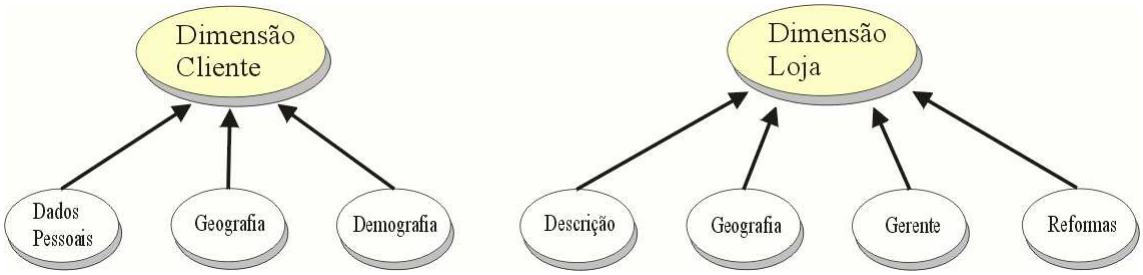


Figura 5.10 – Composição das dimensões Loja e Cliente.

A semântica do histórico não é obscura nos exemplos acima porque os atributos históricos assíncronos são reunidos em uma tabela para representar o histórico de **uma única entidade semântica**.

No MET* definimos o operador de projeção temporal (Π^t , TPROJECT) (ver Seção 5.4.3.4) que permite a recuperação do histórico de um único atributo ou de um subconjunto de atributos de uma tabela de histórico. O operador TPROJECT(HistóricoGerenteReforma, Nome_Loja, Gerente) aplicado sobre a tabela da Figura 5.9 gera a tabela da Figura 5.8. Este operador comprime em um único registro dois ou mais registros idênticos cujos intervalos se interceptam ou são adjacentes. Isso evita que o *lifespan* do valor de um atributo esteja particionado entre vários registros de uma tabela, o que ocorreria se aplicássemos uma projeção comum sobre uma tabela de histórico.

5.4 Formalização do MET*

A restrição de criar uma solução implementável sobre um SGBD relacional comercial fez-nos basearmos em modelos temporais desagrupados (1NF) com atributos de *timestamping* explícitos (ver Seção 5.2). Modelos N1NF são muito complexos para serem implementados em SGBD's relacionais. Certamente esta seria a melhor alternativa para nosso problema porque eliminaria a redundância de dados e o versionamento de chaves e evitaria a criação de novas tabelas exclusivamente para guardar histórico. Entretanto, além de não ser possível implementar este tipo de modelo num SGBD relacional, eles são menos naturais aos olhos de projetistas e usuários acostumados com o modelo relacional clássico.

Alguns modelos e linguagens não utilizam atributos explícitos para *timestamping*. Ao invés disso, eles agregam o fator tempo aos dados através de propriedades implícitas que não podem ser manipuladas diretamente como um atributo comum. Um exemplo é o TSQL2 [Snodgrass et al 94] [Snodgrass 95], uma tentativa de padronização do SQL para manipulação de dados temporais. Entretanto, trabalhar com o tempo como propriedade implícita deixa a semântica da linguagem obscura e de difícil entendimento. Por isso nosso modelo baseia-se em atributos

explícitos (De e Até) e implementa *timestamping* de tuplas ao invés de *timestamping* de atributos.

5.4.1 A Semântica do Tempo

5.4.1.1 Linha do Tempo

A linha do tempo no MET* é isomórfica ao conjunto dos números naturais. Ela é modelada como uma seqüência discreta de pontos indivisíveis, chamados instantes. Um *instante* I é atômico e tem duração desprezível. Entre dois instantes consecutivos não há duração de tempo, ou seja, não há um instante. Consideramos que o *chronon,* ou o menor "pedaço" de tempo fixo que pode ser representado no modelo, é equivalente ao instante. O *chronon* é definido pela granularidade do DW. Para o DW da Figura 5.2 o *chronon* é o dia.

O tempo no MET* é absoluto, limitado em seu início por um instante I_0 e sendo contado a partir deste instante. O tempo é limitado em seu fim por um instante I_n. Esses dois instantes são definidos pelo projetista do DW. Sendo que I_n pode ser um instante num futuro longínquo.

Formalmente a linha de tempo (L) no MET* é definida pelo conjunto finito : $L= \{I_0, I_2, I_3, ..., I_n\}$, onde $I_i \in N$ e dados I_i e I_j com e $i, j \in N$ e $i < j$, então $I_i < I_j$. Se $j= i +1$ o instante I_i (I_j) é dito predecessor (sucessor) de I_j (I_i).

A função ord : $L \rightarrow N_n = \{0,1,...,n\}$ é definida como $ord(I_i) = i$.

O pedaço de L utilizável na modelagem é $LT \subseteq L$. $LT= \{ I_0, I_2, I_3, ..., NOW\}$, onde NOW é uma palavra chave que indica o instante atual. Como o tempo não pára, NOW é sempre variável e LT é redefinida a cada instante. Definimos LT porque um DW é um banco de dados histórico que modela apenas o passado e o presente. Portanto, não guardamos no DW informações futuras.

Um intervalo é definido como o conjunto de instantes entre dois instantes. Formalmente, dados dois instantes $I_i, I_j \in LT$ um intervalo IT sobre LT é dado por $IT = \{I_k \in LT \mid I_i \leq I_k \leq I_j\}$, ou seja, o conjunto de instantes entre dois instantes. Note-se que,

por esta definição, LT também é um intervalo e qualquer outro intervalo é um subconjunto de LT. Um intervalo pode ser expresso por $[I_i,I_j]$, onde I_i e I_j são instantes pertencentes ao intervalo e são os extremos dele, $I_i \leq I_j$ e $I_i, I_j \in$ LT. Chamaremos os extremos do intervalo de IT.De e IT.Até. Escolhemos intervalos fechados nos dois extremos por uma questão de clareza para o usuário final.

O conjunto de todos os intervalos em LT é dado por : CIT = { ($[I_i, I_j]$) | $I_i, I_j \in$ LT e $I_i \leq I_j$}. Note que se i = j o intervalo é da forma $[I_i, I_i]$. Intervalos deste tipo são ditos intervalos elementares e são isomórficos aos instantes de LT.

A Figura 5.11 contém uma representação gráfica dos conceitos discutidos acima.

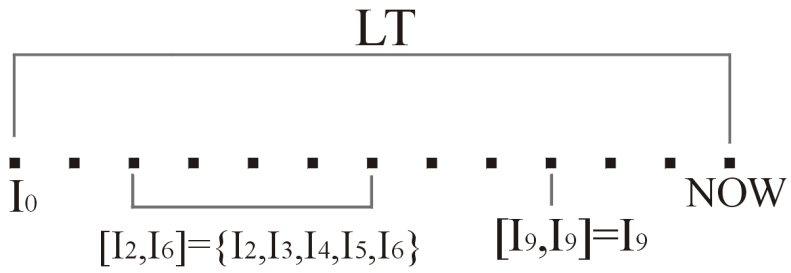


Figura 5.11 – A linha do tempo no MET*.

Instantes são ideais para registrar a ocorrência de eventos. Segundo [Jensen et al 94] um evento é um fato[21] instantâneo, algo que ocorre num instante. Enquanto que intervalos são ideais para modelar estados de entidades e seus atributos. Portanto, instantes são usados na tabela de fatos para o registro de fatos do negócio enquanto que intervalos são usados para modelar os históricos de atributos dimensionais. Uma mudança de estado é geralmente causada por um evento, cabe ao projetista a decisão de registrar ou não tal evento. Por exemplo, a mudança de um gerente é causada pela demissão (evento) do gerente antigo. Esta demissão pode ser de interesse para o usuário executivo podendo ser registrada como um evento.

[21] Neste caso, fato não corresponde estritamente ao conceito definido na teoria dos data warehouses. Fato é apenas um acontecimento qualquer.

5.4.1.2 Tempo Válido X Tempo de Transação X Tempo do DW

Além do tempo válido e do tempo de transação definidos na Seção 5.2.3, [Devlin 97] define mais um tempo : o Tempo de BDW (*Business Data Warehouse Time*), que é o tempo em que os dados são registrados no data warehouse. A definição do tipo de tempo a ser registrado no DW é um ponto crítico do projeto de esquemas MET*. Sabemos que os dados do DW são derivados dos sistemas operacionais e a maioria desses sistemas não possuem nenhum tipo de suporte temporal. Portanto, a não ser que os sistemas operacionais suportem tempo definido pelo usuário, o DW não terá como suportar este tipo de tempo.

Dependendo da técnica de *refresh* [Widom 95] utilizada para a propagação das alterações para o DW, o tempo de transação pode ser recuperado dos arquivos de log dos sistemas operacionais. Para DW's com granularidade mais grossa este tempo pode ser o ideal, pois ele é o mais próximo do tempo válido.

O tempo de BDW é o mais simples de ser registrado no DW. Como a maioria dos DW's é atualizada periodicamente, este tempo é tão mais próximo do tempo de transação quanto mais fina for a granularidade do DW. Para DW's com granularidade muito grossa (semanal, mensal) este tempo não é satisfatório para o particionamento do histórico. Alguns DW's já possuem *refresh* instantâneo, ou seja, alterações nos sistemas operacionais são imediatamente propagadas para o DW. Neste caso o BDW pode ser o tempo escolhido.

O principal objetivo do MET* é refletir o histórico dos dados de forma exata. Portanto, os eventos devem ser registrados com o tempo mais próximo possível do tempo real em que eles ocorreram. Sendo assim, podemos considerar que o MET* modela o tempo válido, que, se possível, será fornecido a partir do tempo definido pelo usuário do BD operacional.

5.4.2 Relações MET*

Do ponto de vista temporal, as relações de um esquema MET* são classificadas como:

Relações estáticas: as tabelas de dimensão do esquema relacional em estrela;

Relações intervalo-temporais: tabelas históricas;

Relações instante-temporais: tabela de fatos do esquema relacional em estrela, em que cada registro corresponde a um instante de tempo.

Seja D_1 ... D_n conjuntos finitos de valores do mesmo tipo (ex. inteiros, reais, racionais, caracteres, *strings*, etc.) , definimos uma tupla t como $t = <a_0, a_1, ..., a_n>$ onde $a_i \in D_i$. Podemos definir uma tupla intervalo-temporal t^T como $t^T = <t', IT>$ onde $t' \subseteq t$ e $IT \in CIT$. t' contém os valores dos atributos históricos de t e IT é um dos intervalos de CIT. Para $IT=[I_i, I_j]$:

- se $I_i \leq I_j$ então t^T é chamada tupla intervalo-temporal;
- se $I_i = I_j$ então t^T é chamada tupla instante-temporal.

Uma relação R estática é definida então como um conjunto de tuplas t, ou melhor, um subconjunto de $D_0 \times D_1 \times ... \times D_n$. Uma relação intervalo-temporal R^T é então, um conjunto de tuplas t^T, ou seja, um subconjunto de $D^T_0 \times D^T_1 \times ... \times D^T_k \times CIT$. Uma relação instante-temporal é um conjunto de tuplas instante-temporais.

A chave primária K de uma relação estática é um subconjunto de $\{a_0, a_1,..., a_n\}$. A chave K não sofre alterações, portanto K e t' são conjuntos disjuntos. $K^c = \{(K, I_i) \mid K \in t$ é uma chave e $I_i \in IT\}$ é o conjunto de chaves candidatas para uma relação intervalo-temporal. Escolhemos como chave primária $K^t = (K,I)$ tal que I= IT.De. Isso evita a presença na relação de duas tuplas com atributos históricos iguais no mesmo tempo.

5.4.2.1 Calendário

Segundo [Snodgrass 95] [Tansel et al 93] [Jensen et al 94] um calendário provê a interpretação humana do tempo. O MET* é totalmente flexível quanto à definição do calendário o qual é definido na dimensão Tempo. A linha do tempo do MET* é dependente da chave dessa dimensão, que geralmente é um inteiro de 4 bytes. Este artifício permite que o projetista defina o calendário do data warehouse sem nenhuma restrição imposta pelo nosso modelo, bem como permite que ele defina vários

calendários diferentes para um mesmo DW através da construção de várias dimensões Tempo.

5.4.2.2 Sincronismo Temporal

Grosso modo, dois atributos são ditos síncronos quando seus valores são alterados sempre ao mesmo tempo, caso contrário eles são assíncronos. Dois atributos assíncronos possuem dependência temporal entre eles. A dependência temporal é uma relação de equivalência entre dois atributos. Formalmente, segundo [Navathe&Ahmed 93], uma relação R^T possui dependência temporal se, dadas duas tuplas t^T_1, $t^T_2 \in R^T$, existe pelo menos um par de atributos a_i, a_j tal que :

1) $t^T_1(K) = t^T_2(K)$,

2) $t^T_1(a_i) = t^T_2(a_i)$ XOR $t^T_1(a_j) = t^T_2(a_j)$ com $i \neq j$, e

3) IT_1 ADJACENT IT_2 (ver Seção 5.4.3.1).

Por exemplo, na Figura 5.6 os atributos Gerente e Reforma são temporalmente dependentes. Nas duas primeiras linhas (tuplas t^T_1 e t^T_2) as chaves K são iguais (K = 1), o atributo reforma possui valores iguais nas duas linhas ($t^T_1(a_j) = t^T_2(a_j)$) e os intervalos dessas linhas são adjacentes ([6,9] e [10, NOW]).

Uma relação está em Forma Normal Temporal (FNT) se, e somente se, ela estiver na Forma Normal de Boyce-Codd e não possuir dependência temporal. A FNT evita que o *lifespan* do valor de um atributo esteja fragmentado entre várias tuplas, que devem ser independentes entre si. Uma tupla é a unidade de informação fundamental de uma entidade. Portanto, o histórico (a entidade) deve estar semanticamente representado em apenas uma tupla (ver Seção 5.3.2).

5.4.3 Álgebra Temporal para o MET*

As operações fundamentais de projeção (Π), união (∪), diferença (-), produto cartesiano (×) e seleção (σ) da álgebra relacional [Korth&Silberschtaz 95] se mantêm inalteradas para as relações estáticas. Porém, algumas novas operações precisam ser definidas para a manipulação de intervalos das relações intervalo-temporais.

Acreditamos que os novos operadores dão o suporte necessário à análise temporal suportando consultas históricas.

5.4.3.1 Predicados de Seleção para Intervalos

Para a álgebra temporal do MET* definimos os predicados de seleção para intervalos abaixo. Sejam [A,B] e [C,D] dois intervalos, então:

- [A,B] BEFORE [C,D] é Verdadeiro se, e somente se, $B < C$;
- [A,B] AFTER [C,D] é Verdadeiro se, e somente se, $A > D$;
- [A,B] DURING [C,D] é Verdadeiro se, e somente se, $(A \geq C)$ e $(B \leq D)$;
- [A,B] OVERLAP [C,D] é Verdadeiro se, e somente se, $(A \leq D)$ e $(C \leq B)$;
- [A,B] FOLLOWS [C,D] é Verdadeiro se, e somente se, $A - D = 1$;
- [A,B] PRECEDES [C,D] é Verdadeiro se, e somente se, $C - B = 1$;
- [A,B] EQUIVALENT [C,D] é Verdadeiro se, e somente se, $(A = C)$ e $(B = D)$;
- [A,B] ADJACENT [C,D] é Verdadeiro se, e somente se, $(C - B = 1)$ ou $(A - D = 1)$;
- [A,B] BEGINS-TOGETHER [C,D] é Verdadeiro se, e somente se, $A = C$;
- [A,B] ENDS-TOGETHER [C,D] é Verdadeiro se, e somente se $B=D$;
- [A,B] STARTS [C,D] é Verdadeiro se, e somente se $(A = C)$ e $(B < D)$;
- [A,B] ENDS [C,D] é Verdadeiro se, e somente se $(A > C)$ e $(B = D)$.

Alguns operadores são equivalentes aos operadores de [Allen 83] (BEFORE (<), EQUIVALENT (=), FOLLOWS (meets) e STARTS). Para definir OVERLAP e DURING preferimos a abordagem de [Navathe&Ahmed 93], onde esses operadores são mais abrangentes. BEGINS-TOGETHER e ENDS-TOGETHER são baseados em [Lorentzos 93]. BEFORE/AFTER e FOLLOWS/PRECEDES são equivalentes entre si. Preferimos

por definir os dois casos para uma melhor expressividade da linguagem. Porém, não há acréscimo de funcionalidade.

Na Figura 5.12 podemos visualizar graficamente quando dois intervalos satisfazem os predicados acima.

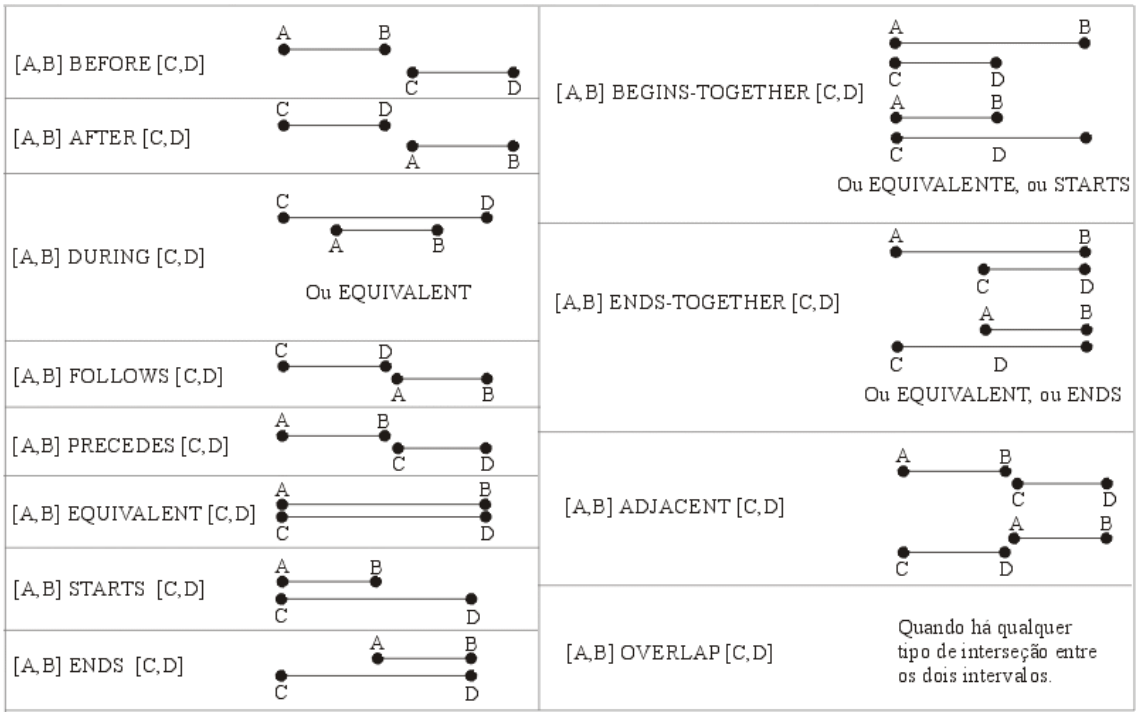


Figura 5.12 – Visualização gráfica dos predicados de seleção para intervalos.

5.4.3.2 Operadores sobre intervalos

Seja CIT^o conjunto de todos os pares de intervalos que satisfazem o predicado OVERLAP, ou seja, $CIT^o = \{(IT_1, IT_2) \mid IT_1, IT_2 \in CIT \text{ e } IT_1 \text{ OVERLAP } IT_2\}$, definimos o operador interseção de intervalos como :

$\cap^{IT} : CIT^o \times CIT^o \rightarrow CIT^o : \cap^{IT}(IT_1, IT_2) = [Max(IT_1.De, IT_2.De), Min(IT_1.Até, IT_2.Até)]$.

Também definimos os operadores:

União de dois intervalos($\cup^{IT}$):

$\cup^{IT}$: $CIT^o \times CIT^o \rightarrow CIT^o$: $\cup_{IT}(IT_1, IT_2)$= [Min(IT_1.De, IT_2.De), Max (IT_1.Até, IT_2.Até)].

Duração em instantes de um intervalo:

DUR: CIT $\rightarrow N_n = \{0,1,...,n\}$: DUR(IT) = IT.Até – IT.De +2 (isto porque tratamos com intervalos fechados nas duas extremidades).

Distância em instantes entre dois intervalos disjuntos:

$$DIS: CIT \times CIT \rightarrow N_n = \{0,1,...,n\} : DIS(IT_1, IT_2) = \begin{cases} IT_2.De - IT_1.Até, \text{ se } IT_1 \text{ BEFORE } IT_2 \\ IT_1.De - IT_2.Até, \text{ se } IT_1 \text{ AFTER } IT_2 \\ 0, \text{ caso contrário.} \end{cases}$$

5.4.3.3 Produto Cartesiano Temporal ($\times^t$)

O produto cartesiano temporal [Sarda 93] estende o produto cartesiano para combinar apenas as tuplas cujos intervalos se interceptam. Formalmente, sejam R^T_1 e R^T_2 duas relações intervalo-temporais, o produto cartesiano temporal é definido como:

$R^T_1 \times^t R^T_2 = \{(t^T_1, t^T_2) \mid t^T_1 \in R^t_1 \text{ e } t^T_2 \in R^t_2 \text{ e } t^T_1.IT \text{ OVERLAP } t^T_2.IT\}$.

5.4.3.4 Projeção Temporal (Π^t)

Uma projeção sobre uma relação intervalo-temporal pode gerar duas ou mais tuplas idênticas com intervalos que se interceptam ou que são adjacentes. Isso pode gerar respostas anômalas para determinadas consultas. Neste caso é preciso "comprimir" essas tuplas numa só unindo seus intervalos. Essa operação é realizada pelo operador *coalesce* [Sarda 93] [Böhlen et al 96] [Chomicki 95] (*fold* em [Lorentzos&Luisson 87] [Lorentzos 93] [Lorentzos& Mitsopoulos 97], *compress* em [Navathe&Ahmed 93]) que definimos a seguir:

Seja R^T uma relação intervalo-temporal, então definimos coalesce(R^T) proceduralmente da seguinte forma:

Coalesce(R^T)

início

enquanto houver tuplas t^T_1 e $t^T_2 \in R^T$

com $t^T_1.t' = t^T_2.t'$ e

(t^T_1.IT OVERLAP t^T_2.IT ou t^T_1.IT ADJACENT t^T_2.IT)

faça

$t^T_1.IT = t^T_1.IT \cup_{IT} t^T_2.IT$

retorne ($R^T - \{ t^T_2 \}$);

fim.

Podemos então definir Π^t da seguinte forma :

$$\Pi^t_{t''}(R^T) = \text{coalesce}(\Pi_{t''}(R^T)) \text{ onde } t'' \subset t'.$$

Várias álgebras e linguagens têm sido desenvolvidas para suportar dados temporais em bancos de dados relacionais. Nenhum padrão ainda foi estabelecido. TSQL2 [Snodgrass et al 94] [Snodgrass 95] é uma promessa. Porém, a sintaxe e a semântica dessa linguagem é de difícil entendimento [Davies et al 95]. Este é o principal fator de relutância à aceitação desse padrão. Talvez essa complexidade sintática e semântica seja uma conseqüência da tentativa de se criar uma linguagem temporal de propósito geral.

Propusemos um modelo simples e eficaz para nosso propósito: guardar o histórico de dimensões. Acreditamos que, por termos definido o escopo de nosso problema, conseguimos construir uma solução simples, fácil de entender, projetar e implementar. O modelo e a álgebra que propomos acima suporta todas as operações, temporais ou não, executadas sobre um esquema MET*.

Baseados na álgebra para o MET*, desenvolvemos uma extensão ao SQL92 capaz de efetuar análise histórica. Como veremos na próxima seção, nossa linguagem tem uma sintaxe e uma semântica simples. Uma consulta analítica MET*SQL é intuitiva

e fácil de construir. O modelo de consulta analítica padrão [Kimball 98] sofre pequenas extensões para suportar valores antigos de atributos dimensionais. A extensão proposta é facilmente implementável em um SGBD extensível objeto-relacional. Não é preciso o vínculo com nenhum protótipo de SGBD Temporal nem tampouco é necessário definir formalmente um novo SQL.

5.5 MET*SQL

Várias extensões ao SQL padrão foram propostas para suportar dados temporais [Toman 97] [Navathe&Ahmed 87] [Navathe&Ahmed 93] [Gadia&Nair 93] [Sarda 93] [Snodgrass et al 94] [Tansel&Tin 97].

Concordamos com [Davies et al 95] que afirma que o modelo relacional clássico não precisa de alterações para suportar aplicações temporais. Codd faz distinção entre infraestrutura, linguagens de manipulação e restrições de integridade do modelo relacional. As poucas alterações necessárias à manipulação de dados temporais devem ser feitas apenas na álgebra e nas restrições de integridade. Concordando com este fato é que não alteramos em nada a infraestrutura do modelo relacional. O modelo relacional temporal que propomos na Seção 5.4 apenas define uma semântica do tempo para ser adotada no projeto de esquemas MET*. Porém, apesar de definir uma linha de tempo rígida, o MET* é flexível quanto à definição do significado real do tempo (calendário). O próprio usuário é quem define seu calendário.

Os dois novos tipos de relação que definimos na Seção 5.4.2 (intervalo-temporais, instante-temporais) não são diferentes da definição de relação clássica do modelo relacional. Optamos por propor essas definições para facilitar o entendimento das extensões que propomos na álgebra do MET*. Nossa real extensão é a álgebra que propomos na Seção 5.4.3. Note-se também que não alteramos as operações clássicas da álgebra relacional, apenas adicionamos novas operações para suportar intervalos. O objetivo dessa nova álgebra é estender o SQL92 para suportar intervalos sem maiores modificações na estrutura da linguagem padrão. Por isso o MET*SQL não apresenta grandes alterações e nem incompatibilidades com o SQL padrão.

É importante destacar neste ponto a escolha de propormos nossas próprias extensões. Percebemos em nosso estudo que uma das causas para uma falta de

padronização para um modelo e uma linguagem temporal é que muitas aplicações necessitam de modelos próprios, ou seja, determinadas aplicações não são satisfeitas por nenhum modelo existente, nem mesmo o TSQL2. As necessidades dos usuários são muito diversas. O impasse na padronização está em se definir um modelo geral, completo e flexível que se adapte a qualquer aplicação. Um modelo que atendesse às necessidades das diversas comunidades[22] de usuários seria demasiadamente complexo [Pissinou 94], como o TSQL2 em certo grau já o é, e mesmo assim não é completo sob diversos aspectos [Böhlen et al 95].

Segundo [Davies et al 95] uma melhor alternativa é "deixar os desenvolvedores usarem o modelo relacional para definir sua própria visão do tempo em suas áreas particulares de aplicação". Usuários possuem representação própria do tempo para suas aplicações, como é o nosso caso. Sendo assim, o que interessa é o quanto pode ser casado dessa representação própria com o BD Temporal, ou melhor, com o modelo adotado pelo BD Temporal[23] [Pissinou 94]. Concordamos com este raciocínio e por isso propomos o MET*, que é facilmente implementável sobre um SGBD relacional.

Adiante apresentamos o MET*SQL através de exemplos. Os operadores formais definidos na Seção 5.4.2 foram mapeados em funções e operadores MET*SQL. Este conjunto de operadores estende o SQL 92 adicionando-lhe funcionalidades temporais. Portanto, o MET*SQL é um superconjunto do SQL92. A Tabela 5.1 contém o mapeamento dos operadores formais em funções e operadores MET*SQL. Nesta tabela também mostramos em qual cláusula da linguagem cada função ou operador pode ser utilizado.

No restante desta seção mostraremos através de exemplos a expressividade do MET*SQL bem como sua simplicidade. Os detalhes de implementação serão assunto do próximo capítulo. A implementação do MET* sobre um SGBD comercial extensível levou a pequenas alterações sintáticas no MET*SQL, porém sua expressividade foi pouco afetada.

[22] Podemos citar como exemplo comunidades que trabalham com aplicações logísticas, médicas, científicas, geográficas, de inteligência artificial, etc.

Operação Algébrica/Função	Operador MET*SQL	Tipo de dado retornado	Cláusula
Predicados de comparação de intervalos	IT_1 **PREDICADO** IT_2 , onde PREDICADO corresponde a um dos predicados da Seção 5.4.3.1.	Boolean	WHERE
$\cap^{IT}(IT_1, IT_2)$ (interseção)	INTER(IT_1, IT_2)	Intervalo	SELECT
$\cup^{IT}(IT_1, IT_2)$ (união)	UNION(IT_1, IT_2)	Intervalo	SELECT
DUR(IT_1) (duração)	DUR(IT_1)	Span (representado por um inteiro)	SELECT/ WHERE
DIST(IT_1, IT_2) (distância)	DIST(IT_1, IT_2)	Span (representado por um inteiro)	SELECT/ WHERE
$R^T_1 \times^t R^T_2$ (produto cartesiano temporal)	TCP(R^T_1, R^T_2)	Relação	FROM
$\Pi^t_{t''}(R^T)$ (projeção temporal)	TPROJECT(R^T, t'')	Relação	FROM

Tabela 5.1 - Operadores MET*SQL.

5.5.1 O Modelo de Consulta Padrão

Em [Kimball 98] encontramos um modelo de consulta padrão, conhecida como *Star Join* [Red Brick 95]. Qualquer consulta analítica sobre um esquema em estrela segue este mesmo padrão.

O MET* possibilita um outro tipo de análise: análise histórica. O modelo de consulta padrão para o MET* faz junção entre tabelas históricas e tabelas de fatos, possibilitando a análise dos fatos referentes à cada valor de um atributo histórico. Por exemplo, uma típica consulta ao MET* da Figura 5.2 é a seguinte :

C1 - "Qual o desempenho das vendas da Loja No. 1, por gerente?"

[23] Um exemplo disso é o fato de algumas aplicações necessitarem de dois ou mais tempos válidos para uma mesma relação, o que não é suportado pelo TSQL2 [Davies 95]. Isso

O comando MET*SQL para esta consulta é:

```
SELECT G.Gerente,SUM(V.Valor_em_Real),
KEYTODATE(G.De),KEYTODATE(G.Até)
FROM TCP(HistóricoGerente G, Vendas V)
WHERE G.Loja# = V.Loja#   //Junção
  AND G.NomeLoja = "Loja No. 1"
GROUP BY G.Gerente;
```

Este é o modelo de consulta histórica padrão para o MET*. Em primeiro lugar é feito um produto cartesiano temporal entre a tabela de histórico e a tabela de fatos (`TCP(HistóricoGerente G, Vendas V))`. Em seguida são feitas as junções e as seleções na cláusula WHERE.

Note-se nesta consulta a presença de uma nova função : KEYTODATE(Tempo#). Esta função simplesmente pesquisa na tabela de dimensão Tempo a data referente à chave de tempo Tempo# em seu argumento. Ela serve para apresentar o resultado ao usuário de forma mais inteligível. Isso evita que ponhamos as datas referentes à De e Até na tabela de histórico para permitir *browsing* temporal. O leitor pode argumentar que uma função deste tipo poderia evitar o acréscimo do atributo Nome_Loja na tabela da Figura 5.5. Porém, a função KEYTODATE só não afeta drasticamente o desempenho do sistema porque a dimensão Tempo é geralmente pequena. Também nos será útil a função DATETOKEY(Data) que faz exatamente o inverso da função KEYTODATE, ou seja, dada uma Data qualquer, ela pesquisa na dimensão Tempo a chave correspondente à ela.

A consulta C1 pode ser mais geral :

C2 - "Qual o desempenho das vendas de cada gerente da rede de lojas?"

```
SELECT G.Nome_Loja, G.Gerente, SUM(V.Valor_em_Real),
KEYTODATE(G.De),KEYTODATE(G.Até)
FROM TCP(HistóricoGerente G, Vendas V)
WHERE G.Loja# = V.Loja#
GROUP BY G.Nome_Loja, G.Gerente;
```

força o uso de um tempo definido pelo usuário. As extensões disponíveis na linguagem para tempo válido não são aplicáveis sobre tempos definidos pelo usuário [Snodgrass 95].

5.5.2 Outros Exemplos

Suponha que o MET* da Figura 5.2 também guarda o histórico das reformas efetuadas nas Lojas da rede. A tabela para guardar o histórico de reformas possui o seguinte esquema : HistóricoReformas(Loja#, Nome_Loja, Reforma, Descrição, Orçamento, Empresa, De, Até).

C3 – "Quais Gerentes foram responsáveis por cada Reforma nas Lojas da rede? Mostre-me os respectivos períodos."

```
SELECT R.Reforma, G.Gerente,KEYTODATE(MAXI(G.De,R.De)),
                            KEYTODATE(MINI(G.Até, R.Até))
FROM TCP(HistóricoGerente G, HistóricoReforma R)
WHERE G.Loja# = R.Loja#;
```

Nesta consulta ao invés de utilizarmos a função interseção de intervalos, utilizamos: MAXI(G.De,R.De) e MINI(G.Até, R.Até).

O resultado da consulta SQL acima é a tabela da Figura 5.6.

C4 – "Qual o desempenho das vendas da Loja No1 antes e depois da Reforma0 ?"

```
SELECT H.Nome_Loja, SUM(V.Valor_em_Real), KEYTODATE(1),
                                          KEYTODATE(H.De)
FROM HistóricoReformas H, Vendas V
WHERE H.Loja# = V.Loja#
  AND H.Nome_Loja = "Loja No 1"
  AND [1,H.De] OVERLAP [V.Tempo#, V.Tempo#]
GROUP BY H.Nome_Loja
UNION
SELECT H.Nome_Loja, SUM(V.Valor_em_Real), KEYTODATE(H.Ate),
                                          KEYTODATE(NOW)
FROM HistóricoReformas H, Vendas V
WHERE H.Loja# = V.Loja#
  AND H.Nome_Loja = "Loja No 1"
  AND [V.Tempo#, V.Tempo#] OVERLAP [H.Até, NOW];
GROUP BY H.Nome_Loja;
```

A palavra chave NOW indica o instante atual. NOW pode ser implementada como uma função que retorna a chave de tempo correspondente à data mais atual do sistema.

C5 – "Qual o desempenho das vendas da Loja No1 quando o gerente era João?"

```
SELECT H.Gerente, SUM(V.Valor_em_Real), KEYTODATE(H.De),
                                        KEYTODATE(H.Até)
FROM TCP(HistóricoReformas H, Vendas V)
```

```
WHERE H.Loja# = V.Loja#
  AND H.Nome_Loja = "Loja No 1"
  AND Gerente = "João"
GROUP BY H.Gerente;
```

C6 – "E qual o desempenho das vendas dessa mesma Loja para os Gerentes anteriores a João?"

```
SELECT G.Gerente, SUM(V.Total_Real), KEYTODATE(H.De),
                                     KEYTODATE(H.Até)
FROM TCP(HistóricoGerente G, Vendas V)
WHERE G.Loja# = V.Loja#
  AND G.NomeLoja = "X"
  AND G.Gerente IN (SELECT G3.Gerente
                    FROM HistóricoGerente G2, G3
                    WHERE G2.Gerente = "JOÃO"
                      AND G2.IT FOLLOWS G3.IT);
```

Nesta consulta IT significa o intervalo [De, Até]. Esta notação é mais agradável e intuitiva. É possível implementa-la através de tipos abstratos de dados dos modernos SGBD's objeto-relacionais (ver Capítulo 6).

C7 – "Quanto tempo durou cada Reforma na Loja No1 e quais as vendas nestes períodos?"

```
SELECT H.Nome_Loja, H.Reforma, SUM(V.Valor_em_Real),
       DUR(H.IT);
FROM TCP(HistóricoReformas H, Vendas V)
WHERE H.Loja# = V.Loja#
H.Nome_Loja = "Loja No 1"
GROUP BY H.Nome_Loja, H.Reforma;
```

C8 - "Quais as vendas dos produtos do departamento FRIOS na Loja No1 antes, durante e depois de JOÃO ser seu gerente?"

```
SELECT P.Produto, SUM(V.Valor_em_Real), KEYTODATE(1),
                                        KEYTODATE(H.De)
FROM HistóricoReformas H, Vendas V, Produtos P
WHERE H.Loja# = V.Loja#
  AND P.Produto# = V.Produto#
  AND H.Nome_Loja = "Loja No 1"
  AND P.Departamento = "FRIOS"
  AND H.Gerente = "João"
  AND [1,H.De] OVERLAP [V.Tempo#,V.Tempo#]
GROUP BY P.Produto
UNION
SELECT P.Produto, SUM(V.Valor_em_Real), KEYTODATE(H.De),
                                        KEYTODATE(H.Até)
FROM TCP(HistóricoReformas H, Vendas V), Produtos P
```

```
WHERE H.Loja# = V.Loja#
  AND P.Produto# = V.Produto#
  AND H.Nome_Loja = "Loja No 1"
  AND P.Departamento = "FRIOS"
  AND H.Gerente = "João"
GROUP BY P.Produto
UNION
SELECT P.Produto, SUM(V.Valor_em_Real), KEYTODATE(H.Até),
                                        KEYTODATE(NOW)
FROM HistóricoReformas H, Vendas V, Produtos P
WHERE H.Loja# = V.Loja#
  AND P.Produto# = V.Produto#
  AND H.Nome_Loja = "Loja No 1"
  AND P.Departamento = "FRIOS"
  AND H.Gerente = "João"
  AND [H.Até,NOW] OVERLAP [V.Tempo#, V.Tempo#]
GROUP BY P.Produto
```

C9 – "Mostre-me as vendas na época de Natal dos gerentes que trabalharam na Loja CGD."

Este é um exemplo particularmente interessante porque permite uma análise real do desempenho dos Gerentes durante um período fixo.

```
SELECT H.Gerente, SUM(V.Valor_em_Real), T.Ano
FROM TCP(HistóricoReformas H, Vendas V), Tempo T
WHERE H.Loja# = V.Loja#
  AND H.Nome_Loja = "Loja No 1"
  AND T.Tempo# = V.Tempo#
  AND T.Temporada = "Natal"
GROUP BY H.Nome_Loja, T.Ano
ORDER BY H.Nome_Loja, T.Ano
```

Para ilustrar a funcionalidade de TPROJECT, suponha que guardamos o histórico de Gerente e Reformas numa única tabela com o seguinte esquema: HISTORICOGERENTEREFORMA(Loja#, Nome_Loja, Gerente, Reforma, Descrição, Orçamento, Empresa, De, Até). Uma consulta sobre esta tabela poderia ser:

C10 – "Quais as vendas do Gerente João em cada Loja que ele traballhou?"

```
SELECT H.Gerente, H.Nome_Loja SUM(V.Valor_em_Real),
KEYTODATE(H.De), KEYTODATE(H.Ate)
FROM TCP(TPROJECT(HistóricoReformas, Nome_Loja, Gerente) H,
         Vendas V)
WHERE H.Loja# = V.Loja#
  AND H.Gerente = "João"
GROUP BY H.Gerente, H.Nome_Loja;
```

```
ORDER BY H.Nome_Loja, T.Ano
```

5.5.3 Tabelas de Histórico Esparso

C11 – "Mostre-me as vendas da Loja No.1 em todas as reformas em que ela sofreu. Mostre-me também as vendas entre uma reforma e outra."

Esta é uma consulta temporal de muito interesse para o usuário executivo. Ela mostrará os impactos causados por cada reforma ao longo do tempo. A consulta permite a análise do impacto dos transtornos causados durante o período da reforma nas vendas bem como dos benefícios que esta reforma traz após sua conclusão.

Neste ponto é importante destacar o seguinte: nós consideramos na Seção 5.3.2, por questão de simplicidade, que o atributo Reforma tem a mesma natureza do atributo Gerente. Um Gerente só é alterado porque ele foi substituído, ou seja, sai um, entra outro. Portanto, não há intervalos de tempo sem Gerentes numa Loja[24]. Não há "*gaps*" de tempo para este a atributo. Entretanto, não é verdade que uma Loja esteja sempre sofrendo Reformas. Há vários períodos de tempo em que uma Loja não sofre nenhuma. Melhor dizendo, há *gaps* (em português: lacunas) de tempo entre uma reforma e outra. Uma típica tabela de histórico de reformas real tem o aspecto da Figura 5.13. Note-se, por exemplo, que entre Reforma0 e Reforma1 na LojaNo1 há o período de 6 até 9 sem reformas.

Loja#	Reforma	De	Até.
1	Reforma1	10	Now
1	Reforma0	4	5
2	Reforma4	8	Now
2	Reforma3	2	5
3	Reforma6	7	Now
3	Reforma5	2	3

Figura 5.13 – HistóricoReforma.

Infelizmente, a consulta C11 não pode ser respondida pelos modelos e operadores MET* que propusemos até agora. Isto porque nas tabelas de histórico do

[24] Pode até haver, mas podemos generalizar o caso real para tratá-lo assim. Se houver, o mesmo método que discutiremos adiante para o atributo Reforma servirá para o atributo Gerente.

MET* (e num data warehouse em geral) são guardados registros apenas do que aconteceu no mundo real, o que não aconteceu não é registrado.

[Navathe&Ahmed 93] propõe uma técnica de ordenação temporal para resolução de problemas de *gaps* (que ele chama de *breaks*, em português: quebras). Os registros de mesma chave K são ordenados global e localmente. Registros com intervalos consecutivos são agrupados segundo sua chave K e são atribuídos números de ordem baseados na ordem de seus intervalos. Esta é a ordenação local. A ordenação global é feita associando um número de ordenação independente do grupo a que o registro pertence. Entre grupos pode haver *breaks. Breaks* são também ordenadas globalmente. Assim consultas podem ser feitas envolvendo *gaps* entre um registro e outro. Esta técnica aumenta a complexidade do modelo e não serve ao nosso caso. Nenhum outro modelo suporta ordenação.

Para chegarmos a uma solução partimos da classificação dos atributos históricos em dois tipos:

- Atributos de histórico contínuo: são aqueles em que não há *gaps* de tempo entre um valor e outro, em outras palavras, a alteração em seu valor é instantânea. Por exemplo, gerente, atributos de hierarquia de produto (entidade semântica hierarquia), atributos demográficos (entidade semântica demografia). Por exemplo, um Produto muda de uma hierarquia à outra instantaneamente.

- Atributos de histórico esparso: são atributos em que há *gaps* de tempo entre um valor e outro, ou melhor, há um período de tempo até um novo valor ser incluído desde que o intervalo do valor antigo foi fechado[25]. Por exemplo, Reforma.

Para os atributos de histórico esparso torna-se necessário guardar um tipo de estado particular: o estado em que o atributo não possui valor algum. Registramos no DW a inexistência de um valor para determinado atributo durante um certo período de tempo. Assim, a tabela da Figura 5.13 pode ser estendida para a tabela da Figura 5.14.

[25] Fechar o intervalo neste caso significa substituir o valor NOW do atributo Até pelo valor do instante em que o valor do atributo histórico deixou de ser válido.

Loja#	Reforma	De	Até.
1	Reforma1	10	Now
1	Nenhum	6	9
1	Reforma0	4	5
2	Reforma4	8	Now
2	Nenhum	6	7
2	Reforma3	2	5
3	Reforma6	7	Now
3	Nenhum	4	6
3	Reforma5	2	3

Figura 5.14 – HistóricoReforma : tabela de histórico esparso.

O leitor pode argumentar que esta técnica provoca o desperdício de espaço de armazenamento, o que é verdade. Porém, é preciso optar entre desperdiçar espaço e aumentar a complexidade do modelo. Neste caso particular o desperdício de espaço não justifica um aumento da complexidade do nosso modelo. Principalmente porque a simplicidade é uma das principais vantagens do MET*.

Destacamos que não há necessidade de alterações no modelo formal do MET*. Apenas incluímos o valor 'nenhum' no domínio dos atributos históricos. Esta técnica permite o uso do MET*SQL para a construção da consulta C11. Note-se que uma consulta deste tipo sobre uma tabela de histórico esparso pode ser construída seguindo o modelo de consulta padrão.

```
SELECT H.Reforma,SUM(V.Valor_em_Real),
       KEYTODATE(H.De),KEYTODATE(H.Até)
FROM TCP(HistóricoGerente H, Vendas V)
WHERE G.Loja# = V.Loja#   //Junção
  AND H.NomeLoja = "Loja No. 1"
GROUP BY H.Reforma
ORDER BY H.De;
```

5.5.4 Time-slice

Time-slice significa fatiar um pedaço da linha de tempo para selecionar apenas os atributos cujos *timestamps* estão dentro dessa fatia. Algumas linguagens temporais executam *time-slice* explicitamente através de algum operador ou cláusula da

linguagem [Navathe&Ahmed 93] [Sarda 93]. Nós preferimos uma abordagem implícita onde a fatia de tempo pode ser determinada em um dos argumentos do operador DURING. Exemplo:

C12 – "Quais as vendas da Loja No 1 por gerente de 1995 a 1999 ?"

```
SELECT G.Gerente,SUM(V.Valor_em_Real),
KEYTODATE(G.De),KEYTODATE(G.Até)
FROM TCP(HistóricoGerente G, Vendas V)
WHERE G.Loja# = V.Loja#   //Junção
  AND G.NomeLoja = "Loja No. 1"
  AND G.IT DURING [DATETOKEY("1-JAN-1995"), DATETOKEY("31-
DEC-1999")]
GROUP BY G.Gerente;
```

Neste consulta serão selecionados apenas os gerentes que trabalham inteiramente dentro do período [1-JAN-1995, 31-DEC-1999]. O restante dos predicados de intervalos permite maior flexibilidade para o *time-slicing*. Podemos definir, por exemplo, *time-slicing* através de OVERLAP.

Capítulo 6

Implementação do MET*

6.1 Introdução

Data Warehousing é uma aplicação crítica por natureza. Os conjuntos de dados manipulados nesses ambientes são da ordem de muitos gigabytes a alguns terabytes [Kimball 98]. Além disso, o ambiente de DWing serve a muitos usuários simultaneamente. Esses fatores exigem alto grau de escalabilidade e paralelismo dos sistemas onde será implantado o DW. O ambiente de hardware e software é um dos principais fatores de sucesso/fracasso no projeto desses sistemas [Kimball 98].

Independente da arquitetura adotada (ver Seção 2.4.2.3), a maioria dos data warehouses hoje é construída sobre SGDB's relacionais (SGBDR). Mesmo arquiteturas que implementam *Data Marts* através de MDDB's necessitam de um data warehouse corporativo [Inmon 97] que geralmente é construído sobre um SGBDR. Vários problemas surgem quando se utilizam bancos de dados relacionais puros para implantação de DW's. Os problemas decorrem das diferenças existentes entre aplicações convencionais OLTP e aplicações analíticas (ver Capítulo 2). Por isso, vários fabricantes de SGBD's relacionais estão adaptando seus produtos para suportar ambientes de Data Warehousing.

Tem crescido substancialmente nos últimos anos a quantidade de aplicações não-convencionais a serem implementadas em bancos de dados. Grosso modo, uma aplicação não-convencional trabalha com tipos de dados complexos como imagens, texto, sons, vídeos, tipos compostos, etc. Essa nova gama de aplicações não encontrou em princípio o suporte necessário nos SGBDR's puros. Como alternativa surgiram os bancos de dados orientados a objetos. Porém, esta tecnologia ainda não está madura o suficiente para substituir por completo os SGBDR's. Em verdade, alguns autores

acreditam que esta substituição nunca ocorrerá [Stonebraker 96] [Anstey 98]. Temendo perder fatias do mercado para fabricantes de SGBD's-OO, os de SGBDR logo trataram de adaptar seus produtos relacionais às novas necessidades. A solução encontrada foi a união dos dois mundos: relacional X orientação a objetos. Dessa união surgiram os modernos SGBD-OR (SGBD's objeto-relacionais), ou SGBD's extensíveis [Stonebraker 96].

É inviável a implantação de data warehouses em protótipos de SGBD's temporais [Böhlen 95]. Tais SGBD's ainda estão longe de atender as reais necessidades desses ambientes. Por outro lado, os principais SGBD's comerciais "de grande porte" que suportam aplicações de DWing não oferecem um bom ferramental para aplicações temporais. Sendo assim, uma melhor alternativa para implementação do MET* são os SGBD's extensíveis. Implementamos nosso modelo como uma camada temporal acima do SGBD Oracle8 Enterprise Manager Versão 8.0.4, da Oracle Corporation [Abbey&Corey 97], utilizando extensões objeto-relacionais.

No restante deste capítulo discutiremos os principais conceitos da tecnologia objeto-relacional do Oracle8 e sua usabilidade na implementação de um DW com o MET*. Nossa discussão abrangerá além da implementação física das tabelas do MET* e das operações definidas no capítulo anterior, alguns aspectos relevantes ao *refresh* desses esquemas.

6.2 Oracle8 Objeto-Relacional

A tecnologia objeto-relacional tem por objetivo trazer para o mundo relacional características da orientação a objetos para dar suporte ao crescente número de aplicações não-convencionais a serem implementadas sobre bancos de dados [Stonebraker 96] [Anstey 98]. Aplicações não-convencionais trabalham com dados complexos, diferentes dos tipos primitivos suportados pelos SGBD's relacionais.

As principais extensões objeto-relacionais ao padrão SQL92 são os tipos de dados e funções definidos pelo usuário [Jaedicke&Mitschang 99]. SGBD's que fazem uso desta tecnologia são conhecidos como SGBD's universais ou SGBD's extensíveis. Através de funções definidas pelo usuário é possível a construção de extensões próprias ao SQL padrão. Isto nos permitiu a implementação do MET*SQL sem maiores

problemas. Entretanto, como veremos mais adiante, a tecnologia objeto-relacional ainda não está de todo madura, o que restringe a implementação de alguns conceitos do nosso modelo.

O Oracle8 implementa os conceitos da orientação a objetos de forma diferenciada de suas definições puras. Isto é necessário para manter compatibilidade com o modelo relacional. Supomos que o leitor esteja familiarizado com os conceitos da orientação a objetos, portanto, discutiremos apenas como o Oracle8 os implementa.

6.2.1 Classes e Objetos

Classes são implementadas no Oracle8 através de Tipos Abstratos de Dados (do inglês: *Abstract Data Types (ADT)*[26]). Um tipo composto pode ser definido em termos de outro tipo composto, ou seja, um ADT pode ser aninhado [Anstey 98].

O Oracle8 permite a implementação de métodos para ADT's[27]. Métodos são funções ou procedimentos escritos em uma linguagem procedural, que pode ser PL/SQL[28] ou uma linguagem de 3ª geração como C ou Java[29]. Neste caso essas funções são implementadas como bibliotecas de vínculo dinâmico. Métodos podem ser públicos ou privados.

ADT's não são construções da linguagem PL/SQL. Eles são objetos do banco de dados assim como tabelas. São criados pelo comando CREATE TYPE. Uma vez criado ele é incluído no banco de dados como um objeto e seus métodos são armazenados da mesma forma dos procedimentos armazenados (*stored procedures*). Adiante temos um exemplo da criação do tipo IT, que corresponde aos intervalos definidos no MET*.

```
REM Listagem 1: Criação do tipo abstrato com o método OVERLAP
CREATE OR REPLACE TYPE IT AS OBJECT (
        De INTEGER,
        Ate INTEGER,
MEMBER FUNCTION Overlap (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Overlap, WNDS, WNPS) );
```

[26] ADT's também são conhecidos como UDT's (*User-defined Types*).

[27] Nem todos os SGBD's-OR implementam UDF's como métodos.

[28] A linguagem procedural do Oracle.

[29] A versão do Oracle que utilizamos não aceita funções escritas em Java. Versões mais novas, como o Oracle8i versão 8.1.5, já aceitam.

A palavra chave PRAGMA define uma diretiva para o compilador PL/SQL. Neste caso ela define as restrições WNDS (*Write No Database State*) e WNPS (*Write No Package State*) para o método Overlap.

Os parâmetros de um método podem ser tipos básicos ou ADT's. Overlap aceita como parâmetros dois objetos IT.

A implementação dos métodos de um ADT é feita numa cláusula separada: CREATE TYPE BODY. A listagem abaixo contém a implementação do método OVERLAP.

```
REM Listagem 2:Implementação do método OVERLAP
CREATE TYPE BODY IT AS
MEMBER FUNCTION Overlap (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN
  IF ((IT1.De <= IT2.Ate) AND (IT2.De <= IT1.Ate)) THEN
    RETURN('T');
  ELSE
    RETURN('F');
  END IF;
END OVERLAP;

END;
```

Objetos no Oracle8 são instanciados a partir de um ADT e podem ser persistentes ou não-persistentes. Os não persistentes são residentes apenas na memória e são usados em programas PL/SQL. Os persistentes são armazenados no banco de dados em tabelas ou atributos de tabelas. Um tabela totalmente definida a partir de um ADT é dita tabela de objetos. Abaixo temos um exemplo de criação de uma tabela de objetos.

```
REM Listagem 3: Criação de uma tabela de intervalos do tipo
REM IT
CREATE TABLE Tabela_IT OF IT;
```

Objetos podem ser armazenados em colunas de uma tabela relacional. A listagem abaixo contém o comando que cria a tabela HISTORICO_GERENTE. A coluna Período contém objetos do tipo IT. Uma tabela deste tipo não é do tipo objeto.

```
REM Listagem 4:Criação da tabela de histórico do gerente
CREATE TABLE HISTORICO_GERENTE (
Loja# INTEGER,
Nome_Loja VARCHAR(50),
```

```
Gerente VARCHAR(50),
Periodo IT,
PRIMARY KEY (LOJA#, Periodo.De));
```

O Oracle8 automaticamente define métodos construtores para inicialização de objetos. Sendo assim, para inserir valores num objeto é preciso chamar seu método construtor. A Listagem 5 mostra a inserção de uma linha na tabela criada pela Listagem 4.

```
REM Listagem 5 : Inserção de valores em HISTORICO_GERENTE
INSERT INTO HISTORICO_GERENTE
VALUES (3, 'MOT', 'GEOVANNI', IT(12, 16));
```

Um objeto persistente possui um identificador único, chamado OID. OID's são valores de 128 bytes associados aos objetos armazenados numa tabela de objetos. Neste caso, cada linha da tabela é um objeto e possui um OID. O Oracle é o responsável pela atribuição de OID's aos objetos. Quando uma tabela de objetos é criada, uma coluna oculta é criada pelo Oracle para manutenção dos identificadores dos objetos. O Oracle também cria automaticamente um índice sobre esta coluna [Feuerstein&Pribyl 97]. OID's são as chaves primárias de uma tabela de objetos, porém não podem ser diretamente manipulados por programas de usuários. Quando um objeto é excluído o valor de seu identificador não é reutilizado pelo Oracle. Objetos não-persistentes não possuem OID's, nem tampouco objetos que fazem parte de uma tabela relacional.

Objetos são referenciados através de apontadores. OID's são manipulados através dos operadores REF, DEREF e VALUE. REF é um apontador para um objeto. REF(Objeto) retorna o OID de um objeto. Value(Objeto) retorna as valores contidos no objeto. DEREF(REF(Objeto)) retorna o valor de um objeto cujo REF é conhecido. Vejamos um exemplo. Podemos criar HISTORICO_GERENTE com a coluna Período não sendo um objeto IT e sim um REF para a tabela Tabela_IT. A Listagem 6 contém o código correspondente.

```
REM Listagem 6 : O Período de HISTORICO_GERENTE é um
apontador para o tipo IT
CREATE TABLE HISTORICO_GERENTE (
Loja# INTEGER,
Nome_Loja VARCHAR(50),
Gerente VARCHAR(50),
Periodo REF IT SCOPE IS Tabela_IT,
PRIMARY KEY (LOJA#, Periodo.De));
```

Neste caso para recuperarmos o intervalo correspondente de uma linha de HISTORICO_GERENTE teríamos que fazer a junção abaixo:

```
REM Listagem 7 : Junção entre HISTORICO_GERENTE e TABELA_IT
SELECT H.GERENTE, T.DE, T.ATE
FROM HISTORICO_GERENTE H, Tabela_IT T
WHERE H.PERIODO = REF(T);
```

6.2.2 Polimorfismo

Oracle8 implementa polimorfismo. Um mesmo método de um ADT pode ter implementações diferentes para parâmetros diferentes. Vejamos por exemplo o método OVERLAP das Listagens 1 e 2. OVERLAP está definido para comparação de intervalos do tipo IT. Utilizamos OVERLAP quando fazemos junção de duas tabelas de histórico e queremos selecionar apenas as tuplas cujos intervalos se interceptam. No entanto, se fizermos junção de uma tabela de histórico com a tabela de fatos, que contém uma chave de tempo (Tempo#) ao invés de um intervalo IT, OVERLAP não poderá ser utilizado. Neste caso sobrepomos a definição de OVERLAP e codificamos uma nova implementação. Veja a listagem abaixo. A segunda implementação de OVERLAP serve a junções entre tabelas de histórico e tabelas de fatos.

```
REM Listagem 8 : Polimorfismo em OVERLAP
CREATE TYPE BODY IT AS
MEMBER FUNCTION Overlap (IT1 IN IT, IT2 IN IT) RETURN CHAR IS
BEGIN
  IF ((IT1.De <= IT2.Ate) AND (IT2.De <= IT1.Ate)) THEN
    RETURN('T');
  ELSE
    RETURN('F');
  END IF;

END OVERLAP;

MEMBER FUNCTION Overlap (IT1 IN IT, I IN INTEGER) RETURN CHAR
IS
BEGIN
  IF ((IT1.De <= I) AND (I <= IT1.Ate)) THEN
    RETURN('T');
  ELSE
    RETURN('F');
  END IF;
END OVERLAP;
END;
```

6.2.3 Encapsulamento

O Oracle8 relaxa o conceito de encapsulamento por causa da natureza declarativa do SQL [Anstey 98] [Feuerstein&Pribyl 97] [Ault 98]. Portanto, é possível acessar os dados de um objeto sem seus métodos. Para prover encapsulamento total é preciso garantir privilégios aos usuários no nível de ADT (EXECUTE) e ao mesmo tempo suspender os privilégios dos usuários no nível de tabela (SELECT, INSERT, UPDATE E DELETE) [Feuerstein&Pribyl 97].

6.2.4 Herança

A versão do Oracle8 que utilizamos não implementa herança. Consultamos a documentação [Portfolio 99] [Leverenz&Rehfield 99] da versão mais atual do Oracle8 (versão 8.1.5) e não há nenhuma referência à herança. Acreditamos que a versão 8.2 trará a implementação deste conceito.

6.2.5 Pacotes (*Packages*)

Packages em Oracle8 provêem um nível mais alto de encapsulamento. Porém, diferente de outras linguagens, *packages* PL/SQL não contém classes nem métodos (ADT's). ADT's são objetos do banco de dados, portanto não são definidos em pacotes ou programas PL/SQL. Os objetos que podem ser encapsulados em um *package* são [Feuerstein&Pribyl 97] :

- Cursores;
- Variáveis (escalares, registros, tabelas PL/SQL, etc.);
- Constantes;
- Nomes de exceções;
- Procedimentos e Funções definidas pelo usuário.

Funções e procedimentos definidos pelo usuário são implementados da mesma maneira dos métodos, exceto que os primeiros pertencem a um pacote e não a uma

classe. Como pertencem a um pacote, podem ser utilizados em quaisquer dados (ou conjuntos de dados) e não apenas nas instâncias dos objetos. Por exemplo, a função KEYTODATE definida no capítulo anterior é uma função de propósito geral, aplicável tanto a campos pertencentes a um objeto quanto a qualquer outro valor, seja ele uma variável, uma constante, um campo de um registro ou uma coluna de uma tabela. Esta função foi implementada no pacote FUNCOES. Veja a listagem abaixo:

```
REM Listagem 9 : Implementação parcial do package FUNCOES
PACKAGE FUNCOES IS

  FUNCTION KEYTODATE (tempo IN NUMBER) RETURN DATE;
  PRAGMA RESTRICT_REFERENCES (TIMETODATE, WNDS, WNPS);
  …
END FUNCOES;

PACKAGE BODY FUNCOES IS

  FUNCTION KEYTODATE( tempo in NUMBER ) RETURN DATE IS
  data DATE;

  BEGIN
    SELECT date_r INTO data
    FROM groceryi.time
    WHERE time_key = tempo;
    RETURN(data);
  END KEYTODATE;
…
END FUNCOES;
```

A forma de se referenciar um objeto pertencente a um pacote é *nome_pacote.nome_objeto*. A listagem abaixo contém uma consulta submetida à tabela da Listagem 4.

```
REM Listagem 10 : Gerentes cujos intervalos contêm o instante
REM 10.
SELECT H.GERENTE, FUNCOES.KEYTODATE(H.PERIODO.DE),
FUNCOES.KEYTODATE(H.PERIODO.ATE)
FROM HISTORICO_GERENTE H
WHERE H.PERIODO.OVERLAP(H.PERIODO, 10) = 'T';
```

6.2.6 Coleções

Coleções são conjuntos de elementos do mesmo tipo. Os tipos de uma coleção podem ser primitivos ou abstratos (ADT's). Coleções podem ser definidas como colunas de uma tabela ou tipos de um objeto. Oracle8 suporta dois tipos de coleções :

nested tables (tabelas aninhadas) e *varrays* (vetores variáveis) [Feuerstein&Pribyl 97] [Anstey 98] [Ault 98] [Portfolio 99].

Varrays

Varray é uma coleção de tamanho fixo de elementos homogêneos. *Varrays* são equivalentes a vetores em linguagens de 3ª Geração. Seus elementos são indexados de 1 até o tamanho máximo declarado. Colunas de uma tabela relacional podem ser definidas como tipo *varray*. Vejamos um exemplo. Na Listagem 11, criamos um *varray* para guardar o histórico do Gerente da dimensão Loja. Depois criamos a dimensão Loja com uma coluna deste tipo. Varrays são armazenados no mesmo espaço de armazenamento de sua "tabela-pai". Atributos de objetos podem ser deste tipo. Se o tipo básico de um varray for um objeto, ele não poderá conter um atributo que seja uma coleção.

```
REM Listagem 11 : Varray armazenado como uma coluna de uma
REM tabela.
CREATE TYPE HIST_GERENTE_t AS OBJECT (
   Nome_Gerente VARCHAR(50),
   De INTEGER,
   Ate INTEGER );

CREATE TYPE HISTORICO_GERENTE AS VARRAY(20) OF
                                          HIST_GERENTE_t;

CREATE TABLE LOJA (
   Loja# INTEGER,
   Nome_Loja VARCHAR(20),
   ...
   Gerente HISTORICO_GERENTE;
   PRIMARY KEY (Loja#) );
```

Nested Tables

Uma *nested table* é uma coleção de tamanho não fixado de elementos do mesmo tipo. Uma *nested table* inicialmente é densa, no entanto, a medida em que são feitas exclusões de seus elementos ela vai se tornando esparsa. Assim como para *varrays*, colunas de uma tabela e atributos de um objeto podem ser deste tipo. No entanto, o tipo básico de uma *nested table* não podem ser do tipo coleção. *Nested tables* declaradas como colunas de tabelas são armazenadas em tabelas conhecidas como *store tables*, que são residentes em espaços de armazenamentos diferentes das "tabelas-pai". Abaixo

construímos uma tabela similar àquela criada na Listagem 11, só que o histórico do gerente é guardado numa *nested table*.

```
REM Listagem 11 : Nested table armazenada como uma coluna da
REM dimensão Loja.
CREATE TYPE HIST_GERENTE_t AS OBJECT (
   Nome_Gerente VARCHAR(50),
   De INTEGER,
   Ate INTEGER );

CREATE TYPE HISTORICO_GERENTE AS TABLE OF HIST_GERENTE_t;

CREATE TABLE LOJA (
   Loja# INTEGER,
   Nome_Loja VARCHAR(20),
   ...
   Gerente HISTORICO_GERENTE;
   PRIMARY KEY (Loja#) )
NESTED TABLE Gerente STORE AS Gerente_Table;
```

A cláusula NESTED TABLE… constrói uma *stored table* chamada Gerente_Table para armazenar a *nested table* Gerente.

Coleções são construções excelentes para atributos multi-valorados. Ao invés de criarmos outras tabelas os armazenamos em coleções.

6.3 Limitações para Implementação do MET*

Na seção anterior analisamos as principais extensões objeto-relacionais do Oracle8. Certamente o leitor deve ter ficado tentado a implementar o MET* através de muitas dessas extensões. Na verdade, a estrutura do MET* se mantém puramente relacional com algumas poucas extensões objeto-relacionais. Isto porque a tecnologia objeto-relacional ainda possui algumas limitações que impedem a implantação de data warehouses integralmente objeto-relacionais. Dividiremos nossa discussão em dois grupos : limitações estruturais e limitações de operadores. O primeiro grupo refere-se às estruturas que podem ser utilizadas para implementação do MET*. O segundo refere-se às limitações encontradas na implementação dos operadores da Seção 5.4.3.

6.3.1 Limitações Estruturais

6.3.1.1 Tabelas Particionadas

Uma importante extensão oferecida pelo Oracle8 para construção de DW's são as tabelas particionadas [Anstey 98] [Oracle 99]. Uma tabela de tamanho muito grande pode ser particionada entre vários *tablespaces* e/ou vários discos. Cada partição é tratada como uma unidade independente. Dois métodos são usados para particionamento : particionamento por faixas de valores de colunas e particionamento por função de *hashing* [Leverenz&Rehfield 99]. Projetistas e usuários não têm controle sobre as partições, exceto na hora de defini-las. Este controle é de total responsabilidade do SGBD.

Da mesma forma que tabelas, índices podem ser particionados. Índices particionados podem ser locais (construído sobre cada partição) ou globais (construído para cada partição, mas com relação à tabela inteira).

Particionamento é uma técnica fundamental para construção de tabelas de fatos. Pelo seu tamanho quase sempre enorme, o particionamento é um excelente meio de se conseguir melhoria de performance em consultas bem como em operações de manutenção dessas tabelas. Particionamento permite que operações sejam executadas paralelamente em várias partições distintas.

Para o MET* recomendamos que as partições da tabela de fatos sejam definidas pela chave de tempo. Independente do MET*, a chave de tempo é um dos principais atributos para se definir partições de tabelas de fatos. Isto porque um DW é essencialmente um BD histórico. No nosso caso ela é o atributo fundamental já que o MET* foi desenvolvido essencialmente para suportar análise histórica. A Listagem 12 contém o comando de criação da tabela de fatos vendas do MET* da Figura 4.5 particionada por anos. O DW deste exemplo armazena dados por cinco anos.

```
REM Listagem 12:Criação da tabela de fatos vendas
REM particionada
CREATE TABLE FATOS_VENDAS (
   Tempo# INTEGER,
   Loja# INTEGER,
   Produto# INTEGER,
```

```
Promoção# INTEGER,
Qtde_vendida INTEGER,
Valor_em_real NUMBER,
Custo_em_real NUMBER,
CONTAGEM_CLIENTES INTEGER,
PRIMARY KEY (TEMPO#, LOJA#, PRODUTO#, PROMOÇÃO#))
PARTITION BY RANGE (TEMPO#)
(PARTITION ANO1 VALUES LESS THAN (365) TABLESPACE ts0,
 PARTITION ANO2 VALUES LESS THAN (730) TABLESPACE ts1,
 PARTITION ANO3 VALUES LESS THAN (1095) TABLESPACE TS2,
 PARTITION ANO4 VALUES LESS THAN (1460) TABLESPACE ts3,
 PARTITION ANO5 VALUES LESS THAN (1825) TABLESPACE ts4);
```

O Oracle8 impõe restrições para o particionamento de tabelas. Tabelas de objetos ou tabelas relacionais que contenham colunas do tipo *REF*, *object*, *nested table* ou *varray* não podem ser particionadas. Como particionamento é fundamental para construção de tabelas de fatos, não podemos construir estas tabelas com nenhuma extensão objeto-relacional. Tabelas de histórico e dimensões, porém, não precisam necessariamente ser particionadas.

6.3.1.2 Coleções

Também há restrições para o uso de coleções [Anstey 98] [Ault 98]. Supomos que o leitor até aqui tenha imaginado que *nested table* é uma boa alternativa para construção de tabelas de histórico. Nós também pensamos assim no início do plano de implementação. Porém, se uma *nested table* ou um *varray* for definido como coluna de uma tabela relacional, seus elementos só podem ser acessados a partir da "tabela-pai" [Ault 98]. Não podemos construir tabelas de histórico através de coleções porque isso torna impossível realizar junções entre tabelas de histórico e tabelas de fatos e sem essas junções não conseguimos realizar análise histórica.

6.3.1.3 OID's

O fato de não podermos usar extensões objeto-relacionais na tabela de fatos também nos impede de construirmos as tabelas de histórico como tabelas de objetos. Tabelas de fatos não podem conter OID's e nem REF's. Portanto, elas nem podem ser apontadas por uma tabela de histórico e nem podem apontar para uma. Construções com chaves estrangeiras ainda são melhores alternativas para junção entre essas tabelas.

Outra restrição que identificamos para o uso de OID's é que seus valores são números de 128 bytes. Comumente chaves de dimensões e de tabelas de histórico são números de 4 a 8 bytes. Certamente haverá desperdício de espaço de armazenamento e perda de desempenho em junções se OID's forem usados como chaves.

6.3.2 Limitações de Operadores

No atual estágio da tecnologia objeto-relacional, funções definidas pelo usuário (U*ser-defined Functions* (UDF)) podem ser usadas em cláusulas SELECT SQL da mesma forma que as funções *built-in* do sistema. Assim como essas funções, UDF's podem ser divididas em funções escalares (UDSF – *User-defined Scalar Function*) e funções de agregação (UDAF – *User-defined Aggregate Function*) [Jaedicke&Mitschang 99]. UDSF's são aplicadas em colunas de algumas linhas de uma tabela (+, -, *, etc.). Funções de agregação são aplicadas sobre grupos de linhas de colunas de uma tabela (MAX, MIN, AVG, SUM, etc.). Algumas UDSF's retornam valores verdadeiro ou falso e portanto são predicados usados na cláusula WHERE[30]. Como exemplo citamos os predicados de comparação de intervalos da Seção 5.4.3.1.

Tanto UDSF's quanto UDAF's retornam um único valor, seja ele de tipo primitivo ou complexo. Alguns SGBD's no entanto permitem que UDF's retornem tabelas inteiras recebendo como parâmetro algum valor escalar. Citamos como exemplo o DB2 Universal Server da IBM Corp.. [Jaedicke&Mitschang 99] as classifica como UDTF's (*User-defined Table Functions*). Entretanto, nenhum SGBD-OR atual permite a implementação de UDF's que recebam como parâmetro duas tabelas, executem junções entre elas, e retorne um tabela resultante. Em outras palavras, nenhum SGBD-OR permite a implementação de novos operadores relacionais [Jaedicke&Mitschang 99]. Ou ainda, UDF's não podem ser chamadas da cláusula FROM do SELECT. Portanto, é impossível implementar os operadores TCP e TPROJECT da Tabela 5.1. Os comandos da listagem abaixo não podem ser executados.

```
REM Listagem 13 : TCP e TPROJECT não podem ser implementados
REM no estágio atual da tecnologia objeto-relacional.
REM As consultas abaixo não podem ser executadas.
```

[30] O Oracle8 não permite a manipulação de valores booleanos na cláusula WHERE. Este tipo de dado só esta disponível na linguagem PL/SQL.

```
SELECT G.Gerente,SUM(V.Valor_em_Real),
       KEYTODATE(G.De),KEYTODATE(G.Até)
FROM TCP(HistóricoGerente G, Vendas V)
WHERE G.Loja# = V.Loja#   //Junção
  AND G.NomeLoja = "Loja No. 1"
GROUP BY G.Gerente;

SELECT G.Gerente,SUM(V.Valor_em_Real),
       KEYTODATE(G.De),KEYTODATE(G.Até)
FROM TPROJECT(HistóricoGerenteReforma, Gerente) G, Vendas V
WHERE G.Loja# = V.Loja#   //Junção
  AND G.NomeLoja = "Loja No. 1"
GROUP BY G.Gerente;
```

6.3.2.1 TCP (Temporal Cartesian Product)

O produto cartesiano temporal combina tuplas de duas tabelas apenas quando seus intervalos se interceptam. Na definição formal da Seção 5.4.3.3 podemos observar que TCP é definido em termos de OVERLAP. Sacrificando um pouco da expressividade do MET*SQL podemos deixar de implementar o TCP sem, no entanto, perder funcionalidade. Um produto cartesiano temporal pode se implementado apenas através do OVERLAP, como na listagem abaixo.

```
REM Listagem 14:OVERLAP permite a execução de um produto
REM cartesiano temporal sem necessidade de uso do TCP.
REM A consulta C1 do Capítulo 5 em sintaxe Oracle8 é:

SELECT H.NOME_LOJA, H.GERENTE, SUM(V.DOLLAR_SALES),
       KEYTODATE(H.PERIODO.DE), KEYTODATE(H.PERIODO.ATE)
FROM HISTORICO_GERENTE H, VENDAS V
WHERE H.LOJA# = V.TEMPO#
  AND H.PERIODO.OVERLAP(H.PERIODO, V.TEMPO#) = 'T' //TCP
GROUP BY H.NOME_LOJA, H.GERENTE, H.PERIODO.DE, H.PERIODO.ATE
ORDER BY H.NOME_LOJA, H.PERIODO.DE;
```

6.3.2.2 TPROJECT (Temporal Projection)

TPROJECT é definido em termos de outro operador relacional : COALESCE. Como não poderemos usar TPROJECT de jeito nenhum em cláusulas SQL FROM, optamos por uma estratégia diferente. TPROJECT está sendo implementado como um programa Oracle8. A tabela resultante de TPROJECT será uma visão construída a partir da tabela original. Ela poderá então ser usada em cláusulas SQL. Como opção será possível materializar a visão resultante para utilizações futuras.

Alguns autores já perceberam as limitações impostas à construção de UDF's pelos atuais SGBD's-OR. [Jaedicke&Mitschang 99] já propõe a superação dessas limitações através dos UDTO's (*User-defined Table Operators*). Acreditamos que em breve teremos flexibilidade suficiente para implementar o MET*SQL integralmente.

6.4 Implementação do MET* e do MET*SQL

Apesar das limitações impostas pelos atuais SGBD's-OR, particularmente o Oracle8, a implementação do MET* é uma tarefa relativamente simples. As principais extensões do MET* são referentes à manipulação de intervalos. De resto, ele pode ser tratado como um Esquema em Estrela comum.

Encapsulamos toda as extensões referentes à manipulação de intervalos num ADT chamado IT. IT contém dois atributos De e Até e os métodos correspondentes às operações que manipulam intervalos. Todos os métodos são polimorfos. Isto porque eles precisam manipular intervalos do tipo IT, instantes (que são intervalos elementares) e intervalos soltos[31]. As demais funções não referentes a intervalos foram encapsuladas num *package* chamado FUNCOES. A Apêndice A contém os scripts PL/SQL para implementação de IT e de FUNCOES.

```
REM Listagem 17:
REM Quais as vendas da Loja 2 por Gerente e Reforma?

SELECT H.NOME_LOJA, H.GERENTE, R.REFORMA,
       SUM(V.DOLLAR_SALES),
       KEYTODATE(MAXI(H.PERIODO.DE,R.PERIODO.DE)),
       KEYTODATE(MINI(H.PERIODO.ATE,R.PERIODO.ATE))
FROM HISTORICO_GERENTE H,HISTORICO_REFORMA R, SALES_FACT V
WHERE H.LOJA# = V.LOJA#
  AND H.LOJA# = R.LOJA#
  AND H.LOJA# = V.LOJA#
  AND H.LOJA# = 2
  AND H.PERIODO.OVERLAP(H.PERIODO, R.PERIODO) = 'T'
  AND R.PERIODO.OVERLAP(R.PERIODO, V.TEMPO) = 'T'
  AND H.PERIODO.OVERLAP(H.PERIODO, V.TEMPO) = 'T'
GROUP BY H.NOME_LOJA, H.GERENTE, R.REFORMA, H.PERIODO.DE,
         H.PERIODO.ATE, R.PERIODO.DE, R.PERIODO.ATE
ORDER BY H.NOME_LOJA, H.PERIODO.DE;
```

[31] Consideramos intervalos soltos intervalos da forma [data, data] onde as data são fornecidas pelo usuário. Geralmente são usados para *time-slicing*.

Na Listagem 17 ilustramos o uso de MAXI e MINI. O operador interseção de intervalos (INTER) não pode ser usado na cláusula SELECT para calcular a interseção dos intervalos de duas tabelas históricas e exibi-la como uma coluna no conjunto resposta. Usamos então MAXI e MINI para substituí-lo.

6.4.1 *Refresh* do MET*

O *refresh* do MET* tem pequenas extensões em relação ao Esquema em Estrela padrão. O principal cuidado que devemos ter é quanto à integridade das chaves primárias das tabelas de histórico.

Assumimos que o tempo dos atributos históricos é fornecido pelos programas de extração. Em relação ao escopo deste trabalho não interessa como este tempo é obtido. O data warehouse trata esse tempo como sendo o tempo válido dos atributos. Numa outra abordagem, se houver dificuldades na extração do tempo dos sistemas operacionais e se o DW tiver granularidade fina, podemos considerar o tempo de BDW como tempo válido. De qualquer forma, assumimos apenas que o tempo é fornecido.

Ao atualizar dimensões é preciso verificar se o atributo sendo atualizado é histórico. Se for, as respectivas atualizações na dimensão e na tabela de histórico são efetuadas. Ao atualizar uma tabela de histórico torna-se necessário garantir a integridade temporal da chave primária. Para um valor de chave de dimensão não é possível incluir um valor de atributo histórico cujo intervalo intercepta outro(s) intervalo(s) presente(s) na tabela. É preciso garantir que os intervalos de uma mesma chave de dimensão não se interceptem.

Alterações em dimensões que possuam atributos históricos não devem ser feitas diretamente pelo comando UPDATE. Esta tarefa deve ser executada por uma função que efetue, além das alterações, as verificações recomendadas no parágrafo anterior. Esta função deve retornar um código de sucesso ou fracasso nas alterações.

É possível estender um Esquema em Estrela já implementado num DW para um MET* que suporta análise histórica de atributos dimensionais sem grandes modificações no ambiente de DWing. Certamente o Esquema em Estrela original ficará inalterado e todas as operações efetuadas sobre ele permanecerão válidas.

Modificações maiores, no entanto, deverão ser efetuadas nas ferramentas de *front-end* para que suportem análise histórica segundo o modelo do MET*. Isto porque incorporamos ao DW um tipo de tabela diferente das dimensões e das tabelas de fatos, e também operações diferentes do *browsing* e do *star join*. Todavia, tais modificações fogem ao escopo deste trabalho, sendo, porém, um interessante tema para trabalhos futuros.

Capítulo 7

Conclusões e Trabalhos Futuros

7.1 Conclusões

Com o advento das ferramentas OLAP e da tecnologia de Data Warehousing o usuário executivo passou a contar com um ferramental poderoso capaz de lhe fornecer os subsídios necessários para análise gerencial no nível corporativo. Informações no nível corporativo diferem sobremaneira das informações dos níveis operacional e tático. Análise estratégica exige informações sumariadas, históricas e agregadas sob diversas perspectivas do negócio. A visão da agregação sob diversas perspectivas exigiu que novos modelos de dados fossem desenvolvidos para permitir uma melhor aproximação entre os modelos lógicos dos projetistas de sistemas de informação e os modelos mentais do negócio do usuário executivo. Surgiu então a Modelagem Multidimensional (MM) que permite que os dados sejam estruturados sob uma perspectiva mais próxima da visão do usuário executivo.

A MM é o núcleo do OLAP. O processamento OLAP permite ao usuário executivo visualizar os dados sob diferentes ângulos gerenciais e provê todas as operações necessárias para a atividade de análise. Tais ângulos gerenciais são chamados dimensões do negócio. Atividades desenvolvidas em conjunto pelas várias dimensões do negócio podem ser quantificadas através de medidas de desempenho (fatos).

A estrutura ideal para representação dos conceitos de dimensões e de medidas quantificáveis é cubo multidimensional. Entretanto, soluções baseadas em Cubos Multidimensionais são proprietárias, além de possuírem outras limitações (ver Seção 1.3.3.3). A solução da indústria relacional para simular o cubo multidimensional é o esquema em estrela. Esta interessante estrutura permite o processamento analítico

(OLAP) sobre grandes conjuntos de dados em plataformas puramente relacionais. Todas as operações OLAP executadas sobre Cubos também podem ser executadas sobre esquemas em estrela.

Durante alguns anos, indústria e academia seguiram caminhos distintos no que se refere à integração de bases de dados heterogêneas para efeito de análise gerencial [Widom 95]. A alternativa comercial (*Data Warehousing*), no entanto, mostrou resultados mais concretos e mais rapidamente. Tal fato levou a uma expressiva migração da academia para pesquisas nessa área. Por ser uma solução desenvolvida para suprir necessidades comerciais, *Data Warehousing* ainda carece de soluções embasadas cientificamente. Muito já se tem feito neste sentido, porém muitos pontos ainda estão abertos a pesquisas.

Um ponto importante que identificamos foi justamente a manipulação do histórico de dimensões dos esquemas em estrela. Não há controvérsias sobre o fato de que data warehouses são bancos de dados históricos por natureza. Entretanto, o tratamento de alguns aspectos temporais das estruturas multidimensionais ainda é tosco. Em princípio imaginava-se que apenas os fatos têm um tempo associado. Entretanto, dimensões não são ortogonais em relação ao tempo, pelo menos alguns de seus atributos podem ser alterados ao longo do tempo, e essas mudanças podem interessar ao usuário executivo. Por isso, é preciso de alguma forma guardar seus valores antigos.

Aparentemente apenas Kimball [Kimball 98] havia tratado deste assunto até o momento em que foi iniciado este trabalho. Analisando suas propostas, percebemos que elas apresentam vários problemas e não atingem o objetivo real que é guardar o histórico correto e preciso dos atributos de dimensões. Aos poucos pudemos comprovar ainda mais a relevância do tema à medida que outros autores foram propondo suas próprias soluções. Destacamos a importância da lista de discussões por e-mail dwlist@datawarehousing.com. Através dela pudemos avaliar o que estava sendo feito sobre o assunto em várias partes do mundo. Todavia, as soluções propostas, em sua totalidade, são alternativas de projeto que foram implementadas para resolver casos específicos em que o usuário executivo necessitava de informações históricas.

A única referência acadêmica que encontramos sobre o tema foi [Bliujute et al 98]. Porém, mesmo esta referência não contém uma solução no nível em que gostaríamos de desenvolver. Em verdade ela contém conceitos interessantes, porém, como pudemos comprovar no Capítulo 4, ela falha em alguns pontos importantes do projeto de esquemas em estrela com suporte ao histórico de atributos.

Certamente a solução para guardar qualquer informação histórica passa pela teoria dos bancos de dados temporais. E foi utilizando os conceitos dessa teoria que propusemos uma extensão aos esquemas em estrela tradicionais: o MET* (Meta-esquema Temporal em Estrela). Impomos ao nosso trabalho alguns requisitos que deveriam ser respeitados. Nossa solução deveria ser simples, geral e formal. Simples porque não deveria afetar a estrutura original do esquema em estrela, completa porque deveria guardar o histórico de forma correta e exata, geral porque deveria ser utilizável em qualquer caso, com qualquer dimensão, e formal porque pretendíamos assegurar sua correção. Além disso, deveríamos manter intactas as operações executáveis sobre um esquema em estrela tradicional.

Nossa solução atende a todos os requisitos discutidos acima. O histórico de atributos dimensionais é rastreado de forma exata nas tabelas de histórico. A construção do MET* proporciona análise histórica que, como comprovado através de exemplos, é de grande utilidade para o usuário executivo. Nossa solução não afeta a estrutura de esquemas já existentes, pois ela é apenas uma extensão dos esquemas tradicionais. Esquemas temporais em estrela podem ser usados para guardar o histórico de qualquer atributo, ou conjunto de atributos, independente das características de tamanho e natureza da dimensão. Optamos por formalizar nosso modelo para evitar qualquer ambigüididade no que concerne aos conceitos e técnicas envolvidos na sua implementação. Partindo da formalização, construímos e implementamos uma linguagem (MET*SQL) que oferece todos as funcionalidades necessárias à manipulação de um esquema baseado no MET*. Nossa solução abrange desde o modelo estrutural em si até as particularidades de manutenção de esquemas dessa natureza. Portanto, qualquer projetista que porventura necessite guardar histórico de dimensões, encontra neste trabalho o ferramental necessário para fazê-lo.

Mostramos ainda como podemos implementar um esquema MET* e um novo SQL de forma fácil e concisa sobre um SGBD-OR. Aproveitamos para discutir algumas

limitações atuais da tecnologia objeto-relacional e mais especificamente do Oracle8 para implementação de data warehouses. Acreditamos que em breve essas limitações serão superadas, o que dará oportunidade de implementar data warehouses sem as limitações do modelo relacional puro.

Realizamos testes no Oracle8 submetendo consultas MET*SQL a um pequeno conjunto de dados. Todos os operadores definidos para a linguagem (exceto o TPROJECT, que conforme discutido na Seção 6.3.2.2 está sendo implementado) foram implementados rigorosamente seguindo suas especificações e suportaram corretamente as várias consultas submetidas. Em todas as situações, o MET* mostrou-se mais completo, mais simples e mais fácil de manipular. Nosso modelo é bem mais abrangente que as outras soluções. Em várias situações, o MET* é capaz de suportar consultas históricas que não são (em parte ou totalmente) suportadas pelas outras soluções.

7.2 Trabalhos Futuros

Apesar dos testes comprovarem que o MET* é mais abrangente e completo que as outras soluções, eles não permitiram analisar exaustivamente um esquema baseado no MET* numa situação real de DWing. Neste caso, o objetivo é analisar se o MET* suporta corretamente e satisfatoriamente consultas analíticas históricas submetidas pelos usuários finais. O primeiro trabalho futuro que pretendemos encaminhar é a construção de um ambiente de DWing próximo de uma situação real com um esquema baseado no MET*. Esse ambiente permitirá:

- Analisar mais profundamente o comportamento do MET* diante de situações reais de consultas submetidas por usuários finais;
- Realizar testes de performance (através de *benchmarks*) comparando o MET* com as outras soluções.;
- Comparar o quesito espaço de armazenamento entre soluções baseadas no MET* e outras propostas.

Pretendemos também avaliar a completude do MET*SQL em relação a linguagens temporais de consulta como TSQL, TSQL2 e HSQL, buscando adaptar tal exame à esfera de Data Warehousing.

Através do estudo de vários modelos temporais, particularmente modelos cujo elemento temporal é o intervalo [Amagasa et al 98] [Amagasa et al 98b] [Böhlen et al 98] [Clifford&Croker 93] [Etzion et al 98] [Dutta 89] [Gadia&Nair 93] [Gadia&Yeung 88] [Jensen&Snodgrass 93] [Lorentzos 93] [Lorentzos& Mitsopoulos 97] [Navathe&Ahmed 93] [Sarda 93] [Snodgrass 95] [Tansel 97] , pudemos perceber que os conceitos da teoria dos bancos de dados temporais podem ser utilizados em outros aspectos de modelagem dos data warehouses.

Especificamente, chaves de tempo de tabelas de fatos que guardam estados [Bliujute et al 98] ao invés de ocorrência de eventos, podem ser intervalos ao invés de instantes. Por exemplo, tabelas de fatos de saldos de contas bancárias guardam o estado de uma conta num determinado instante, ou seja, guardam um instantâneo do saldo. Uma melhor alternativa é guardar o saldo com o respectivo período em que ele foi válido, assim como fazemos com atributos históricos de dimensões. Este exemplo é perfeitamente suportado pelo MET*. Portanto, nosso esquema não apenas guarda histórico de dimensões. Acreditamos que um estudo aprofundado nesta direção trará bons resultados.

Analisando o IXRM *(Interval-extended Relational Model)* de [Lorentzos& Mitsopoulos 97] [Lorentzos&Johnson 87] pudemos perceber que este modelo pode ser de grande valia para a computação e manutenção de agregados. Para isso precisaríamos estender o MET* com conceitos de intervalos generalizados[32]. As grandes contribuições neste caso seriam a enorme economia de espaço proporcionada e uma substancial melhora de performance em consultas. Certamente aprofundaremos nossos estudos neste sentido.

Há no momento, a carência de uma ferramenta CASE para o projeto e manutenção de esquemas temporais em estrela. Este é mais um trabalho que

[32] Qualquer conjunto de um tipo de dado primitivo (inteiros, caracteres, etc.) pode ser ordenado (inclusive *strings*) e, portanto, é possível definir intervalos segundo essa ordem. Neste caso, generalizamos o conceito de intervalos.

pretendemos encaminhar no futuro. No entanto, possivelmente este será o último, já que ainda há estudos promissores que podem estender o MET* para agregar mais funcionalidades.

Bibliografia

[Abbey&Corey 97] ABBEY, Michael, COREY, Michael J., *"Oracle8 A Beginner's Guide"*, Osborne McGraw-Hill, 1997.

[Alalouf 97] ALALOUF, Carole, "*Hybrid OLAP"*, White Paper, Speedware Corporation Inc., Novembro, 1997.

[Allen 83] ALLEN, James F., "Maintaining Knowledge About Temporal Intervals", *Communications of the ACM*, Vol. 26, Nº 11, Novembro 1983.

[Amagasa et al 98] AMAGASA, Toshiyuke, ARITSUGI, Masayoshi, KANAMORI, Yoshinari, MASUNAGA, Yoshifumi, "Interval-Based Modeling for Temporal Representation and Operation", *IEICE Trans. on Information Systems*, Vol. E8t-D, Nº 1, Janeiro 1998.

[Amagasa et al 98b] AMAGASA, Toshiyuki, ARITSUGI, Masayoshi, KANAMORI, Yoshinari, "Implementing Time-interval Class for Managing Temporal Data", *Proc. 9th International Workshop on Database and Expert Systems Applications (DEXA)*, Agosto 1998.

[Anstey 98] ANSTEY, David A., *"High Performance Oracle8 Object-Oriented Design"*, Coriolis Groups, Inc., 1998.

[Ault 98] AULT, Michael R., *"Oracle8 Black Book"*, Coriolis Group Books, 1998.

[Bliujute et al 98] BLIUJUTE, Rasa, SALTENIS, Simonas, SLIVINSKAS, Giedrius, JENSEN, Christian S., "Systematic Change Management in Dimensional Data Warehousing", *Proc. Third International Baltic Workshop on DB and IS,* Abril 1998.

[Böhlen 95] BÖHLEN, Michael H., "Temporal Database System Implementations", *SIGMOD Record*, Vol. 24, Nº 4, Dezembro 1995.

[Böhlen et al 98] BÖHLEN, Michael H., BUSATTO, R., JENSEN, Christian S. , "Point vs Interval-based Temporal Data Models", *Proc. 14th International Conference on Data Engineering*, 1998.

[Böhlen et al 95] BÖHLEN, Michael H., JENSEN, Christian S., SNODGRASS Richard T., "Evaluating the Completeness of TSQL2", *Recent Advances in Temporal Databases*, Workshops in Computing, J. Clifford, A. Tuzhilin eds, 1995.

[Böhlen et al 96] BÖHLEN, Michael H., SNODGRASS, Richard T., SOO, Michael D., "Coalescing in Temporal Databases", *Proc. of the 22nd VLDB Conference*, 1996.

[Chaudhuri&Dayal 97] CHAUDHURI, S., DAYAL, U., "An Overview of Data Warehousing and OLAP Technology", *SIGMOD Record*, Vol. 26, No 1, Setembro 1997.

[Chen&Zaniolo 98] CHEN, Xinmin Cindy, ZANIOLO, Carlo, "Universal Temporal Extensions for Database Languages", *Proc. 15th International Conference on Data Engineering*, 1998.

[Chomicki 95] CHOMICKI, Jan, *"Temporal Query Languages : A Survey"*, Technical Report, Department of Computer and Information Sciences, Kansas State University, Janeiro 1995.

[Chuck et al 98] CHUCK, Ballard, DIRK, Herreman, DON, Schav, RHONDA, Bell, EUNSAENG, Kim, ANN, Valencil, *"Data Modeling Techniques for Data Warehousing"*, Red Book, IBM International Technical Support Organization, http://www.redbooks.ibm.com, Fevereiro 1998.

[Clifford&Croker 93] CLIFFORD, James, CROKER, Albert, "The Historical Relational Data Model (HRDM) Revisited", em [Tansel et al 93], 1993.

[Codd 93] CODD, E. F., *"Providing OLAP (On-line Analytical Processing) to user-analysts : An IT Mandate"*, E.F. Codd&Associates, 1993.

[Davies et al 95] DAVIES, Christina, LAZELL, Brian, HUGHES, Martín, COOPER, Leslie, "Time is Just Another Attribute – or at Least, just Another Dimension", *Recent Advances in Temporal Databases*, Workshops in Computing, J. Clifford, A. Tuzhilin eds, 1995.

[Devlin 97] DEVLIN, Berry, "Managing Time in the Data Warehouse", *InfoDB*, Vol. 11, Nº 1, Junho 1997.

[Dutta 89] DUTTA, Soumitra, "Generalized Events in Temporal Databases", *Proc. Int'l Conference on Data Engineering*, 1989.

[Etzion et al 98] ETZION, O., JAJODIA, S., SRIPADA, S, *"Temporal Databases: Research and Practice"*, Spring Verlag, 1998.

[Feuerstein&Pribyl 97] FEUERSTEIN, Steven, PRIBYL, Bill, *"Oracle PL/SQL Programming"*, Second Edition, O'Reilly, 1997.

[Firestone 97] FIRESTONE, Joseph M, *"Evaluating OLAP Alternatives"*, White Paper, http://www.dkms.com/OLAPALT.html, Março 1997.

[Firestone 98] FIRESTONE, Joseph M., *"Architectural Evolution in Data Warehousing and Distributed Knowledge Management Architecture"*, White Paper, http://www.dkms.com/ARCHEV.html , Julho 1998.

[Gadia&Nair 93] GADIA, Shashi K., NAIR, Sunil S., "Temporal Databases: A Prelude to Parametric Data", em [Tansel et al 93], 1993.

[Gadia&Yeung 88] GADIA, Shashi K., YEUNG, Chuen-Sing, "A Generalized Model for a Relational Temporal Database", *Proc. Conference on Management of Data*, Junho 1988.

[Gardner 98] GARDNER, Stephen R., "Building the Data Warehouse", *Communications of the ACM*, Vol 41, Nº 9, Setembro 1998.

[Golfarelli et al 98] GOLFARELLI, Matteo, MAIO, Dario, RIZZI, Stefano, "Conceptual Design of Data Warehouses from E/R Schemas", *Proc. Hawaii Int'l Conference on Sytem Sciences*, Janeiro 1998.

[Golfarelli&Rizzi 98] GOLFARELLI, Malteo, RIZZI, Stefano, "A Methodological Framework for Data Warehouse Design", *Proc. DOLAP'98*, 1998.

[Gupta 97] GUPTA, Vivek R., *"An Introduction to Data Warehousing"*, White Paper, System Services Corp., http://www.system-services.com, Agosto 1997.

[Inmon 97] INMON, W.H., *"Como Construir o Data Warehouse"*, Tradução da Segunda Edição, Editora Campus, 2ª Edição, 1997.

[Jaedicke&Mitschang 99] JAEDICKE, Michael, Mitschang, Bernhard, "User-defined Table Operators: Enhancing Extensibility for ORDBMS", *Proc. 25th VLDB Conference*, 1999.

[Jensen et al 94] JENSEN, Christian S., CLIFFORD, James, ELMASRI, Ramez, GADIA, Shashi K., HAYES, Pat, JAJODIA, Sushil, eds., "A Consensus Glossary of Temporal Database Concepts", *SIGMOD Record*, Vol. 23, Nº 1, Março 1994.

[Jensen&Snodgrass 93] JENSEN, Christian S., SNODGRASS, Richard T., "Unification of Temporal Data Model", *Proc. IEEE Int'l Conference on Data Engineering*, 1993.

[Jhamb 98] JHAMB, Dushyant, *"Slowly Changing Dimensions"*, Electronic Mail from jhamb@mailexcite.com, 23 Março 1998.

[Kanfonas 98] KAMFONAS, Michael, *"Maintaining dimensional history"*, Electronic Mail from michael.kamfonas@lmco.com, 17 Novembro 1998.

[Kelly 95] KELLY, Floyd, "Implementing an EIS", White Paper, http://www.ceoreview.com/papers/eis.htm, Janeiro 1995.

[Kenan 95] KENAN SYSTEMS CORPORATION, *"An Introduction to Multidimensional Database Technology"*, White Paper, 1995.

[Kimball 97] KIMBALL, Ralph, "A Dimensional Modeling Manifesto", *DBMS Magazine*, Agosto 1997.

[Kimball 98] KIMBALL, Ralph, *"Data Warehouse Toolkit"*, Makron Books, 1998.

[Kimball 98a] KIMBALL, Ralph, "Surrogates Keys", *DBMS Magazine*, Maio 1998.

[Kimball 98b] KIMBALL, Ralph, "Pipelining your Surrogates", *DBMS Magazine*, Junho 1998.

[Kimball et al 98] KIMBALL, Ralph, REEVES, Laura, MARGY, Ross, THORNTHWAITE, Warren, *"The Data Warehouse Lifecycle Toolkit : Experts Methods for Designing, Developing and Deploying Data Warehouses"*, John Wiley & Sons Inc., 1998.

[Klein et al 99] KLEIN, Lawrence Zordam, CAMPOS, Maria Luiza Machado, TANAKA, Astério Kiyoshi, "A Tecnologia Objeto-Relacional em Ambientes de Data Warehouse : Uso de Séries Temporais como Tipo de Dados Não Convencional", *Anais do XIV Simpósio Brasileiro de Bancos de Dados*, 1999.

[Kline 93] KLINE, Nick, "An Update of the Temporal Database Bibliography", *ACM SIGMOD Record*, Vol. 22, Nº 4, Dezembro 1993.

[Korth&Silberschatz 95] KORTH, Henry F., SILBERSCHATZ, Abraham, *"Sistema de Bancos de Dados"*, 2ª Edição Revisada, Makron Books, 1995.

[Leverenz&Rehfield 99] LEVERENZ, Lefty, REHFIELD, Diana, *"Oracle 8i : Concepts"*, Release 8.1.5, Part No. A67781-01, Oracle Corporation, Fevereiro 1999.

[Lomet&Salzberg 93] LOMET, David, SALZBERG, Betty, "Transaction-Time Databases", em [Tansel et al 93], 1993.

[Lorentzos 93] LORENTZOS, Nikos A., "The Interval-extended Relational Model and Its Application to Valid-time Databases", em [Tansel et al 93], 1993.

[Lorentzos& Mitsopoulos 97] LORENTZOS, Nikos A., MITSOPOULOS, Yannis G., "SQL Extension for Interval Data", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, Nº 3, Maio / Junho 1997.

[Lorentzos&Johnson 87] LORENTZOS, Nikos A., JOHNSON, Roger G., "TRA : A Model for a Temporal Relational Algebra", *Proc. Conf. Temporal Aspects in Information Systems*, C.Rolland, F. Bodart e M. Leonard eds, Maio 1987.

[McKenzie 86] MCKENZIE, E., "Bibliography : Temporal Databases", *Proc. ACM SIGMOD International Conference on Management of Data*, Vol. 15, Nº 4, Dezembro 1986.

[MicroStrategy 95] MICROSTRATEGY INC., *"The Case for Relational OLAP"*, White Paper, http://www.strategy.com, 1995.

[Musicant&Dabu 97] MUSICANT, Dave, DABU, Mihaela Patricia, *"Efficient Storage of Temporal Data in a Data Warehousing Context"*, Technical Report, Computer Science Department, University of Wisconsin-Madison, Dezembro 1997.

[Navathe&Ahmed 87] NAVATHE, S. B., AHMED, Rafi, "TSQL : A Language Interface for History Databases", *Proc. Conf. Temporal Aspects in Information Systems*, C.Rolland, F. Bodart e M. Leonard eds, Maio 1987.

[Navathe&Ahmed 93] NAVATHE, Shamkant B., AHMED, Rafi, "Temporal Extensions to the Relational Model and SQL", em [Tansel et al 93], 1993.

[Netz 99] NETZ, Amir, *"OLAP Services : Managing Slowly Changing Dimensions"*, White Paper, Microsoft Corporation, Abril 1999.

[Next Action 98] NEXT ACTION TECHNOLOGY, *"Strategies for Complex Analytical Applications – A Migration Towards ROLAP"*, White Paper, http://www.AnswerSets.com, Março 1998.

[OLAP Council 95] OLAP COUNCIL, *"OLAP and OLAP Server Definitions"*, White Paper, http://www.olapcouncil.org/research/glossaryly.htm, 1995.

[OLAP Council 97] OLAP COUNCIL, "*OLAP Council White Paper"*, White Paper, http://www.olapcouncil.org/research/whtpapco.htm, 1997.

[Oracle 99] ORACLE CORPORATION, *"Oracle8i for Data Warehousing: Fast and Simple for More Data and More Users"*, An Oracle Technical White Paper, Fevereiro 1999.

[Pendse 98] PENDSE, Nigel, "*What is OLAP? An Analysis of What the Increasingly Misued OLAP term is Supposed to Mean"*, White Paper, http://www.olapreport.com/FASMI.HTM, 1998.

[Peterson 94] PETERSON, Stephen, "*Stars : A Pattern Language for Query Optimized Schema"*, White Paper, Sequent Computer Systems Inc., http://c2.com/ppr/stars.html, 1994.

[Pimenta 98] PIMENTA, Sunil, "*Maintaining dimensional history"*, Eletronic Mail from pimenta@ssdi.sony.com.sg, 12 Novembro 1998.

[Pissinou 94] PISSINOU, Nikki, "Towards an Infrastructure for Temporal Databases", *SIGMOD Record*, Vol. 23, Nº 1, Março 94.

[POE et al 98] POE, Vidette, KLAUER, Patricia, BROBST, Stephen, *"Building a Data Warehouse for Decision Support"*, Prentice Hall PTR, 1998.

[Portfolio 99] PORTFOLIO, Tom, *"Oracle8i: PL/SQL Users Guide and Reference"*, Release 8.1.5, Part No. A67842-01, Oracle Corporation, Fevereiro 1999.

[Power 99] POWER, D. J., *"A Brief History of Decision Support Systems"*, White Paper, DSS Resources, http://dss.cba.uni.edu/dss/dsshistory.html, 1999.

[Ravat 99] RAVAT, Franck, TESTE, Olivier, ZURFLUH, Gilles, "Towards Data Warehouse Design", *Proc. 8th Int'l Conference on Information Knowledge Management (CIMK)*, Novembro 1999.

[Red Brick 95] RED BRICK SYSTEMS, "*Star Schemas and Star Join Technology"*, White Paper, Setembro 1995.

[Sanders 98] SANDERS, Richard, "*Managing SDC 's: another option ?"*, Eletronic Mail from Richard.Sanders@tlc.state.tx.us, 30 Abril 1998.

[Sarda 93] SARDA, N. L., "HSQL : A Historical Query Language", em [Tansel et al 93], 1993.

[Schiel 96] SCHIEL, Ulrich, *"Aspectos Temporais em Sistemas de Informação"*, Relatório Técnico, DSC/001/96, 2ª Edição, 1996.

[Segev&Shoshani 93] SEGEV, Arie, SHOSHANI, Arie, "A Temporal Data Model Based on Time Sequences", em [Tansel et al 93], 1993.

[Sen&Jacob 98] SEN, Arun, JACOB, Varghese S., "Industrial-strength : Data Warehousing, Why Process is so Important- and so often ignored", *Communications of the ACM*, Vol. 41, Nº 9, Setembro 1998.

[Snodgrass 95] SNOGRASS, Richard T., *"The TSQL2 Temporal Query Language"*, Kluwer Academic Publishers, 1995.

[Snodgrass et al 94] SNODGRASS, Richard T., "TSQL2 Language Specification", *SIGMOD Record*, Vol. 23, Nº 1, Março 1994.

[Soo 91] SOO, M., "Bibliography on Temporal Databases", *Proc. ACM SIGMOD International Conference on Management of Data*, Vol. 20, Nº 1, Março 1991.

[Souza 99] SOUZA, Aídre da Cunha Guedes de, *"Ambiente Computacional para Gerência Estratégia – Data Warehousing : Uma Survey"*, Dissertação de Mestrado, Departamento de Sistemas e Computação, Universidade Federal da Paraíba, Abril 1999.

[Sprague&Watson 91] SPRAGUE, Ralph H., WATSON, Hugh J., *"Sistema de Apoio à Decisão : Colocando a Teoria em Prática"*, Editora Campus, 1991.

[Stam&Snodgrass 98] STAM, R., SNODGRASS, R., "A Bibliography on Temporal Databases", *Database Engineering*, Vol. 7, Nº 4, Dezembro 1988.

[Stonebraker 96] STONEBRAKER, Michael, MOORE, Dorothy, "*Object Relational DBMS's : the Next Great Wave"*, Morgan Kaufmann Publishers Inc. , 1996.

[Tansel 93] TANSEL, Abdullah Uz, "A Generalized Framework for Modeling Temporal Data", em [Tansel et al 93], 1993.

[Tansel 97] TANSEL, Abdullah Uz, "Temporal Relational Data Model", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, Nº 3, Maio/Junho 1997.

[Tansel et al 93] TANSEL, Abdullah Uz, CLIFFORD, James, GADIA, Shashi K.,

JAJODIA, Shushil, SEGEV, Arie, SNODGRASS, Richard, "*Temporal Databases: theory, design and implementation*", The Benjamin/Cummings Publishing Company Inc., 1993.

[Tansel&Tin 97] TANSEL, Abdullah Uz, TIN, Erkan, "The Expressive Power of Temporal Query Languages", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 9, Nº 1, Janeiro/Fevereiro 1997.

[Thomsen 97] THOMSEN, Erik, "*OLAP Solutions : Building Multidimensional Information Systems*", John Wisley & Sons Inc., USA, 1997.

[Toman 96] TOMAN, David, "Point vs. Interval-based Query Languages for Temporal Databases", *Proc. ACM PODS* ,1996.

[Toman 97] TOMAN, David, "A Point-based Temporal Extension of SQL", *Proc. DOOD'97*, 1997.

[Tsotras&Kumar 96] TSOTRAS, Vassilis J., KUMAR, Anil, "Temporal Database Bibliography Update", *ACM SIGMOD Record*, Vol. 25, Nº 1, Março 1996.

[Widom 95] WIDOM, Jennifer, "Research Problems in Data Warehousing", *Proc. 4th Int'l Conference on Information and Knowledge Management (CIKM)*, Novembro 1995.

[Widom&Yang 97] WINDOM, Jennifer, YANG J., "Maintaining Temporal Views over Non-Historical Information Sources for Data Warehousing", Proc. *6th International Conference on Extending Database Technology*, Março 1998.

[Wu&Buchmann 97] WU, Ming-Chuan, BUCHMANN, Alejandro P., "Research Issues in Data Warehousing", *Proc. BTW'97*, Março 1997.

Apêndice A

Scripts PL/SQL

A.1 Implementação de IT

```
CREATE OR REPLACE TYPE IT AS OBJECT (

De NUMBER,
Ate NUMBER,

MEMBER FUNCTION Overlap (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Overlap, WNDS, WNPS),

MEMBER FUNCTION Overlap (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Overlap, WNDS, WNPS),

MEMBER FUNCTION Overlap (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Overlap, WNDS, WNPS),

MEMBER FUNCTION During (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (During, WNDS, WNPS),

MEMBER FUNCTION During (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (During, WNDS, WNPS),

MEMBER FUNCTION During (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (During, WNDS, WNPS),

MEMBER FUNCTION Adjacent (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Adjacent, WNDS, WNPS),

MEMBER FUNCTION Adjacent (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Adjacent, WNDS, WNPS),

MEMBER FUNCTION Adjacent (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Adjacent, WNDS, WNPS),

MEMBER FUNCTION Precedes (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Precedes, WNDS, WNPS),

MEMBER FUNCTION Precedes (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Precedes, WNDS, WNPS),
```

```
MEMBER FUNCTION Precedes (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Precedes, WNDS, WNPS),

MEMBER FUNCTION Follows (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Follows, WNDS, WNPS),

MEMBER FUNCTION Follows (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Follows, WNDS, WNPS),

MEMBER FUNCTION Follows (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Follows, WNDS, WNPS),

MEMBER FUNCTION Before (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Before, WNDS, WNPS),

MEMBER FUNCTION Before (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Before, WNDS, WNPS),

MEMBER FUNCTION Before (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Before, WNDS, WNPS),

MEMBER FUNCTION After (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (After, WNDS, WNPS),

MEMBER FUNCTION After (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (After, WNDS, WNPS),

MEMBER FUNCTION After (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (After, WNDS, WNPS),

MEMBER FUNCTION Equivalent (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Equivalent, WNDS, WNPS),

MEMBER FUNCTION Equivalent (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Equivalent, WNDS, WNPS),

MEMBER FUNCTION Equivalent (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Equivalent, WNDS, WNPS),

MEMBER FUNCTION Begins_Together (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Begins_Together, WNDS, WNPS),

MEMBER FUNCTION Begins_Together (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Begins_Together, WNDS, WNPS),

MEMBER FUNCTION Begins_Together (IT1 IN IT, De IN DATE, Ate IN DATE)
RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Begins_Together, WNDS, WNPS),

MEMBER FUNCTION Ends_Together (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Ends_Together, WNDS, WNPS),

MEMBER FUNCTION Ends_Together (IT1 IN IT, I IN NUMBER) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Ends_Together, WNDS, WNPS),
```

```
MEMBER FUNCTION Ends_Together (IT1 IN IT, De IN DATE, Ate IN DATE)
RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Ends_Together, WNDS, WNPS),

MEMBER FUNCTION Starts (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Starts, WNDS, WNPS),

MEMBER FUNCTION Starts (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR,
PRAGMA RESTRICT_REFERENCES (Starts, WNDS, WNPS),

MEMBER FUNCTION Ends (IT1 IN IT, IT2 IN IT) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Ends, WNDS, WNPS),

MEMBER FUNCTION Ends (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN CHAR,
PRAGMA RESTRICT_REFERENCES (Ends, WNDS, WNPS),

MEMBER FUNCTION ITUnion (IT1 IN IT, IT2 IN IT, ITResult OUT IT) RETURN
NUMBER,

MEMBER FUNCTION ITDur (IT1 IN IT, IT2 IN IT) RETURN NUMBER,
PRAGMA RESTRICT_REFERENCES (ITDur, WNDS, WNPS)

);


REM  Implementação dos métodos de IT

CREATE TYPE BODY IT AS


MEMBER FUNCTION Overlap (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF ((IT1.De <= IT2.Ate) AND (IT2.De <= IT1.Ate)) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Overlap;


MEMBER FUNCTION Overlap (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF ((IT1.De <= I) AND (I <= IT1.Ate)) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Overlap;
```

```
MEMBER FUNCTION Overlap (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF ((IT1.De <= AteK) AND (DeK <= IT1.Ate)) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Overlap;


MEMBER FUNCTION During (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF ((IT1.De >= IT2.De) AND (IT1.Ate <= IT2.Ate)) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END During;

MEMBER FUNCTION During (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF ((I >= IT1.De) AND (I <= IT1.Ate)) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END During;


MEMBER FUNCTION During (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF ((IT1.De >= DeK) AND (IT1.Ate <= AteK)) THEN
  RETURN('T');
ELSE
```

```
  RETURN('F');
END IF;

END During;


MEMBER FUNCTION Adjacent (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF ((IT2.De - IT1.Ate) = 1) OR ((IT1.De - IT2.Ate) = 1) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Adjacent;

MEMBER FUNCTION Adjacent (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF ((IT1.De - I) = 1) OR ((I - IT1.Ate) = 1) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Adjacent;


MEMBER FUNCTION Adjacent (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF ((DeK - IT1.Ate) = 1) OR ((IT1.De - AteK) = 1) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Adjacent;


MEMBER FUNCTION Precedes (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF (IT2.De - IT1.ATe) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
```

```
END IF;

END Precedes;

MEMBER FUNCTION Precedes (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF (I - IT1.ATe) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Precedes;


MEMBER FUNCTION Precedes (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF (DeK - IT1.ATe) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Precedes;


MEMBER FUNCTION Follows (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF (IT1.De - IT2.ATe) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Follows;

MEMBER FUNCTION Follows (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF (IT1.De - I) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;
```

```
END Follows;


MEMBER FUNCTION Follows (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF (IT1.De - AteK) = 1 THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Follows;


MEMBER FUNCTION Before (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF IT1.Ate < IT2.De THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Before;

MEMBER FUNCTION Before (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF IT1.Ate < I THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Before;


MEMBER FUNCTION Before (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);
```

```
IF IT1.Ate < DeK THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Before;


MEMBER FUNCTION After (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF IT1.De > IT2.Ate THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END After;

MEMBER FUNCTION After (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF IT1.De > I THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END After;


MEMBER FUNCTION After (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN CHAR
IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF IT1.De > AteK THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END After;


MEMBER FUNCTION Equivalent (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF (IT1.De = IT2.De) AND (IT1.Ate = IT2.Ate) THEN
```

```
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Equivalent;

MEMBER FUNCTION Equivalent (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF (IT1.De = I) AND (IT1.Ate = I) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Equivalent;


MEMBER FUNCTION Equivalent (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF (IT1.De = DeK) AND (IT1.Ate = AteK) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Equivalent;


MEMBER FUNCTION Begins_Together (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF IT1.De = IT2.De THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Begins_Together;


MEMBER FUNCTION Begins_Together (IT1 IN IT, I IN NUMBER) RETURN CHAR
IS

BEGIN

IF IT1.De = I THEN
```

```
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Begins_Together;


MEMBER FUNCTION Begins_Together (IT1 IN IT, De IN DATE, Ate IN DATE)
RETURN CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF IT1.De = DeK THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Begins_Together;


MEMBER FUNCTION Ends_Together (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF IT1.Ate = IT2.Ate THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Ends_Together;


MEMBER FUNCTION Ends_Together (IT1 IN IT, I IN NUMBER) RETURN CHAR IS

BEGIN

IF IT1.Ate = I THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Ends_Together;


MEMBER FUNCTION Ends_Together (IT1 IN IT, De IN DATE, Ate IN DATE)
RETURN CHAR IS

DeK NUMBER;
AteK NUMBER;
```

```
BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF IT1.Ate = AteK THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Ends_Together;


MEMBER FUNCTION Starts (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF (IT1.De = IT2.De) AND (IT1.Ate < IT2.Ate) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Starts;


MEMBER FUNCTION Starts (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN
CHAR IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF (IT1.De = DeK) AND (IT1.Ate < AteK) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Starts;


MEMBER FUNCTION Ends (IT1 IN IT, IT2 IN IT) RETURN CHAR IS

BEGIN

IF (IT1.De > IT2.De) AND (IT1.Ate = IT2.Ate) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Ends;
```

```
MEMBER FUNCTION Ends (IT1 IN IT, De IN DATE, Ate IN DATE) RETURN CHAR
IS

DeK NUMBER;
AteK NUMBER;

BEGIN

DeK := FUNCOES.DATETOKEY(De);
AteK := FUNCOES.DATETOKEY(Ate);

IF (IT1.De > DeK) AND (IT1.Ate = AteK) THEN
  RETURN('T');
ELSE
  RETURN('F');
END IF;

END Ends;

MEMBER FUNCTION ITUnion (IT1 IN IT, IT2 IN IT, ITResult OUT IT) RETURN
NUMBER IS

BEGIN

  IF OVERLAP(IT1, IT2) = 'T' THEN
    BEGIN
      ITResult.De:= FUNCOES.MINI(IT1.De, IT2.De);
      ITResult.Ate:= FUNCOES.MAXI(IT1.Ate, IT2.Ate);
      RETURN(0);
    END;
  ELSE
    RETURN(-1);
  END IF;
END ITUnion;

MEMBER FUNCTION ITDur (IT1 IN IT, IT2 IN IT) RETURN NUMBER IS


BEGIN
  IF BEFORE(IT1, IT2) = 'T' THEN
    RETURN(IT2.De - IT1.Ate);
  ELSE IF AFTER(IT1, IT2) = 'T' THEN
         RETURN(IT1.De - IT2.Ate);
       ELSE
         RETURN(0);
       END IF;
  END IF;
END ITDur;

END;
```

A.2 Implementação de FUNCOES

```
CREATE OR REPLACE PACKAGE Funcoes IS

FUNCTION KEYTODATE (tempo IN NUMBER) RETURN DATE;
```

```
PRAGMA RESTRICT_REFERENCES (KEYTODATE, WNDS, WNPS);

FUNCTION DATETOKEY (data IN DATE) RETURN NUMBER;
PRAGMA RESTRICT_REFERENCES (DATETOKEY, WNDS, WNPS);

FUNCTION MAXI (I1 IN NUMBER, I2 IN NUMBER) RETURN NUMBER;
PRAGMA RESTRICT_REFERENCES (MAXI, WNDS, WNPS);

FUNCTION MINI (I1 IN NUMBER, I2 IN NUMBER) RETURN NUMBER;
PRAGMA RESTRICT_REFERENCES (MAXI, WNDS, WNPS);

END Funcoes;


CREATE PACKAGE BODY Funcoes IS

IS

FUNCTION KEYTODATE( tempo in NUMBER ) RETURN DATE IS
data DATE;

BEGIN

  SELECT date_r INTO data
  FROM groceryi.time
  WHERE time_key = tempo;

RETURN(data);

END KEYTODATE;


FUNCTION DATETOKEY( data in DATE ) RETURN NUMBER IS
tempo NUMBER;

BEGIN

  SELECT time_key INTO tempo
  FROM groceryi.time
  WHERE date_r = data;

RETURN(tempo);

END DATETOKEY;


FUNCTION MAXI (I1 IN NUMBER, I2 IN NUMBER) RETURN NUMBER IS

BEGIN

  IF I1 >= I2 THEN
    RETURN(I1);
  ELSE
    RETURN(I2);
  END IF;

END MAXI;
```

```
FUNCTION MINI (I1 IN NUMBER, I2 IN NUMBER) RETURN NUMBER IS

BEGIN

  IF I1 <= I2 THEN
    RETURN(I1);
  ELSE
    RETURN(I2);
  END IF;

END MAXI;

END Funcoes;
```

A.3 Criação de Tabelas de Histórico

```
REM Criação da tabela de histórico do gerente

CREATE TABLE HISTORICO_GERENTE (
  LOJA#     NUMBER,
  NOME_LOJA VARCHAR(50),
  GERENTE   VARCHAR(50),
  PERIODO   IT,
  PRIMARY KEY (LOJA#, PERIODO.De)
);


CREATE TABLE HISTORICO_REFORMA (
  LOJA#        NUMBER,
  NOME_LOJA    VARCHAR(50),
  REFORMA      VARCHAR(50),
  DESCRICAO    VARCHAR(200),
  EMPRESA_RESP VARCHAR(50),
  ORCAMENTO    NUMBER,
  PERIODO      IT,
  PRIMARY KEY (LOJA#, PERIODO.De)
);
```