Rafael Mueller

Construção de um Cubo OLAP para Análise de Metas em Sistemas Baseados em Balanced Scorecard

Florianópolis
2006

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Construção de um Cubo OLAP para Análise de Metas em Sistemas Baseados em Balanced Scorecard

Trabalho de conclusão de curso apresentado
como parte dos requisitos para obtenção do
grau de Bacharel em Ciências da Computação.

Rafael Mueller

Florianópolis - SC

2006 / 07

Construção de um Cubo OLAP para Análise de Metas em Sistemas Baseados em BSC

Rafael Mueller

'Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.'

Prof. José Leomar Todesco, Dr.
Orientador

Banca Examinadora:

Prof. Frank Siqueira, Dr.

Marcelo Meyer

Denilson Sell

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No atual ambiente de negócios, com um altíssimo nível de competitividade, faz-se necessária agilidade e inteligência das organizações na busca de vantagens competitivas. Neste cenário, o conhecimento gerado pelo uso correto das informações disponíveis tem um papel importantíssimo como suporte à tomada de decisão tática e estratégica.

As primeiras tentativas de sistemas de informação que possam auxiliar a tomada de decisão remontam à década de 70, mas em sua maioria, esses sistemas não atingiram seus objetivos. Contudo, na medida em que o ambiente de negócios se torna mais competitivo e dinâmico, a procura e A necessidade por sistemas de informação crescem significativamente.

A maioria dos sistemas tradicionais de medição de desempenho utilizados pelas empresas utilizam valores e idéias da Era Industrial, ou seja, são baseados em indicadores financeiros adquiridos dentro de uma visão departamental.

A evolução cada vez maior da tecnologia possibilita a criação de sistemas de auxílio tomada de decisão muito maiores e mais eficazes do que os projetos pioneiros, que apesar do esforço, não atendiam as necessidades dos responsáveis pela tomada de decisão na organização, pois eram desenvolvidos utilizando tecnologias e conceitos apropriados para sistemas transacionais. A necessidade das grandes corporações de melhores ferramentas para suporte a tomada de decisão, levou ao desenvolvimento dos conceitos de *Data Warehouse* e de ferramentas OLAP que possibilitam o uso da informação para obter vantagens estratégicas.

Em muitos casos, a medição de desempenho é utilizada como um mecanismo de controle, e em muitos casos, assumindo um papel de repressão, limitando suas outras aplicações dentro de uma organização. Atualmente a medição de desempenho está tendo outra função além do papel de controle, pois está servindo como auxílio para a aprendizagem organizacional.

A principal função da medição de desempenho é permitir o uso eficaz das informações estratégicas da organização, permitindo que a organização tenha a flexibilidade exigida pelo mercado e auxiliando a tomada de decisão de forma correta.

Informações mensuráveis devem sempre servir como auxílio para o processo de tomada de decisão, pois tornan este processo menos subjetivo, permitindo que a organização se adeque à realidade em que se encontra.

Uma das metodologias para o gerenciamento de medições de desempenho utilizada atualmente é o *Balanced Scorecard*, "que é um sistema de gestão estratégica para administrar a estratégia a longo prazo" [**?**].

Kaplan e Norton [**?**] propõem transformar a estratégia em um processo contínuo, através da análise de informações sobre o desempenho de uma organização nas quatro perspectivas básicas do *Balanced Scorecard*. Contudo, essa análise só é possível de ser realizada, se, além dessas informações serem processadas e armazenadas, elas também puderem ser apresentadas de uma maneira flexível, consistente e ágil para quem irá tomar as decisões da organização.

O *Balanced Scorecard* é uma metodologia de gestão empresarial que mede o desempenho da organização. Além dos indicadores financeiros, índices de satisfação do cliente, eficácia dos processos internos e a capacidade de inovação da organização são medidos e comparados com o planejamento previamente feito. O conjunto de resultados obtidos por todas essas medições pode assegurar um melhor resultado financeiro e irá ajudar a organização a seguir na direção de seus objetivos estratégicos de longo prazo.

1.1 Justificativa

O *Balanced Scorecard* é empregado por diversas organizações de variados portes em todo mundo e vem ganhando aceitação cada vez maior. Um dos fatores para esse crescimento, é a tecnologia da informação (TI), que facilitou a implementação da metodologia e permitiu aplicá-la mesmo nas organizações mais complexas.

O problema que vem acontecendo nos sistemas que aplicam o *Balanced Scorecard*, é que a facilitadade gerada pela tecnologia da informação, vem criando sistemas de *Balanced Scorecard* cada vez maiores, com um enorme volume de dados. Contudo, nem sempre as ferramentas para análise desses dados, evoluíram na mesma velocidade.

Por isso, é necessário uma ferramenta que possa permitir a análise dos dados para o usuário final. Essa análise deve ser feita em cima de de toda massa de dados que sistemas de *Balanced Scorecard* com ajuda da tecnologia da informação geram.

1.2 Perguntas de Pesquisa

É possível que toda a informação gerada por sistemas de *Balanced Scorecard* seja apresentada de uma maneira ágil e flexível para os responsáveis pela tomada de decisão de uma organização?

Qual é a tecnologia mais adequada para auxiliar essa solução?

Como deve ser o processo de implantação dessa tecnologia em um sistema de *Balanced Scorecard* que já esta em funcionamento?

Como deve ser feita a migração dos dados do sistema já em funcionamento, para esse novo sistema?

1.3 Objetivos Gerais

Este projeto visa construir um *Data Mart* para dar suporte à tomada de decisão a partir de um sistema de *Balanced Scorecard*.

1.4 Objetivos Específicos

Pesquisar as diferentes abordagens para modelagem de dados em um *Data Mart*.

Desenvolver uma ferramenta de ETL (Extract Transform and Load) em JAVA que será responsável por migrar os dados da base de dados do sitema de *Balanced Scorecard* para o *Data Mart*.

Mapear os indicadores definidos no *Balanced Scorecard* na modelagem proposta.

1.5 Limitação de Escopo

O escopo desse projeto não abrange a ferramenta para acesso, visualização e análise da informação do CUBO. Não será construída nem utilizada nenhuma ferramenta de *front end*.

1.6 Estrutura do Trabalho

Para alcançar os objetivos propostos, esse trabalho foi dividido em capítulos relacionados aos temas envolvidos.

O segundo capítulo apresenta o conceito de *Balanced Scorecard*, com uma descrição do histórico e de sua criação. Em seguida é apresentado o modelo de *Balanced Scorecard*.

No terceiro capítulo é apresentada a teoria sobre *Data Warehouse*. Primeiramente é apresentado o conceito teórico e suas características. Em seguida são apresentadas as questões referentes à arquitetura. Segue-se apresentando um tópico sobre as opções de modelagem de dados e as partes finais desse capítulo referem-se ao projeto físico e à ferramenta de ETL.

No quarto capítulo são apresentadas quais as etapas e como foi desenvolvido o projeto do CUBO e são apresentados os resultados do projeto.

No quinto e último capítulo são apresentadas as conclusões finais e as recomendações para trabalhos futuros.

Capítulo 2

Balanced Scorecard

2.1 Introdução

Sistemas de medição de desempenho tradicionais são insuficientes para medir a desempenho e guiar a organização no cenário atual com uma economia complexa e em constante mudança. As organizações precisam associar medidas de desempenho com estratégia para promover resultados futuros e comparar com a desempenho do passado.

O *Balanced Scorecard* é uma poderosa metodologia para implementar a estratégia e continuamente monitorar a desempenho. Criar uma organização focada na estratégia é um desafio, uma mudança cultural para muitas organizações. Para o sucesso na realização dessas mudanças é necessário:

- Suporte e envolvimento dos executivos da empresa. Educação, comunicação e visibilidade da estratégia e medidas de sua eficácia através da organização.
- Ferramentas para habilitar que usuários não técnicos entendam as medidas de desempenho.
- Tradução da estratégia para termos operacionais, assim o alinhamento da estratégia e a implementação devem ocorrer em todos os níveis da organização.

Organizações que têm implementado com sucesso o *Balanced Scorecard* têm conseguido grandes transformações em sua desempenho financeira.

Muitos aspectos do desenvolvimento do *Balanced Scorecard* depende do uso eficiente da tecnologia, de ferramentas para auxiliar na implementação e medição da estratégia.

2.2 Histórico

O *Balanced Scorecard* surgiu entre o final dos anos 80 e início dos anos 90 como uma metodologia para auxiliar as organizações a controlar seu complexo e crescente ambiente de negócio.

As organizações estavam enfrentando muitos desafios. Suas fatias de mercado estavam desaparecendo rapidamente devido a globalização, inovação tecnológica e acordos econômicos. A economia estava numa transição de uma economia voltada para o produto para uma economia voltada para o serviço. A economia estava mudando, e as organizações precisam seguir essa mudança.

Contra todas essas mudanças, a maior parte das organizações ainda é baseada no tradicional modelo de medição de desempenho baseada em velhos modelos que falharam em apontar a verdadeira situação da organização. A necessidade de melhores informações para responder a rápida mudança das condições do mercado era óbvia.

Em resposta a esta situação, Robert Kaplan e David Norton começaram a formar o conceito de *Balanced Scorecard* durante um projeto de pesquisa com 12 organizações no final dos anos 80. Eles entenderam as limitações que ocorrem quando uma organização confia demais apenas em medidas financeiras. Eles descobriram que muitos métodos usados para melhorar a desempenho financeira a curto prazo (como demissão de funcionários, redução da quantidade de treinamento para os funcionários, propaganda, entre outros) podem na verdade prejudicar financeiramente a organização a longo prazo. Inversamente a esta situação, organizações que apresentam resultados financeiros modestos porque estão investindo na organização, tendem a ter uma melhor desempenho futura. Além disso, eles perceberam a limitação de confiar demais em indicadores que são usados para comparar com situações anteriores da organização mas que geralmente não provêem nenhuma indicação confiável do desempenho futuro.

Kaplan e Norton também perceberam que funcionários da organização geralmente não entendem qual a sua função em relação à estratégia, deixando os empregados sem condições de ajudar na melhoria do que estava sendo medido.

Kaplan e Norton introduziram o *Balanced Scorecard* como uma maneira para muitas organizações medir seu desempenho em uma maneira balanceada:

- Múltiplas Perspectivas
- Medição desde a produtividade interna até a satisfação do cliente

O resultado inicial de sua pesquisa com as 12 organizações foi publicado em 1992. Impulsionados pelas críticas positivas sobre seu artigo inicial, e pelo sucesso de sua consultoria, Kaplan e Norton continuaram a desenvolver o conceito de *Balanced Scorecard* e publicaram o livro *The Balanced Scorecard* em 1996. Nesta época, o foco do *Balanced Scorecard* havia evoluído para uma metodologia, promovendo gestão estratégica para a organização.

Como mais e mais organizações iniciaram a adoção do *Balanced Scorecard*, um grande número de ferramentas e técnicas foram sendo criadas, baseadas em muitos dos conceitos iniciais. No ano 2000, Norton e Kaplan lançaram seu segundo livro, *The Strategy Focused Organizaion*, onde é descrita a evolução de um conceito da gestão estratégica de uma empresa.

O *Balanced Scorecard* é uma metodologia dinâmica, e o entendimento de seu potencial aprofunda-se assim que Kaplan e Norton proseguem com um trabalho inovador, como desenvolvendo *scorecards* para auxiliar funções como recursos humanos e tecnologia da informação.

2.3 O Modelo

Segundo Kaplan e Norton [**?**], o *Balanced Scorecard* é uma metodologia de gestão estratégica que auxilia as organizações a traduzirem suas estratégias em ações. O *Balanced Scorecard* trata de modo balanceado indicadores financeiros e não financeiros, como índice de satisfação do cliente, eficácia dos processos operacionais internos e a capacidade

de inovação. Os resultados obtidos em todas essas áreas (e não apenas na área financeira) podem assegurar a continuidade de bons resultados financeiros e podem conduzir a organização na direção correta para alcançar seus objetivos estratégicos de longo prazo.

O equilíbro das quatro perspectivas analisadas através do processo, permite que os funcionários treinados na metodologia compreendam as conseqüências financeiras de suas ações e decisões. As medidas e os objetivos utilizados no *Balanced Scorecard* devem derivar de um processo hierárquico, de cima para baixo, guiado pela missão e visão do futuro da organização. É necessário para os processos de implementação e utilização do *Balanced Scorecard* o acompanhamento constante e o empenho efetivo da direção da organização.

O *Balanced Scorecard* trata de uma mudança de paradigma, que exige uma postura diferente de todos que compõem a organização, pois exibe a eficácia das ações que levaram a um determinado balanço financeiro, identificando os responsáveis por esse resultado. Depois de definidas a missão e a visão da organização como um todo, é necessário que cada departamento estabeleça suas missões e visões, e que essas missãoes e visãoes sejam conhecidas por todos da organização, tentando estabelecer uma rede de compromissos, permitindo uma cobrança de resultados, sem que haja a possibilidade de alegar desconhecimento da missão e visão.

O *Balanced Scorecard* possibilita esclarecer e traduzir a missão e a visão da organização, que é iniciado com uma equipe traduzindo a estratégia da organização em objetivos estratégicos específicos. Além disso, possibilita planejar, estabelecer metas e alinhar as iniciativas estratégicas, já que as metas que permitem a evolução da organização são traçadas conjuntamente com de três a cinco anos de antecedência.

O desdobramento de objetivos é feito de cima para baixo, e em cada passo é necessário definir quais os índices - cada qual em sua perspectiva, tema estratégico e objetivo estratégico - permitindo a mensuração do cumprimento dos objetivos em relação ao planejado.

2.4 As Quatro Perspectivas

As metas de curto prazo e as medidas de desempenho devem ser divididas em, pelo menos, quatro perspectivas. São elas: a) a financeira; b) a do cliente; c) dos processos

internos; e d) do aprendizado e crescimento.

2.4.1 Perspectiva Financeira

O *Balanced Scorecard* deve servir de incentivo para que as unidades de negócio vinculem seus objetivos financeiros à estratégia da empresa. Qualquer indicador selecionado deve fazer parte de uma cadeia de causa e efeito que culmine com a melhoria do desempenho financeiro. Ao selecionar indicadores financeiros a serem usados no *Balanced Scorecard*, deve-se ter em mente dois objetivos: definir o desempenho financeiro esperado da estratégia e servir de meta principal para os objetivos e medidas de todas as outras perspectivas do *Balanced Scorecard*.

A escolha dos indicadores financeiros depende da fase do ciclo de vida (tabela 2.1) em que a empresa ou unidade de negócios se encontra. Kaplan e Norton [**?**] apontam três fases no ciclo de vida: crescimento, sustentação e colheita. O ciclo de vida se inicia com a fase de crescimento, na qual são necessários elevados níveis de investimento para criar a infra-estrutura, implantar os processos internos necessários ao funcionamento da empresa e ampliar rapidamente a fatia de mercado. A fase de sustentação caracteriza-se pela busca da lucratividade e retorno do capital investido, melhoria dos processos internos, e é a fase na qual a maior parte das empresas e unidades de negócio se encontram. Quando a empresa atinge maturidade, passa para a última fase do ciclo, na qual a meta principal é a maximização do fluxo de caixa operacional em benefício da empresa e diminuição da necessidade do capital de giro. É responsabilidade dos gerentes identificar em que fase encontra-se a sua empresa ou unidade de negócios para estabelecer os objetivos e indicadores financeiros adequados, levando em conta os temas estratégicos da empresa (aumento de receita, redução de custos e utilização dos ativos).

Tabela 2.1: Ciclo de Vida

Fase	**Características**	**Objetivos Financeiros**
Crescimento	• Investimentos elevados em infra-estrutura. • Criação/consolidação dos processos internos • Desenvolvimento da base de clientes	• Velocidade de crescimento da receita (aumento das vendas) em mercados previamente determinados
Sustentação	• Retorno sobre o capital investido • Investimento visando melhoria contínua dos processos internos • Ampliação gradual da capacidade de produção	• Lucratividade • Aumento da receita operacional e margem bruta • Aumento da razão receita contábil sobre capital investido • Aumento do valor econômico agregado

continua na próxima página

Tabela 2.1 – continuação da página anterior

Fase	**Características**	**Objetivos Financeiros**
Colheita	• Colheita dos investimentos realizados nas etapas anteriores • Realização de investimentos somente para manter os equipamentos e capacidades existentes ou com retorno rápido certo • Redução de despesas em Pesquisa e Desenvolvimento	• Maximizar os fluxos de caixa • Diminuição da necessidade de capital de giro

2.4.2 Perspectiva do Cliente

O propósito da perspectiva do cliente no *Balanced Scorecard* é identificar os melhores segmentos de clientes nos quais competir. Provavelmente nenhuma empresa conseguirá ser eficiente se tentar cobrir um leque exageradamente extenso de segmentos. Além disso, cada um deles tem uma reciprocidade particular em termos da lucratividade que é capaz de gerar. Assim, como regra geral, é possível dizer que as organizações devem enfocar os segmentos de clientes que proporcionam as melhores margens de lucro em detrimento dos segmentos menos lucrativos [**?**]. Entretanto é igualmente importante perceber que a contribuição de um segmento de clientes para a organização extrapola a perspectiva financeira, embora deva, ao final, resultar em vantagem financeira de algum modo. Por isso, pode ser interessante adotar indicadores que reflitam contribuições de cada segmento de clientes em perspectivas não financeiras.

Para avaliar a perspectiva do cliente, Kaplan e Norton [**?**] sugerem um grupo de medidas essenciais que incluem a participação de mercado, retenção, captação, satisfação e lucratividade de clientes. Uma dimensão básica no relacionamento dos clientes com seus fornecedores é a proposta de valor, que se refere a um conjunto de atributos dos produtos ou serviços da organização capazes de atrair o interesse dos clientes e resultar em bons indicadores nas medidas acima mencionadas. A proposta de valor pode contemplar três categorias: a) proposta de valor nos atributos dos produtos ou serviços - referem-se a funcionalidades, qualidade e preço dos produtos ou serviços da organização para o cliente; b) proposta de valor no relacionamento com clientes - refere-se a capacidade da organização de perceber as necessidades dos clientes e agir de acordo com essas percepções; e c) proposta de valor na imagem e reputação - refere-se à capacidade da organização de comunicar-se com o público e persuadí-lo quanto às vantagens de realizar negócios com ela [**?**].

Os clientes são fontes de valor para a organização também em aspectos não financeiros. Eles oferecem treinamento para os funcionários, incentivando a competência interna com as suas exigências. Ao conversarem uns com outros, estão fazendo propaganda e trabalhando para a formação de uma imagem institucional [**?**]. Por isso, é importante analisar o fluxo de conhecimento gerado pelos clientes para a organização e vice-versa.

A percepção dos clientes em relação à proposta de valor da organização é influenciada por fatores culturais do cliente: status sócio-econômico, sua sensibilidade aos instrumentos de marketing e suas necessidades prioritárias no momento. De um ou de outro modo, é preciso buscar a comunicação com o cliente como forma de captar informações a seu respeito nessas quatro áreas.

2.4.3 Perspectivas dos Processos Internos

O propósito da perspectiva dos processos internos é identificar os processos mais críticos para a realização dos objetivos dos acionistas e dos clientes, e tratar esses processos adequadamente. Enquanto nos modelos tradicionais as medições de desempenho estão focadas na estrutura produtora, no *Balanced Scorecard* há uma preocupação de criar medidas para avaliar o desempenho do ciclo inovação/operação/pós-venda, atravessando toda a organização (cadeia de valor) [**?**]. Cadeia de valor é a seqüência de transformações pelas quais passam

os insumos do processo, ganhando gradativamente mais valor para o cliente. A cadeia de valor pode ser dividida em três fases:

- inovação - é a fase de detecção e análise das necessidades dos clientes, das condições de mercado, da formalização de alternativas e desenvolvimento de soluções
- operações - é a fase em que ocorre a geração do produto ou serviço
- serviço de pós-venda - esta fase abrange o período posterior à venda, no qual são realizados procedimentos relacionados com a garantia, consertos, devoluções, processamento de pagamento e apoio ao cliente

2.4.4 Perspectiva de Aprendizado e Crescimento

De acordo com Kaplan e Norton [**?**], uma das mudanças mais radicais no pensamento gerencial dos últimos anos foi a transformação do papel dos funcionários, que passou de provedor de força física a analisador de dados cada vez mais abstratos, muitas vezes captados em ambientes automatizados. Todos os ativos e estruturas, quer tangíveis ou intangíveis, são resultados das ações humanas; dependem, em última instância, das pessoas para existir. Adicionalmente, os autores do *Balanced Scorecard* evidenciam que as idéias que permitem melhorar o desempenho para os clientes emanam, cada vez mais, dos funcionários operacionais, que atuam mais diretamente nos processos internos e junto aos clientes [**?**]. Esta mudança de perspectiva implica em um recrutamento mais criterioso, para obter funcionários com maior capacidade analítica. São três as principais categorias de indicadores para a perspectiva de aprendizado e crescimento, de acordo com a experiência dos idealizadores do *Balanced Scorecard*: a) capacidade dos funcionários; b) capacidades dos sistemas de informação; e c) motivação e alinhamento [**?**]. Um grupo essencial de indicadores monitora os resultados dos investimentos feitos em funcionários, sistemas e alinhamento organizacional.

2.5 Conclusões

O *Balanced Scorecard* é uma metodologia para implementar a estratégia e continuamente monitorar a desempenho de uma organização. Diferentemente de outros sistemas de gestão

empresarial, o *Balanced Scorecard* busca um melhor resultado financeiro através da melhoria em diversas áreas de empresa.

Para conseguir um melhor resultado financeiro, devem ser medidos indicadores nas quatro perspectivas. Na perspectiva financeira, os objetivos financeiros representam a meta de longo prazo da organização: gerar um retorno maior do que o capital investido. A partir deles, todos os objetivos e medidas das outras perspectivas deverão estar relacionadas à execução de um ou mais objetivos desta perspectiva. Toda medida deve fazer parte de uma relação de causa e efeito que termina com objetivos financeiros. Na perspectiva cliente o objetivo é definir em qual segmento do mercado a organização irá competir. Na perspectiva dos processos internos, as medidas são escolhidas de maneira a proporcionar a excelência nos processos que são críticos para atingir a estratégia estabelecida. Na perspectiva de aprendizado e crescimento, busca-se estabelecer a infra-estrutura necessária para suportar os objetivos elaborados pelos processos internos.

O *Balanced Scorecard* com todos os indicadores definidos e seus resultados gera uma grande quantidade de informações sobre os setores de um organização, gerando um problema que é permitir a manipulação e a análise de toda essa informação.

Capítulo 3

Data Warehouse

3.1 Conceito

Segundo Inmon [**?**] o *Data Warehouse* é uma coleção de dados orientada por assuntos, integrada, variante no tempo, e não volátil. Pode ser considerado como um conjunto de *snapshots* de dados extraídos do ambiente de produção da empresa, em determinado intervalo de tempo que foram selecionados e depurados, tendo sido otimizados para processamento de consulta e não para processamento de transações. Em geral, um *Data Warehouse* requer a consolidação de outros recursos de dados além dos armazenados em banco de dados relacionais, incluindo informações provenientes de planilhas eletrônicas, documentos textuais entre outros [**?**]. Ou seja, o *Data Warehouse* é um banco de dados usado unicamente para a produção de relatórios, o que contrasta com os bancos de dados tradicionais.

Os sistemas operacionais são baseados em transações e são conhecidos como OLTP (On Line Transaction Processing). Estes sistemas são configurados e otimizados alteração e exclusão de dados, tornando tais transações rápidas e confiáveis. Os dados são dinâmicos e mudam com grande freqüência [**?**]. Segundo Singh [**?**], *Data Warehouse* é uma tecnologia de gestão e análise de dados, constituindo ”um ambiente de suporte a decisão que alavanca dados armazenados em diferentes fontes e os organiza e entrega aos tomadores de decisões da empresa, independente de plataforma que utilizam ou de seu nível de qualificação técnica”. O principal objetivo do *Data Warehouse* é suportar todas as exigências analíticas relacionadas às necessidades de gerenciamento da empresa, que sejam críticas para sua competi-

tividade, ao invés de simplesmente enfocar problemas operacionais. Um *Data Warehouse* bem projetado contém os dados necessários para solucionar problemas de análise empresarial quanto a questões sobre: ”O que?”, ”Quando?”, ”Por quê?”e ”O quê se?”, eliminando a possibilidade de um fim prematuro de uma pesquisa, pela falta de determinada informação ou de tempo para o processamento [**?**].

3.2 Características

3.2.1 Orientado ao Assunto

Os bancos de dados transacionais, em geral, armazenam informações sobre subconjuntos da empresa. A maioria das empresas possuem sistemas específicos para cada foco de atuação. Ou seja, cada um dos bancos de dados transacional trabalha com a sua base, seu devido subconjunto de informações, mas não há uma visão do assunto macro, como a quantidade vendida de um determinado produto. Os bancos de dados OLAP devem ser voltados para os assuntos da empresa de uma forma macro, contendo informações sobre as vendas, independentemente da origem (sem descartar a informação da origem da venda) e informações de todos os clientes e produtos, gerando várias tabelas relacionadas no data warehouse [**?**][**?**].

3.2.2 Integrado

Os bancos de dados OLAP são oriundos de diversas fontes de informações. Contudo, como as bases se encontram em locais, formatos, origens e com codificações diferentes é necessário um processo de transformação dos dados, eixando todos em um mesmo formato, que fará com que estes dados se tornem integrados [**?**].

3.2.3 Não Volátil

O *Data Warehouse* é um banco de dados somente para consultas, no qual se mantém um registro histórico dos sistemas transacionais. Desta forma, seria incorreto permitir manipulações

de dados no data warehouse. Isso seria, segundo Corey [**?**], reescrever a história.

3.2.4 Histórico

No *Data Warehouse*, o elemento tempo é fundamental. O ambiente do *Data Warehouse*, é destinado a geração de relatórios, o histórico é algo de suma importância para auxiliar o estudo de tendências [**?**]. Sistemas operacionais são projetados para respostas rápidas, ou seja os dados serão eliminados assim que não forem mais necessários, ignorando a possibilidade de consultas futuras. Nem sempre os dados históricos são armazenados com o mesmo nível de granularidade de informação que é utilizado no banco de dados transacionas. Muitas vezes trabalha-se com dados resumidos, como resumos semanais, mensais ou anuais. A medida que os dados vão se tornando mais velhos dentro do *Data Warehouse*, pode-se diminuir a granularidade destes dados antigos [**?**].

3.3 Por que o *Data Warehouse*?

Os sistemas de banco de dados operacionais, em sua maioria, não suportam todos os quatro critérios de um *Data Warehouse*, pois trabalham com sistemas orientados a transações (*On-Line Transaction Processing* - OLTP), que tem como objetivo facilitar as principais operações da empresa. O *Data Warehouse* tem seu foco em sistemas capazes de fornecer rapidamente informações que possam ajudar a tomada de decisão. Esses sistemas são conhecidos como ferramentas ou sistemas OLAP.

Projetos de banco de dados OLTP são altamente normalizados e complexos. Caso seja necessário algum relatório sobre todas as vendas de algum determinado período, o tempo de espera para geração do relatório seria inaceitável, além da enorme dificuldade que um sistema OLTP pode causar na geração de um relatório complexo e até a impossibilidade da geração do relatório (Impossibilidade gerada pelo fato que os dados necessários para a geração do relatório estarem em fontes distintas).

Em um sistema de banco de dados operacional é necessário que as informações sejam atualizadas em tempo real. Se alguma característica do produto for alterada, esta alteração

deve refletir em todos os sistemas. No *Data Warehouse* é importante que que determinadas mudanças sejam acompanhadas, para que possa permitir a análise de tendência do produto com essa nova característica [**?**].

Os sistemas de banco de dados operacionais são projetados para uma entrada rápida dos dados. Assim que a informação é lida, ela deve ser atualizada na base e a resposta deve ser dada ao usuário. No *Data Warehouse* os dados são migrados, e geralmente existe uma janela de tempo grande para a atualização das bases, onde a ênfase é a resposta rápida à pesquisa em quantidades enormes de informações.

Os padrões de utilização dos sistemas de banco de dados operacionais não sofrem grandes alterações. Os sistemas de data warehouse não possuem padrões de utilização, pois em qualquer momento pode haver alguma reunião da diretoria onde será necessário fazer pesquisas intensas por algumas horas, e depois o data warehouse pode ficar quase que inativo por alguns dias.

Na tabela 3.1 há um comparativo entre sistemas de banco de dados operacionais (OLTP) e um *Data Warehouse* (OLAP).

Tabela 3.1: Comparativo OLTP x OLAP

OLTP	**OLAP**
Orientado a transações	Orientado a consultas
Facilitam as operações do dia-a-dia da empresa, automatizando algumas tarefas.	Auxilia na tomada de decisão, comparando padrões e procurando tendências.
Trabalham com um grande número de transações rápidas e acessos a registros individuais.	Trabalha sumarizando informação, em grupos de informação.
Tempo de resposta em poucos segundos.	Tempo de resposta pode ser de até alguns poucos minutos.
Acessa a informação segundo um padrão.	O acesso a informação não segue um padrão, variando para cada relatório visualizado.
continua na próxima página	

Tabela 3.1 – continuação da página anterior

OLTP	**OLAP**
Otimizado para que o dado esteja sempre disponível e para atender as transações.	Otimizado para facilitar a iteração com o usuário e para o uso de grande quantidade de dados.
Utilizado por operadores treinados, geralmente incumbidos da entrada de dados.	Utilizado pelas pessoas que tem o poder de tomada de decisão dentro da instituição.
Os usuários usam os aplicativos predefinidos.	Os usuários podem gerar consultas *ad-hoc*.
Foco na consistência dos dados.	Foco na confiabilidade dos dados.
Volume de dados entre pequeno e médio.	Enorme volume de dados.
Guarda informações detalhadas.	Guarda informações detalhadas e sumarizadas.
Os dados são atualizados a qualquer instante.	Os dados são atualizados de período estipulados.
Dados atômicos, concorrentes, isolados.	Dados integrados, históricos e podem ser sumarizados.

3.4 Arquitetura

3.4.1 Componentes

É possível analisar os componentes do *Data Warehouse* com relação aos seguintes aspectos: papéis, ferramentas/processos envolvidos e dados.

3.4.1.1 Papéis

No projeto de um *Data Warehouse*, profissionais de diversas áreas estão envolvidos, desde profissionais em processamento de dados, analistas de negócios até o usuário que irá utilizar o sistema. Entre estes profissionais estão os administradores do projeto, administradores de banco de dados dos sistemas de origem, os projetistas do data warehouse, os desenvolvedores e analistas das ferramentas de ETL e das ferramentas de acesso ao data warehouse e os usuários finais. Os papéis podem ser divididos em:

- Carga de dados: desenvolvedores que fazem o mapeamento entre os sistemas operacionais e o *Data Warehouse* e também são responsáveis pela filtragem e a integração dos dados.
- Usuário final: são os responsáveis pela tomada de decisão na instituição, que utilizam a informação para dar suporte a essa tomada de decisão. Estes usuários apresentam uma grande familiaridade com os termos do negócio e estão sempre em busca da solução de um problema ou de novas oportunidades [**?**]. Estes usuários podem ser divididos em dois grupos: os usuários diretos e os usuários indiretos. Os usuários diretos são aqueles que acessam livremente o *Data Warehouse* enquanto os indiretos acessam os *Data Marts* especializados [**?**].
- Desenvolvimento e manutenção do *Data Warehouse*: são os administradores de dados e administradores de banco de dados dos sistemas operacionais. São responsáveis pela arquitetura interna do banco de dados para o armazenamento e pelo desempenho das consultas.

3.4.1.2 Ferramentas e processos envolvidos

Os processos do *Data Warehouse* consistem em extrair os dados das fontes, organizar e integrar esses dados de forma consistente e no acesso destes dados de forma integrada e flexível. Esses processos devem ser feitos com o propósito de garantir a consistência e a integridade das informações. Normalmente é necessário a compra ou desenvolvimento de sistemas para a extração e atualização dos dados do *Data Warehouse*. Esses sistemas geralmente são responsáveis pela filtragem, sumarização, limpeza e concentração dos dados que

estão em fontes diversificadas. É necessário que os analistas envolvidos tenham conhecimento tanto das fontes de onde serão extraídas as informações, como da base de dados onde serão armazenadas[**?**].

As ferramentas que o usuário irá utilizar devem permitir um fácil acesso aos dados, possibilitando que sejam facilmente analisado os dados com maior importância. O sucesso de um data warehouse pode depender da disponibilidade de ferramentas adequadas para as necessidades de cada usuário. Para que seja possível essa flexibilidade, normalmente são empregados [**?**]:

- Ferramentas para relatórios: ferramentas simples que permitem que o usuário faça a análise dos dados e tendência, contudo não permite muita flexibilidade na visualização da informação, pois a informação retornada seguirá sempre o padrão de cada relatório criado.

- Ferramentas OLAP: essas ferramentas também permitem que o usuário faça a análise dos dados e procure por tendências, contudo elas podem apresentar a informação de uma forma mais dinâmica, pois os dados podem ser apresentados para o usuário em diferentes padrões e formatos. Existem algumas abordagens diferentes em relação às ferramentas OLAP, entre elas estão:

    1. ROLAP (OLAP Relacional): ferramentas OLAP que acessam bancos de dados em um banco relacional.
    2. MOLAP (OLAP Multidimensional) : ferramentas OLAP que acessam bancos de dados multidimensionais, que suportam o conceito de cubo e hipercubo.
    3. HOLAP (OLAP Híbrida): ferramentas que permitem o acesso tanto a banco de dados relacionais como a banco de dados multidimensionais.

- Sistemas de informações executivas: apresentam as informações de forma consolidada e com uma interface mais simples: não requerem do usuário experiência e tempo para analisar os dados.

3.4.1.3 Dados

Os dados podem estar em diferentes tipos de sintetização, como: dados detalhados, dados levemente sintetizados e dados altamente sintetizados ou podem ser metadados [**?**].

Os dados detalhados são os que exigem maior atenção, pois refletem os acontecimentos mais atuais da instituição, que são sempre de grande interesse. Por serem os mais detalhados, também são os mais volumosos e geralmente são armazenados em disco, um acesso rápido mas de gerenciamento difícil e caro [**?**].

Os dados levemente sintetizados são os dados que sofreram alguma sumarização a partir dos dados detalhados. Neste nível, o responsável por essa sintetização geralmente enfrenta algumas questões como [**?**]:

- Em qual janela de tempo o resumo será feito?
- Qual conteúdo deve ser sintetizado, e qual deve ser mantido detalhado?

Dados altamente sintetizados são de fácil acesso e compactos. Contudo não necessariamente esses dados altamente sintetizados ficam armazenados dentro do data warehouse. [**?**].

Metadados são informações sobre os dados do data warehouse. Por conter informações sobre os dados, essas informações não são retiradas das fontes de origem de dados. Entre as informações que os metadados armazenam, estão[**?**]:

- A estrutura dos dados.
- Os algoritmos usados na sintetização.
- As fontes de dados do data warehouse.
- O modelo de dados.
- O relacionamento entre o modelo de dados e o data warehouse.
- O histórico das migrações das fontes para o data warehouse.

Os metadados provêem informações que auxiliam a localização dos dados no data warehouse, o mapeamento dos dados desde a sua extração das fontes até a transformação e o armazenamento no data warehouse e para auxiliar os algoritmos de sintetização, informando quais devem ser mais sintetizados.

3.4.2 *Data Mart*

O *Data Mart* é um conjunto de dados de um determinado assunto e baseado na necessidade de um determinado departamento, organizado para auxiliar a tomada de decisão de uma área específica do negócio [**?**]. Dentro de uma instituição pode existir um *Data Mart* para o departamento de marketing, outro para o departamento de vendas e outro para o departamento de recursos humanos, por exemplo.

Dependendo da arquitetura escolhida para a construção do *Data Warehouse* o *Data Mart* pode ser classificado como *Data Mart* Dependente ou *Data Mart* Independente.

Em uma arquitetura com *Data Marts* independentes, cada *Data Mart* possui dados específicos, podendo tornar difícil a localização de uma determinada informação e causando redundância de informações.

Em uma arquitetura com *Data Marts* dependentes, existe um *Data Warehouse* como centralizador de todos os dados. A fonte de dados dos *Data Marts* dependentes é esse *Data Warehouse*. As principais diferenças entre *Data Mart* dependente e *Data Mart* independente estão listadas na tabela 3.2

As principais características de um *Data Mart* segundo Inmon [**?**] são:

- São específicos para cada assunto ou área da empresa.
- Normalmente são modelados utilizando o esquema estrela.
- Geralmente apresentam uma granularidade menor que a granularidade do *Data Warehouse*
- Tem bons resultados quando armazenados em um Sistema Gerenciador de Banco de Dados Multidimensional (SGBDM), pois estes fornecem uma flexibilidade melhor,

só que possuem alguns problemas quando trabalham com grandes quantidades de informações, como em um *Data Warehouse*

- Os dados são armazenados em um ambiente altamente indexado.

Tabela 3.2: Comparação entre *Data Mart* dependente e *Data Mart* Independente

***Data Mart* Dependente**	***Data Mart* Independente**
Tem o *Data Warehouse* como fonte de dados.	Tem os sistemas operacionais como fonte de dados.
Todos os *Data Marts* são atualizados da mesma fonte, o *Data Warehouse*.	Cada *Data Mart* é carregado a partir dos sistemas operacionais que possuem informações sobre o seu assunto.
Arquitetura que permite uma fácil expansão.	Arquitetura que dificulta a expansão

O problema dos *Data Marts* Independentes é que suas deficiências geralmente são notadas apenas depois da construção de alguns *Data Marts*.

A tabela 3.3 apresenta as principais diferenças entre um *Data Warehouse* e um *Data Mart*[**?**].

Tabela 3.3: Diferenças entro *Data Mart* e *Data Warehouse*

Data Warehouse	***Data Mart***
Corporativo	Departamental
Granularidade baixa, dados detalhados.	Dados sintetizados, granularidade alta.
Estrutura normalizada.	Utiliza o esquema estrala como estrutura de dados.
continua na próxima página	

Tabela 3.3 – continuação da página anterior

Data Warehouse	***Data Mart***
Excelente para processos de exploração.	Excelente para consultas.
Grande volume de dados	Por ser orientado para um assunto e por possuir uma granularidade maior, o volume de dados é menor.
Utiliza tecnologia para armazenamento de grandes quantidades de dados.	Utiliza tecnologia para melhor analise de dados - multidimensional.
Levemente indexado	Altamente indexado

3.4.3 Tipos de Arquitetura

Os modelos de arquitetura para um *Data Warehouse* ainda estão evoluindo, não existindo apenas um modelo que seja considerado correto. Isto ocorre devido a dificuldade de integração de todos os componentes e a complexidade crescente de sistemas de *Data Warehouse*.

3.4.3.1 Arquitetura *Top-Down*

Essa abordagem foi introduzida por Inmon [**?**], sendo hoje a mais conhecida e considerada como a arquitetura padrão. A figura 3.1 representa essa arquitetura.

Este modelo de arquitetura baseia-se na extração, transformação e carga dos dados provenientes das fontes de dados para o *Data Warehouse*. Nesta abordagem o *Data Warehouse* mantém dados detalhados e históricos, e a partir desses dados é que os *Data Marts* são carregados, com os dados já sumarizados [**?**].

As principais vantagens dessa abordagem são:

- Todos os *Data Marts* originados a partir de um *Data Warehouse*, utilizam os dados e a arquitetura desse *Data Warehouse*, permitindo uma fácil manutenção e expansão.

- O *Data Warehouse* concentra todos os negócios da empresa, possibilitando extrair níveis menores de informações.
- O *Data Warehouse* irá manter um repositório centralizado dos metadados do sistema, facilitando as manutenções.
- Essa abordagem garante a existência de um único conjunto de aplicação para extração, limpeza e integração dos dados, além desses processos estarem centralizados.

As principais desvantagens são:

- O tempo de implementação desse ambiente de *Data Warehouse* é muito longo, podendo fazer com que o projeto perca apoio político e orçamentário.
- Existe uma alta taxa de risco, pois não há garantia de um retorno sobre o investimento que é feito.

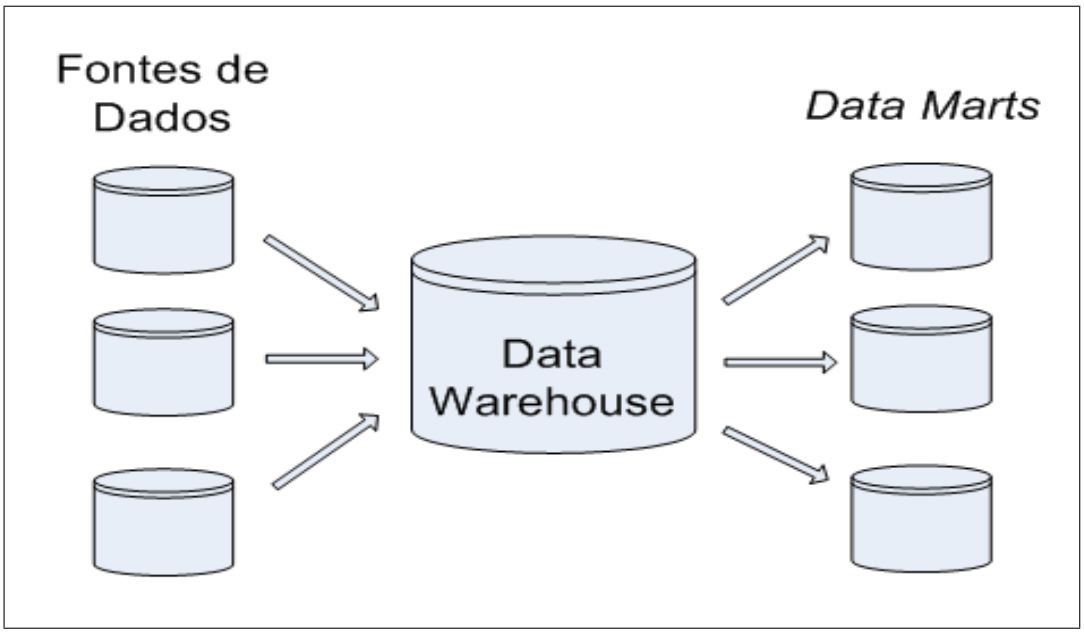

Figura 3.1: Arquitetura Top Down

3.4.3.2 Arquitetura *Bottom-Up*

A implantação da arquitetura *Top-Down* é difícil pois exige um alto orçamento, e os resultados são demorados. Devido a isso, a arquitetura *Bottom-Up* vem se tornando mais popular. A figura 3.2 mostra um exemplo de arquitetura *Bottom-Up*.

A idéia central por trás desta arquitetura é a construção de um *Data Warehouse* de uma maneira incremental, a partir do desenvolvimento de *Data Marts* independentes. Os processos de extração, transformação e carga de cada *Data Mart* são disjuntos. Um grande problema desta arquitetura é a falta de um gerenciador que garanta padrões únicos de metadados A falta de padronização (ausência de metadados compartilhados) faz com que a arquitetura *Bottom-Up* tenha um sucesso inicial, graças a sua rápida implementação e retorno. Contudo geralmente falha ao longo do tempo exatamente no ponto ao qual se propõe, que é construir um ambiente de *Data Warehouse*.

Outra desvantagem dessa arquitetura é o fato de que cada setor da organização desenvolve soluções individuais, não levando em consideração a arquitetura global, inviabilizando futuras integrações e dificultando a administração e coordenação das várias equipes que desenvolvem em paralelo seus *Data Marts*.

Novas abordagens surgiram como variações da arquitetura *Bottom-Up*. Elas visam otimizar o processo de desenvolvimento e garantir a consistência dos metadados e a facilidade de integração.

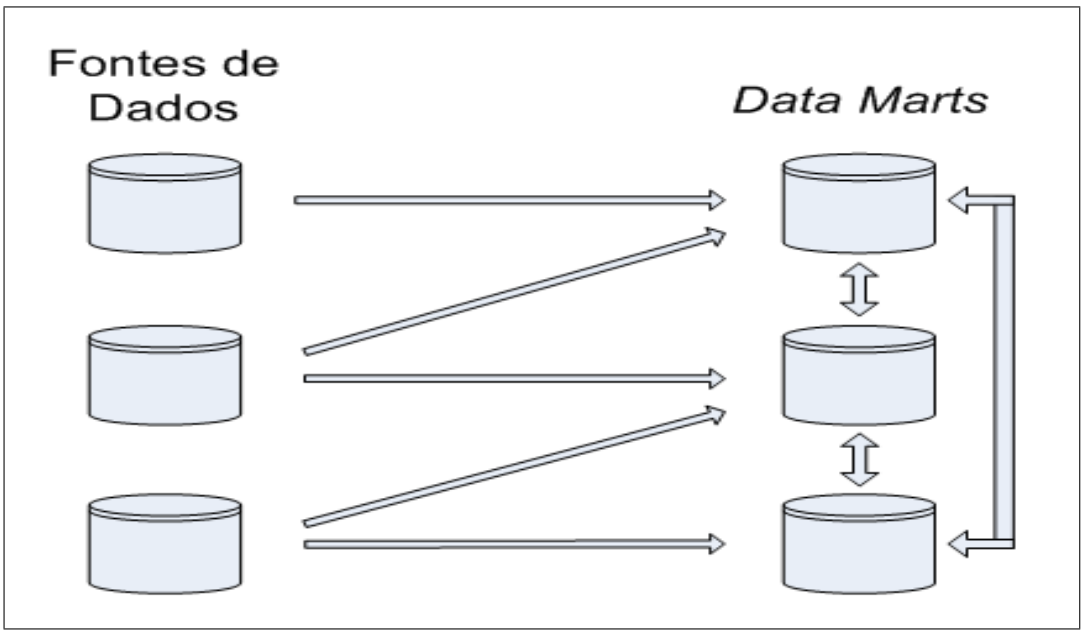

Figura 3.2: Arquitetura Bottom Up

3.5 Modelagem dos Dados

A compreensão dos dados é um dos maiores desafios na construção de um *Data Warehouse*. A construção de um bom modelo de dados é fundamental para o desenvolvimento do *Data Warehouse*, ajudando a compreender as regras de negócio e a organização dos dados para o melhor tempo de resposta [**?**].

Um modelo de dados é um conjunto de conceitos que podem auxiliar a descrever um conjunto de dados e as operações para sua manipulação [**?**].

A ausência, ou má utilização de um modelo de dados, implica em um crescimento desorganizado e alto custo para manutenções e integrações. O modelo proporciona para o usuário um melhor entendimento dos dados, facilitando a visualização da conseqüência de qualquer mudança dentro do ambiente [**?**].

A modelagem dos dados em um *Data Warehouse* é diferente da modelagem utilizada em sistemas operacionais (OLTP) [**?**]. Em um *Data Warehouse*, a modelagem utiliza alguma técnica que permita uma melhor análise dos dados, geralmente uma modelagem multidimensional, conhecida como modelo estrela, é utilizada. Por outro lado, nos sistemas operacionais, é utilizada uma modelagem com alto nível de normalização dos dados.

3.5.1 SGBD Multidimensional (SGBDM)

O SGBDM é um sistema gerenciador de banco de dados que utiliza uma estrutura multidimensional para o armazenamento de dados. Para garantir um melhor desempenho na análise das informações, geralmente SGBDM trabalham com uma estrutura de cubos de informações. Alguns bancos multidimensionais requerem uma completa recarga do banco quando um reestruturação ocorre [**?**].

A tabela 3.4 apresenta algumas diferenças entre SGBD relacional e um SGBD multidimensional, quando tratando com o modelo dimensional [**?**].

Tabela 3.4: Diferenças entre SGBDM e SGBDR

SGBDM	**SGBDR**
Não suporta uma quantidade muito grande de dados e existem restrições quanto ao número de dimensões.	Suporta um grande volume de dados, e um grande número de dimensões.
Tecnologia ainda esta crescendo.	Tecnologia comprovada.
Desempenho otimizado para processamento de apoio a decisão.	Desempenho um pouco inferior na maioria dos casos.
Estrutura de dados pode ser otimizada para um padrão de acesso conhecido.	Não pode ser otimizada exclusivamente para um padrão de acesso.
Não apresenta estrutura flexível para acessar dados por caminho não preparado.	Fácil acesso aos dados.

3.5.2 Visão Multidimensional

Representa a forma como os analistas de negócios e especialista analisam a informação de uma organização. Normalmente eles não trabalham apenas com uma informação específica, mas sim com um cruzamento entre elas.

3.5.3 Modelagem Multidimensional

A modelagem multidimensional representa como as informações de uma organização serão modeladas. O principal objetivo dessa modelagem é a elaboração de um projeto de banco de dados que permita uma visão multidimensional, assim como a visão do usuário. O modelo final deve ser facilmente entendido pelos desenvolvedores e pelos usuários finais.

Primeiramente deve ser feita uma modelagem lógica - geralmente utilizando o modelo

estrela - , independente do SGBD que venha a ser utilizado, permitindo que depois seja implementada em um SGBD relacional ou em um SGBD multidimensional.

A modelagem multidimensional facilita as operações realizadas pelas ferramentas de consultas ao *Data Warehouse*, como por exemplo:

- *Rollup*: (Figura 3.3) permite que o usuário diminua o nível de detalhamento, utilizando uma granularidade maior, aumentando o nível de agregação.

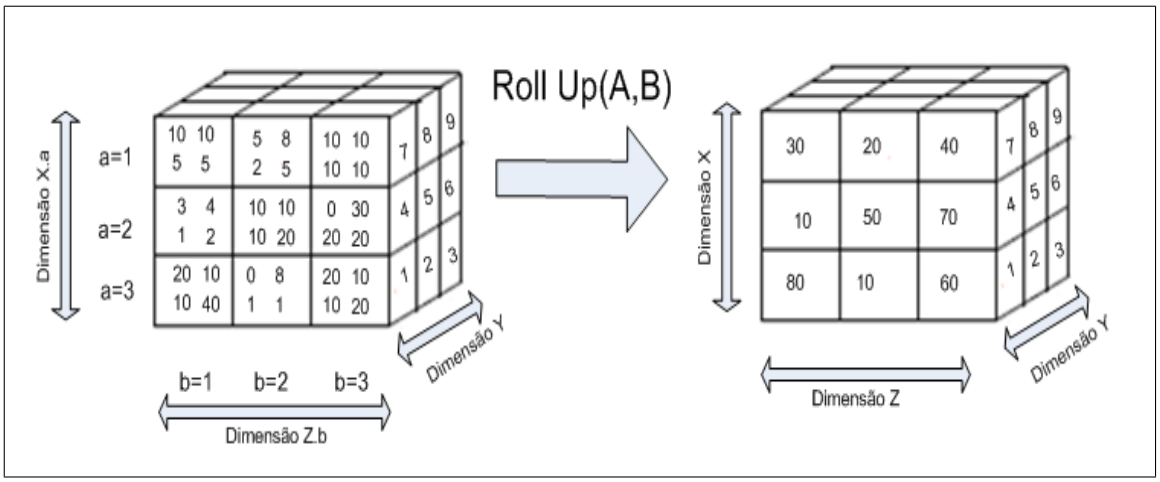

Figura 3.3: Roll Up

- *Drill-Down*: (Figura 3.4) é o processo inverso do *Rollup*, aumentando o detalhamento, utilizando uma granularidade menor e diminuindo o nível de agregação.

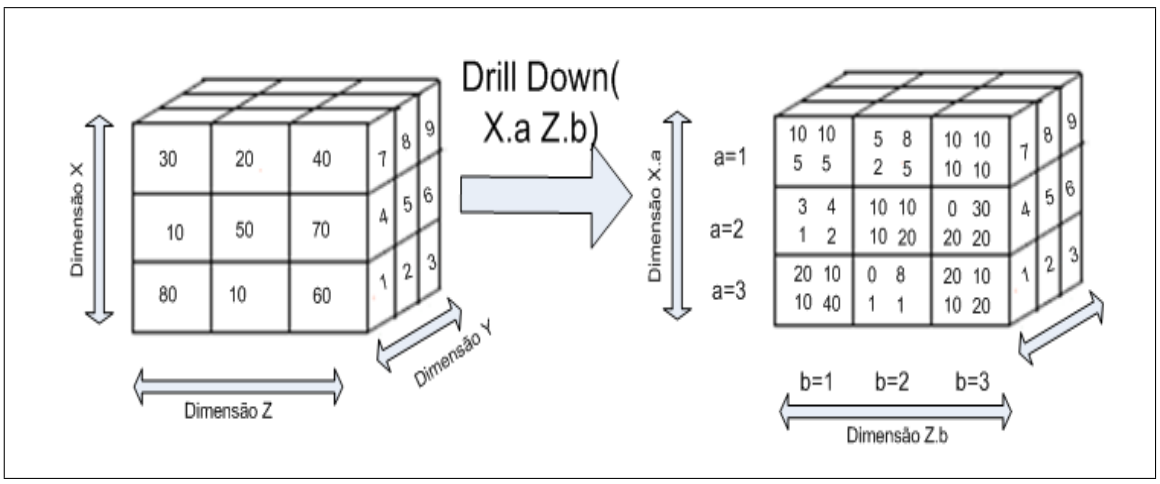

Figura 3.4: Drill Down

- *Slice-and-Dice*: (Figura 3.5) são operações que "permitem acessar um *Data Warehouse* por meio de qualquer uma de suas dimensões de forma igual" [**?**]. Através destas operações é possível navegar através dos dados do cubo ao longo de qualquer dimensão [**?**].

  Slice corta o cubo mantendo a mesma perspectiva de visualização dos dados, permitindo que o usuário fixe a apresentação em um determinado detalhe [**?**].

  Dice, ou rotação significa mudar a perspectiva de visão, girando o cubo e invertendo uma ou mais dimensões [**?**] [**?**].

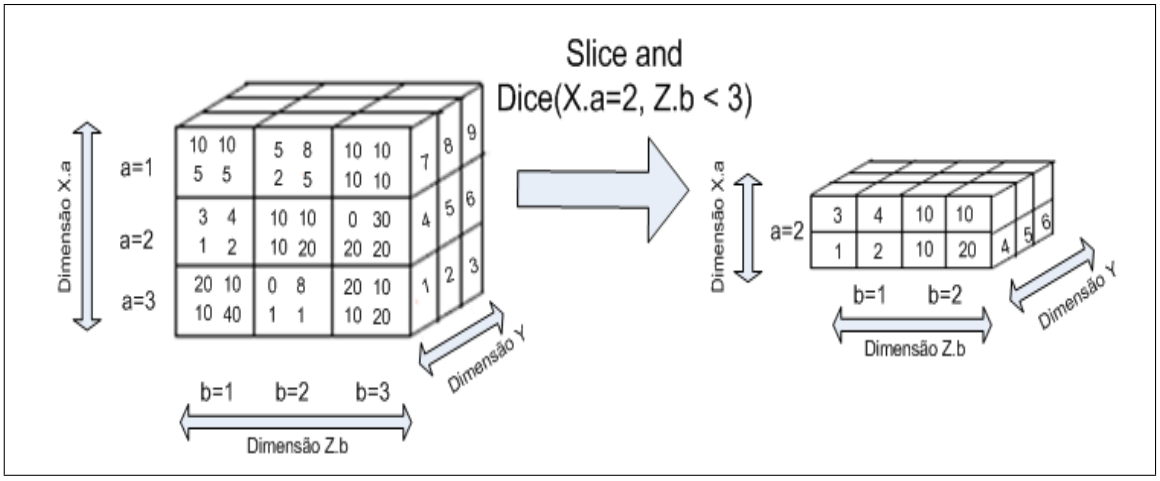

Figura 3.5: Slice and Dice

- *Pivot* (Figura 3.6) permite que o usuário altere a disposição das dimensões.

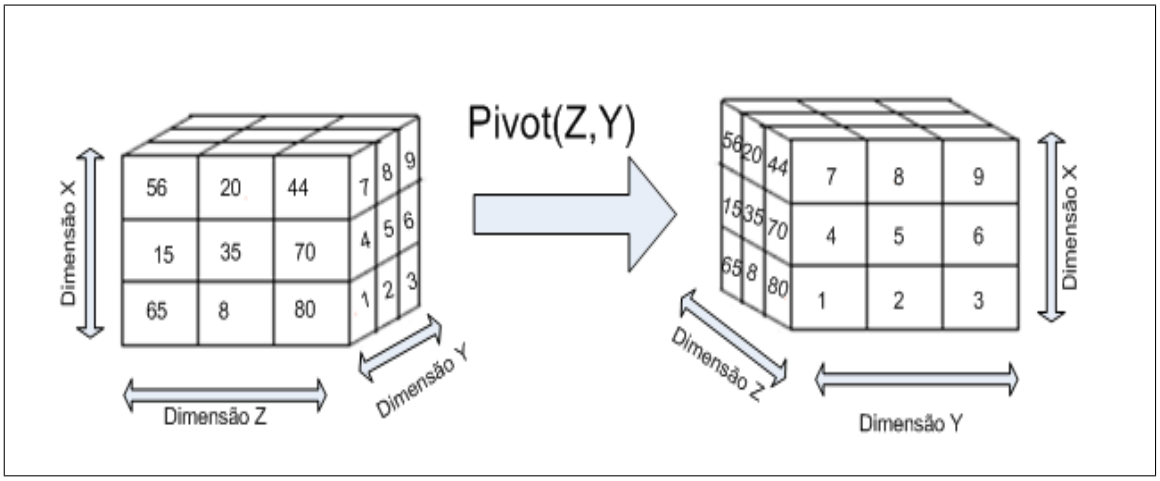

Figura 3.6: Pivot

3.5.4 Esquema Estrela

Esquema estrela é uma modelagem onde o objetivo é limitar o número de uniões entre tabelas, com junções bem definidas, reduzindo a complexidade dessas uniões. Em um esquema estrela os dados são ligeiramente normalizados, possibilitando obter ganhos de desempenho em relação às estruturas altamente normalizadas [**?**].

Um esquema estrela permite [**?**]:

- Facilidade de leitura e entendimento tanto por analistas como por usuários finais que não são familiarizados com estruturas de banco de dados.
- Um projeto de banco de dados que se pareça com a visão do usuário final.
- Criação de um banco de dados que permita consultas rápidas e eficientes.
- Navegação nos metadados pelos desenvolvedores e usuários finais.
- Utilização de várias ferramentas de *front-end*.

Este esquema apresenta alguns problemas quando existem muitas dimensões ou as dimensões são muito grandes. Não é aconselhado o uso desse esquema em sistemas que apresentam essas características [**?**].

Esta modelagem recebe o nome de esquema estrela devido à disposição do modelo. Ele é formado por uma tabela central, a tabela de fatos - que contém as transações ou valores que estão sendo analisados - que é relacionada com "n" tabelas de dimensões - que contém informações descritivas a respeito dessas transações ou valores. A figura 3.7 apresenta este esquema.

Tanto o modelo lógico, como o modelo físico do banco de dados podem ser representados pelo esquema estrela [**?**]. Uma representação simples de um modelo dimensional contém um esquema estrela com uma tabela de fatos relacionada com tabelas de dimensões. Um modelo dimensional pode ter uma ou mais tabelas de fatos.

3.5.4.1 Tabela de Fato

Cada registro de uma tabela de fato contém uma chave primária constituída de uma concatenação de chaves estrangeiras das tabelas de dimensão e as medidas identificadas exclusivamente por essa chave primária. Quando é projetada uma tabela de fatos, é necessário identificar quais medidas devem ser guardadas para serem analisadas. A tabela de fatos é uma tabela altamente normalizada, pois cada registro consiste em um número de atributos que podem ser designados a apenas uma chave primária. Ela não possui grupos de repetição, todos os atributos são dependentes da chave primária e nenhum dos atributos é dependente de atributos que não são chave, o que garante que esta tabela esteja na terceira forma normal. A desnormalização do esquema estrela ocorre nas tabelas de dimensão.

3.5.4.2 Tabela de Dimensão

Para criar as tabelas de dimensão, são unidas várias tabelas normalizadas Assim, cada registro descreve totalmente um elemento de dimensão. Isto melhora o desempenho da consulta do usuário, pois quando as consultas forem executadas, o trabalho necessário para unir as tabelas já foi realizado. Tabelas de dimensão tendem a ter muitas colunas, devido à desnormalização, e a ser curtas, se comparadas com as tabelas de fatos, pois podem haver milhares de registros na tabela de fatos que correspondem a apenas um registro na tabela de dimensão. Um campo de indicação sobre o estado geralmente está presente para auxiliar a armazenar o histórico da dimensão.

A qualidade do banco de dados é proporcional a dos atributos de dimensões, portanto deve ser dedicado tempo e atenção a sua descrição, ao seu preenchimento e a sua garantia de qualidade [**?**].

3.5.5 Esquema Floco de Neve

O esquema floco de neve é uma variação do esquema estrela, onde todas as tabelas de dimensão estão normalizadas na terceira forma normal, ou seja, são retirados das tabelas os campos que são funcionalmente dependentes de outros campos que não são chaves.

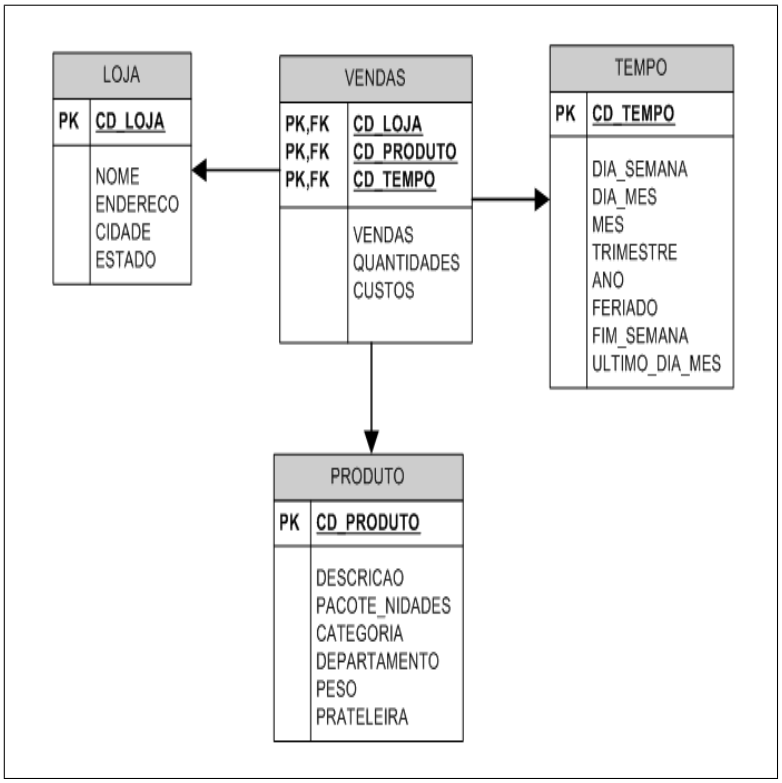

Figura 3.7: O esquema estrela (Fonte [**?**])

Recomenda-se utilizar o esquema floco de neve apenas quando a linha de dimensão ficar muito longa e começar a ser relevante do ponto de vista de armazenamento. A figura 3.8 representa um modelo floco de neve.

A normalização das tabelas de dimensões podem acarretar em alguns problemas, como o aumento da complexidade do modelo de dados, diminuindo a compreensão do modelo pelos usuários finais.

O uso desse esquema impossibilita o uso de esquemas de indexação mais eficientes, como a indexação *bitmap*.

O desempenho durante a fase de carga de dados no *Data Warehouse* será melhor com as tabelas de dimensões normalizadas, contudo o tempo de carga, na maioria dos casos não é uma operação crítica, pois geralmente é realizada durante a noite ou em momentos em que não estejam sendo executadas consultas.

O desempenho das consultas realizadas em um esquema floco de neve tende a diminuir devido ao aumento do número de junções em relação ao esquema estrela, e esse fator geralmente é decisivo na escolha de qual modelagem aplicar no *Data Warehouse*.

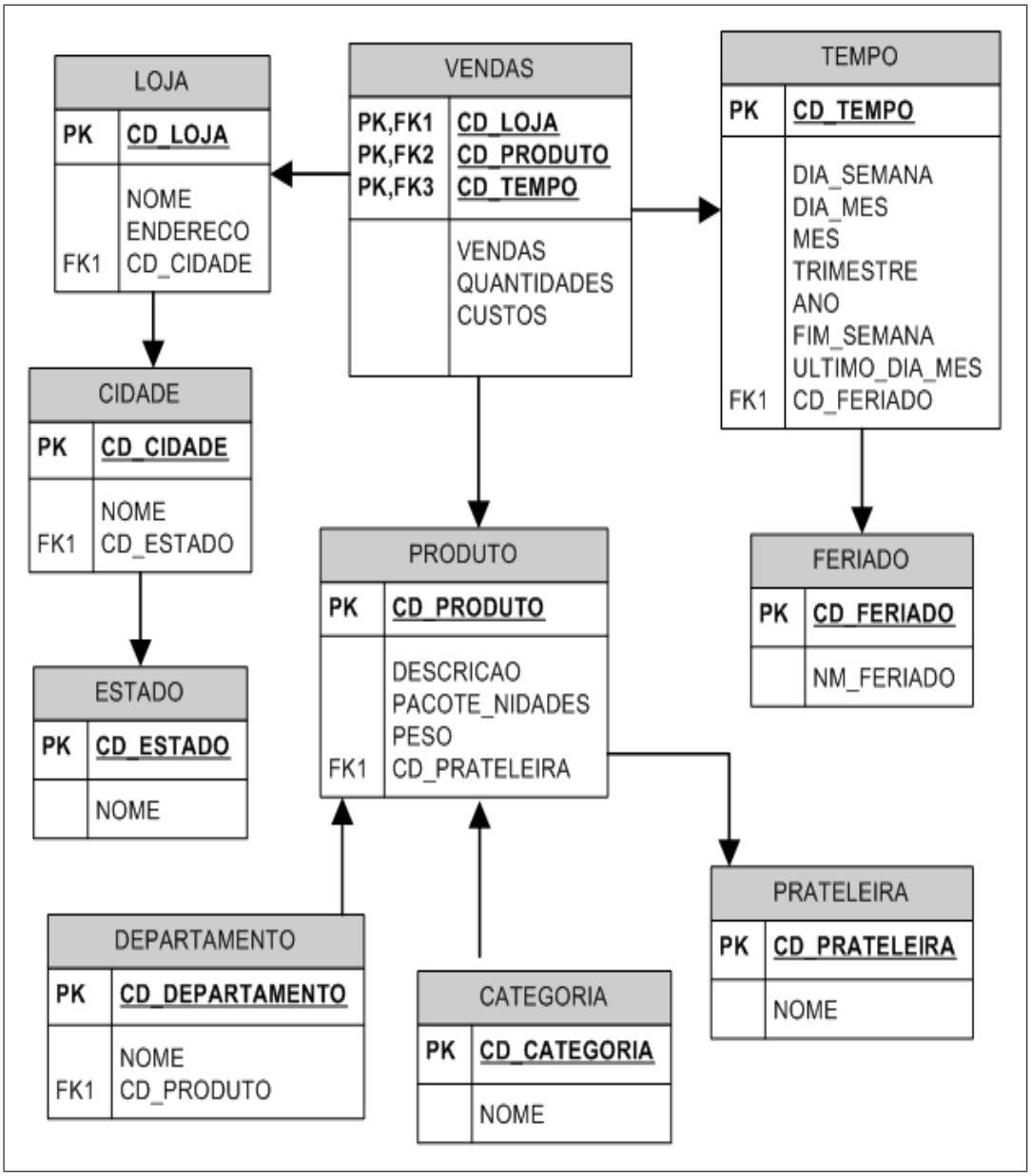

Figura 3.8: O esquema floco de neve

3.6 Projeto Físico do *Data Warehouse*

Após a criação do esquema dimensional do *Data Warehouse*, é recomendado que inicie-se a etapa de conversão desse esquema para o esquema físico. O esquema físico deve buscar ao máximo corresponder com o sistema lógico.

Um esquema físico bem implementado, pode ser o fator determinante entre o sucesso e o fracasso de um *Data Warehouse*. A implementação do *Data Warehouse* depende muito de componentes individuais do projeto, como o modelo lógico de dados, o volume de dados, o

SGBD utilizado, ferramentas de acesso e o bom uso de padrões, por isso se faz necessário que o projeto físico leve em consideração, desde o início, todos esses fatores, para que não ocorram falhas no decorrer do projeto.

Segundo Kimball [**?**] essas são as principais etapas do processo de transformação do esquema lógico para o esquema físico.

3.6.1 Definição de Padrões

Todo projeto de banco de dados - inclusive projeto de *Data Warehouse* - busca a consistência entre os dados que o compõem. Devem ser definidos dois importantes grupos de padrões: padrões relativos à nomenclatura de objetos de dados e os relativos aos nomes dos arquivos físicos e suas localizações.

O ideal é que os nomes do esquema físico reflitam os nomes do esquema lógico e sejam os mais descritivos possíveis. Ferramentas de modelagem podem auxiliar neste processo.

Uma boa padronização de nomenclatura permite, por exemplo, diferenciar as tabelas dos processos de ETL das tabelas da base de dados *Data Warehouse*.

3.6.2 Criação de Chaves

Nesta etapas as chaves estrangeiras e primárias especificadas no modelo lógico são criadas. A integridade referencial é importante para um *Data Warehouse*, pois este é um ambiente onde são executadas consultas que envolvem uma grande quantidade de dados e ocorrem inúmeras junções entre as tabelas. Geralmente a tabela de fatos possui uma chave composta pelas chaves das tabelas de dimensões a ela relacionada, melhorando o desempenho na recuperação de dados das tabelas de dimensões.

Também serão criadas novas chaves nas tabelas de dimensões do *Data Warehouse*. Essa nova identificação é decorrente dos seguintes fatores:

- necessidade de remapear a chave, evitando a dependência da chave original. O reuso de chaves é comum no ambiente operacional, devido à pequena periodicidade do armazenamento. Por outro lado, o *Data Warehouse* armazena os registros por um longo

período, exigindo uma nova chave para evitar a duplicidade e inconsistências para as consultas [**?**].

- criação de chaves genéricas que permitam a mudança na descrição dos itens sem provocar alterações das chaves dos mesmo. Uma solução comumente adotada no nível físico é acrescentar alguns dígitos ao final da chave original. Estes novos dígitos indicam a versão do item. As chaves genéricas permitem o rastreamento de modificações pela generalização da chave primária [**?**].

Os tipos de dados escolhidos para as colunas que são chaves devem ser os que trarão o maior beneficio no desempenho das consultas. Na maioria dos casos, o tipo inteiro (*integer*) é o mais indicado. Entretanto há casos onde o tipo de dados caracter (*char*) com tamanho fixo pode ser mais eficiente. Deve ser feita uma análise das características do SGBD para decidir qual será a melhor opção.

Também deve ser avaliada a substituição das chaves compostas de colunas do tipo data por chaves artificiais compostas de colunas do tipo inteiro. Isso permite que seja criada uma dimensão Tempo, onde além da data, podem ser armazenadas outras informações que podem ser importantes quando o usuário final for utilizar o *Data Warehouse* (por exemplo, podem ser armazenadas informações sobre alguma promoção que estivesse ocorrendo em uma determinada data, ou algum feriado). Além disso, como a maior parte das consultas em um *Data Warehouse* envolvem colunas do tipo data, a criação de uma dimensão tempo acaba gerando que as consultas sejam executadas mais eficientemente.

3.6.3 Mecanismos de Controle dos Processos de ETL

Para controlar e monitorar a execução dos processos de ETL no *Data Warehouse* é necessário que se crie uma tabela de controle para este fim. Assim, esta tabela iria armazenar o histórico de carga de dados no *Data Warehouse*, contendo informações como: data de início e data do fim, estado do processo, quantidade de registros atualizados, quantidade de registros rejeitados entre outros que possam ser úteis para a equipe que desenvolve a ferramenta de ETL.

3.6.4 Indexação

Em um *Data Warehouse*, o desempenho é uma preocupação constante. Um dos métodos mais eficientes em busca de uma melhor performance é o uso de índices. Uma indexação do *Data Warehouse* bem projetada irá minimizar o número de índices e melhorar a velocidade de acesso à informação.

É necessário ressaltar que adicionar índices que satisfaçam qualquer tipo de acesso a uma tabela não é uma boa escolha, pois quanto maior o número de índices, maior será o tempo da carga no *Data Warehouse*, pois os índices precisam ser atualizados. Um número maior de índices também representa um custo maior de administração do ambiente, pois a cada índice adicionado, o resultado será uma maior ocupação de espaço nas estruturas de armazenamento.

Algumas ações podem ajudar a construir um bom projeto de indexação do *Data Warehouse*:

- quando o *Data Warehouse* é implementado sobre um SGBD relacional, índices são criados nas colunas definidas como chaves primárias das tabelas. Índices adicionais serão criados assim que novos caminhos de acesso aos dados forem sendo utilizados.
- os índices criados nas chaves primárias se baseia na ordem nas quais as colunas foram criadas. Como na maioria das consultas em um *Data Warehouse* ocorre alguma filtro por data, é recomendado que as colunas de datas sejam as primeiras da chave primária. Isto também irá gerar uma melhor performance nos processos de cargas incrementais, que, em geral, são baseados em datas;
- use poucos ou nenhum índice em tabelas com uma alta taxa de inserções/exclusões/atualizações, para acessos de um grande volume de tuplas em tabelas em uma ordem específica e para tabelas de dimensões pequenas mas que sejam altamente acessadas. Nestes casos, o acesso seqüencial pode produzir melhores resultados;
- é inviável tentar descobrir qual a forma como os dados serão acessados pelo usuário, tentando criar índices para esses futuros acessos.

É importante que o projeto de indexação utilizado seja revisto regularmente, com base nos acessos aos dados realizado pelos usuários, fazendo com que índices não utilizados sejam apagados e que novos índices sejam criados [**?**].

Deve ser destacado que a existência de índices no *Data Warehouse* no momento da carga, pode degradar muito o desempenho desse processo. Segundo alguns fornecedores de SGBD's, em tabelas onde a carga de dados representa mais que 10% do total de registros da mesma, é mais eficiente excluir os índices antes da carga e recriá-los quando o processo for encerrado.

3.6.4.1 Tipos de Índice

A escolha do tipo de índice a ser utilizado é outro fator que pode fazer muita diferença no desempenho de um *Data Warehouse*. Abaixo alguns dos tipos índices mais utilizados [**?**]:

- Índice Arvore-B: é eficaz para colunas de alta cardinalidade. É o tipo de índice padrão para a maioria dos SGBD.
- Índice Bitmap: este índice é recomendado para colunas com baixíssima cardinalidade. Basicamente, constitui uma seqüência de bits para cada valor possível de uma determinada coluna. Este índice, quando usado em colunas com baixa cardinalidade, permite uma grande economia de espaço e pode ser processado com pouco ou quase nenhuma operação de E/S.
- Índice Hash: tem melhor desempenho em colunas com altíssima cardinalidade. Ao invés de construir um índice, ele utiliza uma fórmula matemática para calcular onde o registro que esta sendo buscado está localizado.
- Índice de Junção: este tipo de índice otimiza as consultas que envolvem diversas tabelas, como é o caso do *Data Warehouse* modelado com um esquema estrela. Ele é estruturado sobre condições de junções entre duas ou mais tabelas do *Data Warehouse*. Estas condições são colocadas em conjunto no índice, onde cada referência nada mais é do que um identificador de tuplas de ligações. Como é um índice composto por mais

de uma tabela, ele é complexo e mais difícil de se criar e manter que um índice comum, de uma única tabela.

- Outros tipos de Índices: alguns SGBD's utilizam estruturas proprietárias para indexação. Quando for possível utilizar alguma dessas estruturas, deve primeiramente ser feita uma análise sobre o desempenho de cada tipo de índice.

3.6.5 Dimensionamento do *Data Warehouse*

Para poder ser calculada a quantidade de investimento em *hardware* que será necessário, deve-se ser capaz de estimar os requisitos de armazenamento do *Data Warehouse*. É importante que as estimativas sejam o mais precisas possíveis, evitando problemas quando o *Data Warehouse* já estiver em funcionamento.

Alguns fatores devem ser considerados para uma estimativa mais precisa [**?**]:

- estimar o tamanho ocupado por cada tupla, considerando que colunas do tipo *VARCHAR* e outras colunas que aceitem o valor *null* irão ocupar um espaço menor que o máximo;
- estimar o tamanho de cada tabela com a carga inicial completa e como ela evoluirá com as cargas incrementais, levando em consideração, que usualmente os dados mais antigos são armazenados com uma granularidade maior;
- nos SGBD's tradicionais, o espaço reservado para os índices deve ser o mesmo espaço ocupado pelos dados das tabelas em que o índice é baseado;
- para operações de ordenação e agrupamento de dados de uma tabela, a área temporária deve ser dimensionada, para que tenha no mínimo o mesmo tamanho desta tabela. A área temporária também deve ter o dobro do espaço de um índice para poder realizar a ordenação eficientemente;
- reservar espaço para os metadados;
- as tabelas de agregados tendem a ocupar um espaço considerável, dependendo do grau de agregação e a profundidade das hierarquias dos dados. Geralmente pode-se considerar que ocuparão o mesmo espaço das tabelas em que se baseiam;

O tamanho ocupado pelas tabelas de dimensões é, na maioria dos casos, desprezível quando comparado ao tamanho da tabela de fatos. Os índices da tabela de fatos ocupam um espaço maior do que as tabelas de dimensões e seus índices.

3.6.6 Particionamento

Se o SGBD oferecer suporte ao particionamento de dados, essa estratégia irá melhorar o desempenho do *Data Warehouse*, quando for utilizada corretamente. Quando utilizada de forma incorreta, o particionamento acarretará uma degradação no desempenho. Caso as consultas seguidamente precisem acessar diversas partições, o desempenho da tabela particionada será pior que a performance de uma tabela não particionada. [**?**].

Usualmente, somente tabelas de fatos, seus índices e grandes tabelas de dimensões são candidatas ao particionamento [**?**]. Algumas vantagens do particionamento são:

- a consulta irá acessar somente as partições necessárias para retornar a informação. Quando o particionamento é realizado corretamente, isso resulta em uma melhor performance das consultas. Para obter melhores resultados com o particionamento, deve ser feito uma análise para descobrir qual é o melhor método de particionar cada tabela;
- gerenciar uma partição é um trabalho muito mais simples do que gerenciar toda uma tabela. Manutenções podem ser feitas apenas em algumas partições, enquanto as demais continuam acessíveis para o usuário. Carga de dados e *backups* são realizados mais facilmente, visto que tratam com uma quantidade menor de dados;

3.6.7 Acompanhamento

O *Data Warehouse* esta sempre em evolução. Isso significa que toda tarefa sempre deve ser revista e adequada a situação atual *Data Warehouse*.

Tempo de resposta, concorrência de usuários, utilização de índices, tempo de carga, tamanho do *Data Warehouse*, entre outros, são informações que o administrador do *Data Warehouse* deve sempre ter disponível, para que possa sempre que necessário, adequar o *Data*

Warehouse para uma melhor performance. Essas informações também são necessárias para que possa ser medida e estimada a taxa de crescimento do *Data Warehouse*, para que possa ser estimado o quanto de investimento será necessário para suportar esse crescimento [**?**].

3.6.8 Agregados

Agregados é um dos recursos, se não o recurso, de maior eficácia para otimização do desempenho do *Data Warehouse* [**?**]. Isto ocorre pois a maior parte das consultas realizadas no *Data Warehouse* analisam somente um subconjunto de dados, com um nível maior de granularidade, ou seja, em um nível onde os dados já se encontram agrupados de alguma determinada forma.

As tabelas de um *Data Warehouse* apresentam um volume muito alto de dados, fazendo com que as consultas sejam complexas de ser executadas devido ao nível de recursos necessários. Agregados são a pré-computação desses registros, reduzindo significativamente o número de registros resultantes e o nível de recursos necessário.

Um plano de agregação deve atingir as seguintes metas [**?**]:

- realizar ganhos substanciais de performance para o maior número possível de consultas;
- causar o menor impacto possível nos processos de carga de dados. A maior parte dos agregados precisam ser recriados ou atualizados a cada nova carga;
- balancear o aumento do desempenho com o aumento do espaço de armazenamento necessário, evitando que seja ocupado espaço de armazenamento com registros que não trarão nenhum, ou trarão pouco benefício no desempenho das consultas;

Além de um grande impacto positivo no desempenho, agregados produzem outro benefício; a possibilidade de garantir que eles englobam conceitos corretos. Por exemplo, a definição de um ”devedor”de uma empresa, para o departamento de cobranças pode ser a pessoa que comprou e não pagou no dia correto determinada dívida, já para o departamento financeiro pode ser a pessoa que ainda tem alguma dívida a ser paga mais adiante.

3.7 ETL

Um fator muito importante para o sucesso de um *Data Warehouse* é a qualidade da informação que ele contém. O processo de ETL (*Extract, Transform and Load*) deve extrair dados de várias fontes distintas, transformá-los em informações consistentes e de qualidade e carrega-los do *Data Warehouse*. Especialistas afirmam que o processo de ETL consome cerca de 80% do esforço do projeto de implementação do *Data Warehouse* e geralmente consome mais tempo que o previsto [**?**].

3.7.1 Extração

A extração é onde são identificadas as fontes de dados e é realizada a obtenção dos dados. Os dados que serão extraídos são selecionados seguindo um modelo de dados, que deve ser gerado durante a modelagem do *Data Warehouse*, ou seja, a qualidade dos dados é muito dependente da qualidade da modelagem. Geralmente as fontes de dados estão disponibilizados em diferentes plataformas e tecnologias, demandando modos de extração diferenciados, tornando ainda mais complexo esse processo.

A seguir algumas situações que devem ser respeitadas para um melhor processo de extração:

- Se dentro de uma organização houverem diferentes versões do mesmo dado, deve-se escolher o dado mais completo ou o mais correto.
- Os dados devem sempre ser os mais recentes possíveis.
- A carga dos dados deve ser feita após o fechamento do dia, mês, quinzena ou do período em que a organização trabalha, para que dados incompletos não sejam migrados para o *Data Warehouse*.

Durante a extração de dados para um *Data Warehouse* podem ser utilizadas diferentes técnicas.

3.7.1.1 Extração Total

Todos os dados são extraídos das fontes de dados. Os processos seguintes (transformação e carga) serão responsáveis por selecionar quais dados serão utilizados. São possíveis duas formas de implementação (Tabela 3.5):

- Carga incremental: todos os dados são obtidos dos sistemas fontes e somente uma parte é transferida para o *Data Warehouse*. É realizada uma comparação entre os dados obtidos durante o último processo de carga com os dados atuais, determinando quais sofreram mudanças, e consequentemente serão atualizados no *Data Warehouse*.
- Carga total: todos os dados obtidos das fontes de dados são transformados e carregados no *Data Warehouse*.

Tabela 3.5: Comparativo entre os métodos de extração total

Critério	**Carga Incremental**	**Carga Total**
Tempo de processamento	Mais lento, pois envolve comparações de registros	Mais rápido, pois não precisa comparar registros
Tempo de transferência para o *Data Warehouse*	O tráfego pode ser reduzido se a comparação ocorrer antes da transferência	A transferência completa de dados pela rede pode ser altamente custosa para grandes volumes de dados
Complexidade	Requer mecanismos que implementem comparações eficientes	Não necessita de comparações
Desempenho	Somente os dados que mudaram são carregados	É impactado com as deleções/ atualizações de registros redundantes

3.7.1.2 Extração Parcial

Neste método, somente os dados que foram inseridos, excluídos ou atualizados serão obtidos para serem carregados no *Data Warehouse*. Para identificar quais dados foram alterados, podem ser usadas *flags* de alterações ou *logs*.

A tabela 3.6 lista as principais diferenças entre os dois métodos.

Tabela 3.6: Comparativo entre extração total e parcial

Critério	**Extração Total**	**Extração Parcial**
Implementação	Não depende de como os dados são armazenados	Depende fortemente de como os dados são armazenado
Impacto nos sistemas fontes de dados durante a execução	Pode gerar uma degradação no desempenho dos ambientes fontes	A extração de somente os dados alterados faz com que a degradação do desempenho seja quase imperceptível
Redundância	Dados que já estão no *Data Warehouse* são extraídos novamente, sendo descartados somente por outros processos	Somente os dados alterados são extraídos
Tráfego na rede	Tráfego gerado será proporcional ao tamanho do *Data Warehouse*	A extração de somente os dados alterados gera pouco tráfego na rede
Desempenho do *Data Warehouse* durante a carga	Todos os dados serão recarregados (carga total) ou somente os dados alterados (carga incremental)	Somente os dados alterados serão inseridos
Uso	Melhores resultados em tabelas pequenas	Melhor opção para tabelas grandes

3.7.2 Transformação

Nesta etapa, os dados que foram carregados dos sistemas fontes serão transformados para se tornarem informações úteis para o usuário final.

Esta etapa é especialmente complexa pois o *Data Warehouse* trata com dados de diferentes sistemas que rodam em diferentes ambientes, mas que devem ser tratados de uma mesma forma no *Data Warehouse*.

É essencial a existência de metadados para auxiliar esse processo. As regras que os metadados informam auxiliam a garantir a consistência dos dados. Algumas etapas do processo de transformação são apresentadas a seguir.

3.7.2.1 Integração

Nesta etapa os dados provenientes de diferentes sistemas são combinados e é feito o mapeamento desses dados no *Data Warehouse*. Todo dado obtido dos sistemas fontes devem ser formatados para seguir o padrão definido para o *Data Warehouse*.

3.7.2.2 Limpeza

Essa é uma das principais etapas do processo de ETL, pois ela é responsável pela qualidade dos dados que estarão disponíveis no *Data Warehouse*. Se essa etapa for desconsiderada, o resultado poderá ser um *Data Warehouse* inconsistente, tornando-se sem utilidade [**?**].

Os dados provenientes das fontes de dados precisam ser validados, existe muita inconsistência nos dados dos sistemas fontes, que estão lá, simplesmente por não atrapalharem o funcionamento desses sistemas.

A implementação da ferramenta de ETL deve definir os níveis de qualidade dos dados que serão consideradas aceitáveis. A seguir, uma lista com as principais características de qualidade dos dados [**?**].

- A informação dentro de um *Data Warehouse* deve ser consistente.

- Não podem acontecer duplicidades dentro do *Data Warehouse*.
- A informação presente no *Data Warehouse* deve ser completa, representando todo o conjunto de informações relevantes. Se for analisado a venda de um determinado produto, não pode-se excluir nenhuma filial da análise (exceto se essa exclusão for intencional).
- As informações do *Data Warehouse* devem refletir exatamente as informações das fontes de dados.

Pode-se aplicar três tipos de limpeza nos dados. Sintática, estrutural e semântica:

1. Sintática: Na limpeza sintática é procurado corrigir erros de inconsistência que são gerados por erros de digitação, ortografia ou por serem utilizadas diferentes siglas para representar a mesma coisa. Por exemplo, ”UFSC”e ”U.F.S.C.”representam a mesma informação, porém, estão escritas de formas diferentes.
2. Estrutural: A limpeza estrutural diz respeito à inconsistência no modo em que a mesma informação é apresentada.
    - Registros que significam a mesma coisa podem ser representados de formas diferentes. Para representar o sexo masculino ou feminino, pode-se usar ”m”e ”f”, ”0”e ”1”, ”masculino”e ”feminino”.
    - Dados podem estar representados em unidades diferentes.Um sistema fonte pode fazer uma medição em centímetros, enquanto outro sistema pode fazer a mesma medição em polegadas.
3. Semântica : A limpeza semântica trata de inconsistências causadas por diferentes interpretações de um mesmo dado.
    - Registros podem ter diferentes significados embora tenham o mesmo nome.
    - A integridade referencial pode estar quebrada: podem existir dados em uma tabela e não existir seu correspondente em outra tabela. Um determinado aluno está matricula em alguma disciplina, mas ele na consta não lista de alunos.

- Como não mantém informações históricas, as fontes de dados podem reutilizar identificadores que não existem mais. Uma seqüência em um banco de dados que é fonte, pode ser cíclica e estar recriando o mesmo código para diversos alunos.
- Ausência de valor ou valor desconhecido (nulo) pode ser tratado de forma diferente nas fontes de dados, isso deve ser tratado para que não cause erros na análise no *Data Warehouse*.

3.7.2.3 Sumarização

Nesta etapa são aplicadas regras de negócios nos dados (exemplo, cálculos). Sumarização é a transformação de informações que estão em um nível de detalhamento maior, para um nível de detalhamento menor. Isto causa um impacto positivo no desempenho, evitando o tempo desperdiçado com os cálculos mais comuns.

3.7.3 Carga

Esta é a última etapa do processo de ETL. Nesta etapa são executas mais algumas etapas de preparação dos dados, a carga propriamente dita e algumas tarefas após a carga ser realizada, como o tratamento dos dados que foram rejeitados e a certificação da qualidade dos dados inseridos no *Data Warehouse*.

Em relação à carga de dados, algumas questões devem ser consideradas:

- Deve-se determinar a janela de tempo que será utilizada para realizar a carga. Como um *Data Warehouse* é um sistema de consulta, na maioria dos casos ele não precisa estar operacional o dia todo.
- Deve-se determinar a freqüência com que as cargas serão executadas. Isto irá depender da quantidade de mudanças e o crescimento dos dados nas fontes.
- o desempenho do processo de carga é diretamente proporcional ao desempenho dos recursos de hardware.

A etapa de carga pode ser decomposta da seguinte forma:

1. Assegurar que os dados utilizados no processo de extração e limpeza estão no mesmo padrão dos dados do *Data Warehouse*.

2. Carregar os dados no *Data Warehouse*. Durante a carga dos dados, as restrições de integridade referencial devem estar ativas, pois irão identificar algum problema de inconsistência que não foram tratado previamente.

3. Geralmente, haverá dados que serão rejeitados. Deve ser analisado o motivo da rejeição de cada dado, para que possa ser identificada e corrigida.

4. Quando a carga dos dados no *Data Warehouse* estiver completa, os índices devem então ser reconstruídos.

5. Deve-se checar a qualidade dos dados que foram carregados no *Data Warehouse*.

6. Finalmente deve-se avisar aos usuários que uma nova carga foi realizada e que os dados mais recentes estão acessíveis no *Data Warehouse*.

3.8 Conclusões

Este capítulo apresentou uma revisão teórica sobre *Data Warehouse* mostrando quais características esses sistemas devem possuir, demonstrando os conceitos e as etapas de construção.

Em seguida foram apresentadas as duas arquiteturas mais comuns, a *Top Down* e a arquitetura *Botton Up*, mostrando quais as vantagens e desvantagens de cada. Pode-se concluir que o sistema de *Data Warehouse* irá depender muito da arquitetura escolhida.

A modelagem de dados é um dos maiores desafios durante a construção do *Data Warehouse*. Foram apresentados dois esquemas de modelagem multidimensional, o estrela e o floco de neve.

Por último foram apresentados tópicos a respeito do projeto físico do *Data Warehouse* e das etapas de extração, transformação e carga da informação no *Data Warehouse*.

A tarefa de construir um *Data Warehouse* é complicada e deve ser realizada com um equipe competente, para garantir o sucesso do projeto.

Capítulo 4

O *Data Mart* de Indicadores do *Balanced Scorecard*

O objetivo do projeto foi criar um CUBO OLAP que auxilie a tomada de decisão em sistemas que implementam a metodologia de *Balanced Scorecard*.

4.1 Características do *Data Mart*

Foi construído um *Data Mart* seguindo a arquitetura *Bottom Up*. A fonte de dados é um sistema que implementa a metodologia *Balanced Scorecard*, que pode utilizar como fonte de dados três diferentes SGBD: Oracle, Postgresql ou Mysql. Foi construído um *Data Mart* em cada SGBD.

A modelagem dos dados foi feita utilizando-se o esquema estrela, permitindo uma melhor análise da informação contida no *Data Mart*.

A ferramenta de ETL foi desenvolvida em JAVA, dando suporte aos três SGBD citados acima, e aproveitando das vantagens da linguagem de programação JAVA.

O *front end* não foi desenvolvido, pois estava fora do escopo deste projeto.

4.2 Etapas da Construção do *Data Mart*

4.2.1 Planejamento

Inicialmente foi realizado um estudo teórico sobre *Data Warehouse*, *Data Mart* e *Balanced Scorecard*. A etapa seguinte foi realizar o levatamento dos requisitos e depois construir a modelagem de dados.

Após ser feita a modelagem de dados, foi escolhida a arquitetura utilizada no *Data Mart*. Então foi desenvolvida uma ferramenta para executar o processo de ETL.

4.2.2 Levantamento de Requisitos

Uma das tarefas mais difíceis é o levantamento de requisitos. Nesta etapa é preciso saber quais informações os usuários finais precisam ter a sua disposição, e em qual nível de detalhamento, possibilitando que o *Data Mart* auxilie na tomada de decisão.

4.2.2.1 Escolha do Assunto

A etapa de escolha do assunto foi definida baseada nas necessidades dos usuários finais. Foi decidido que os indicadores seriam o assunto principal do *Data Mart*.

4.2.2.2 Definição do Grão

O nível de detalhamento escolhido foi o mesmo nível de detalhamento existente no sistema de *Balanced Scorecard*. Os indicadores foram migrados para o *Data Mart* com seus valores ”padrão”, ”meta”, ”realizado”e outras informações que são necessárias para análise, como a fórmula, unidade de medida, indicador de melhoria, frequência de coleta e categoria. Para as outras informações, como a área, perspectiva e objetivo no qual o indicador pertence, foram migrados apenas os nomes. Não foi utilizada nenhuma sumarização dos dados.

4.2.2.3 Escolha das Dimensões

As dimensões escolhidas foram todas as estruturas que fazem parte da hierarquia do *Balanced Scorecard* (Área - Perspectiva - Objetivo - Indicador), exceto o tema estratégico.

Além destas, foram necessárias criar dimensões para armazenar as informações sobre usuários, para definir quais usuários podem acessar determinadas informações.

4.2.2.4 Escolha dos Fatos

A tabela de fatos foi construída com os valores dos indicadores. Além da informações dos valores e as vinculações com as tabelas de dimensões, a tabela de fato possuí uma data correspondente.

4.2.3 Modelagem Dimensional

A modelagem multidimensional foi desenvolvida com base nas dimensões, fatos, assunto e granularidade que foram descobertos durante o levantamento de requisitos.

Para modelar foi utilizado um esquema estrela, pois este oferece um desempenho melhor nas consultas ao CUBO.

4.2.4 ETL

O processo de ETL foi realizado através da implementação de uma ferramenta em JAVA, que extrai os dados de três SGBD diferentes (Oracle, Postgresql, Mysql) e pode armazenar a informação nos três SGBD. As rotinas de carga utilizarão o padrão extração total com carga total dos dados.

4.3 Desenvolvimento

4.3.1 Arquitetura

A arquitetura *Bottom Up* foi escolhida por atender melhor as necessidades. A fonte de dados é única e pode utilizar três tipos diferentes de SGBD. Outro motivo pela escolha da arquitetura *Bottom Up* é o fato de que as empresas que já possuem o sistema de *Balanced Scorecard* não precisarão fazer investimentos na compra de hardware novo, pois fisicamente a fonte de dados e o *Data Mart* podem estar localizados no mesmo servidor, embora estejam separadas numa visão lógica da arquitetura. (figura 4.1).

Toda informação relacionada aos metadados, estará na ferramenta de ETL, que será responsável pela carga de todos os *Data Marts*.

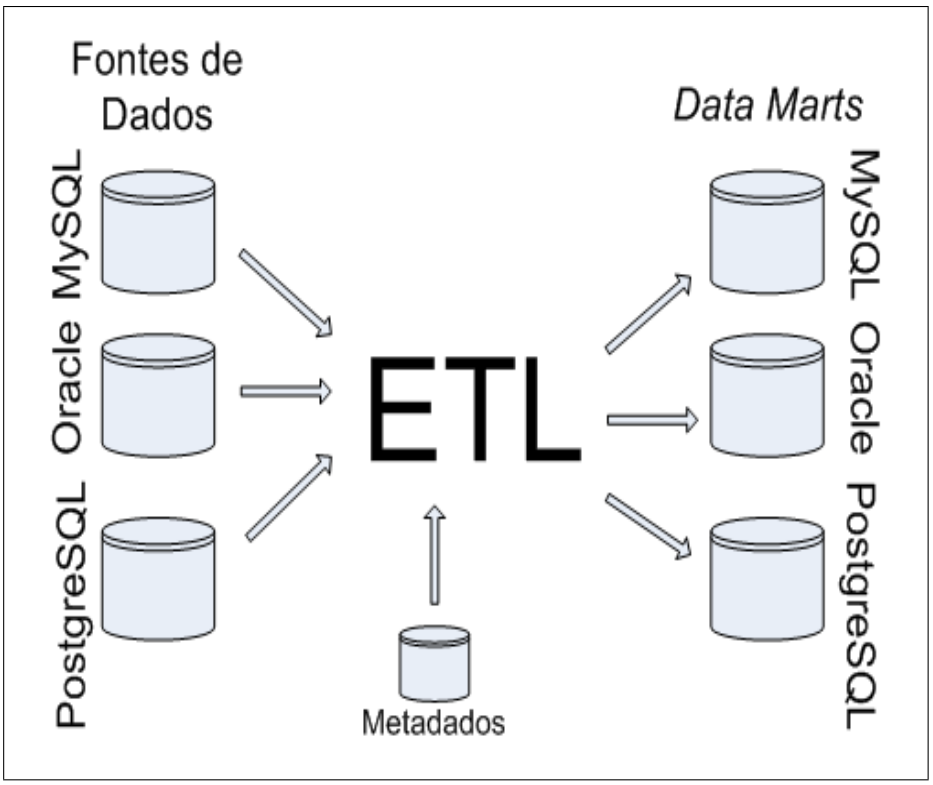

Figura 4.1: Arquitetura Escolhida

4.3.1.1 Modelagem

Antes de começar a modelagem do esquema estrela, durante o levantamento de requisitos foram definidos quais dados seriam importantes para serem carregados, e qual seria a

granularidade dos dados.

Após definido quais dados e com qual granularidade seriam migrados, foi construído um modelo estrela (figura 4.2) que suportasse o dados que seriam migrados.

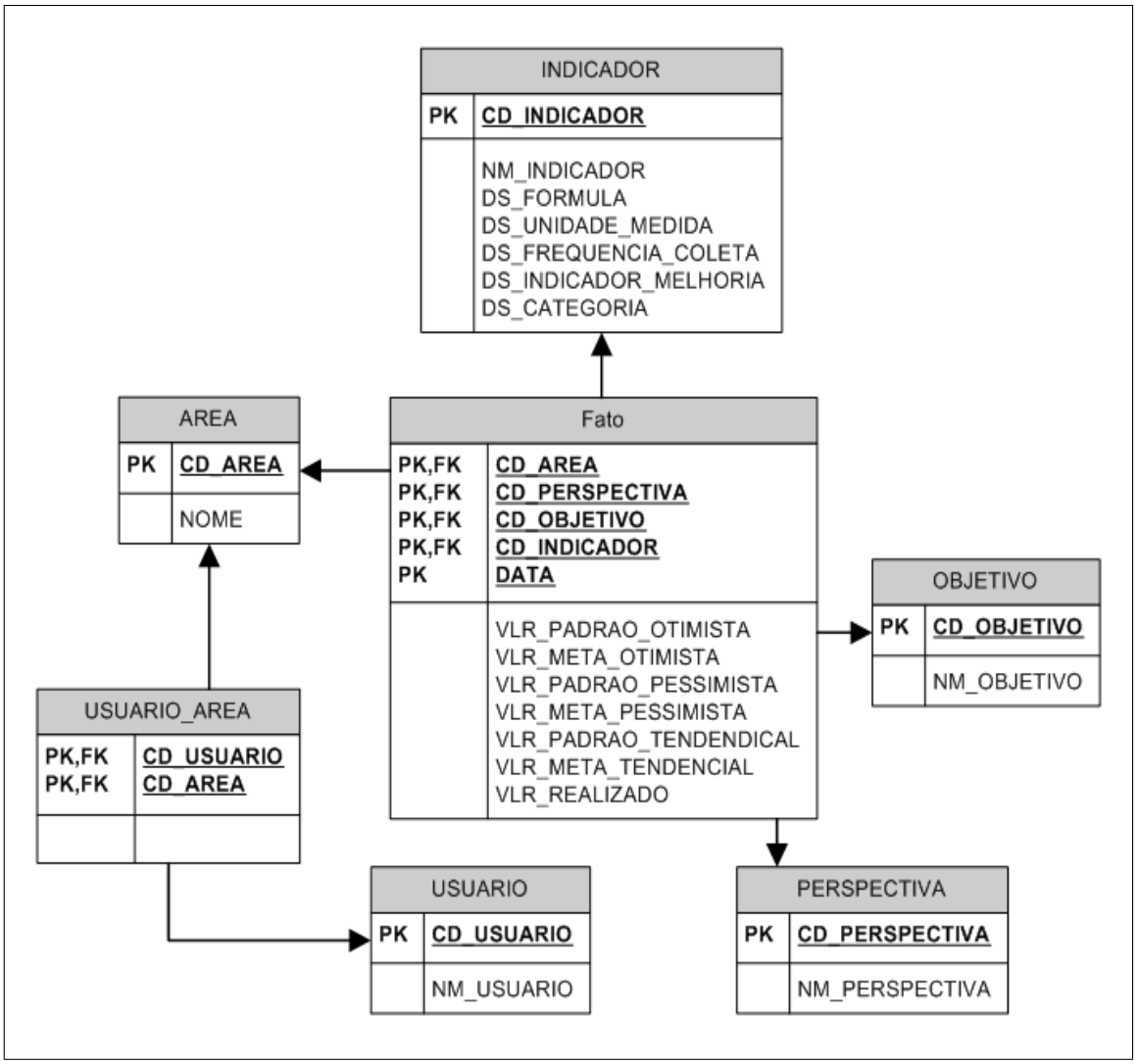

Figura 4.2: Esquema Estrela Final

4.3.2 ETL

A ferramenta de ETL foi desenvolvida em JAVA, utilizando-se três camadas para facilitar melhorias futuras do código. Para o desenvolvimento a metodologia TDD (*Test Driven Development*) foi utilizada.

Optou-se por utilizar o processo de ETL com extração total e carga total dos dados.

4.3.2.1 Camada do Modelo do ETL

Nesta camada, foram modeladas algumas classes que seriam importantes para o processo de ETL. As classes definidas são exibidas na figura 4.3.

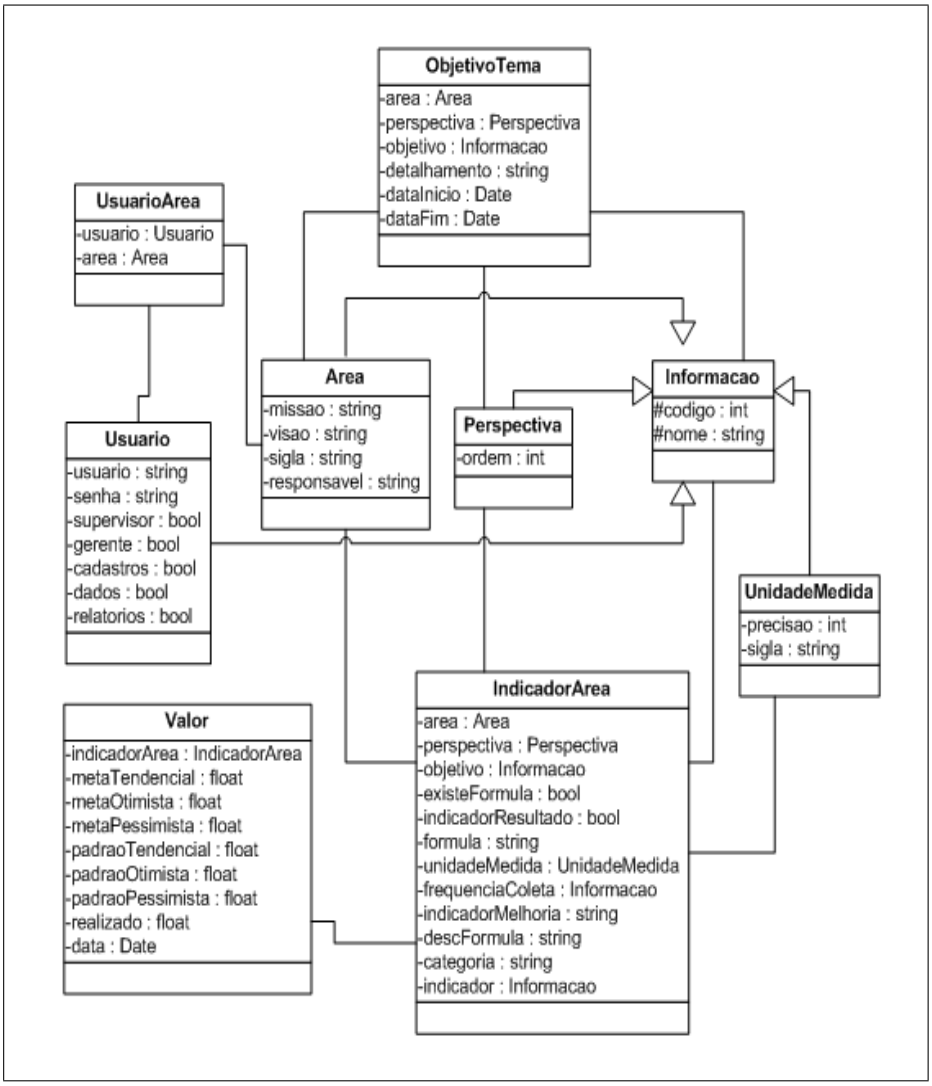

Figura 4.3: Diagrama de classes do modelo lógico do sistema

Para extrair as informações, e carregá-las novamente nas bases de dados, foi utilizado o padrão DAO (*Data Access Object*). Foram estabelecidas duas classes abstratas, uma para extração (DAOMigradorOrigem) e outra para carga (DAOMigradorDestino) de dados, e todas as classes que forem extrair ou carregar dados de alguma base devem implementar essas classes abstratas. Para auxiliar, foi criada uma classe que utiliza o padrão *Factory*, que retorna o objeto que será responsável por extrair o objeto que será responsável por carregar os dados.

A classes abstratas utilizadas no DAO estão representadas na figura 4.4.

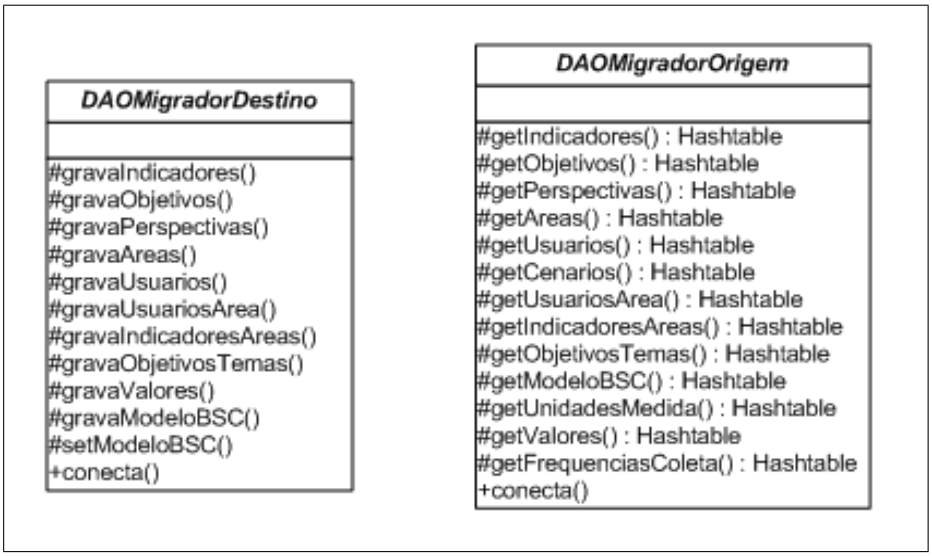

Figura 4.4: Classes abstratas do DAO

A classe responsável por extrair os dados no banco de dados executa uma extração total. É realiazada uma verificação de inconsistência dos dados durante essa extração através de junções de tabelas e somente os dados consistentes são extraídos para serem carregados no *Data Mart*.

4.3.2.2 Camada de Controle

Na arquitetura onde foi desenvolvido esse projeto, não houve a necessidade da camada de controle executar algumas etapas da transformação, como a integração, limpeza estrutural e limpeza semântica. Esta camada executa apenas uma limpeza sintática.

O controle também é responsável por iniciar a extração dos dados, e por criar um *log* do que está sendo executado, para que as informações a respeito do processo de ETL fiquem armazenadas para possíveis inspeções. A classe que armazena essas informações é a classe Auditoria. Nesse classe é armazenado o tempo inicial e o tempo final da extração, o tempo inicial e o tempo final da carga, assim como o número de registros atualizados.

4.3.2.3 Camada de Visualização

Essa camada é muito simples, pois após iniciar o processo, o usuário apenas é informado do término, com algumas informações como tempo total de execução e número de registros carregados.

Capítulo 5

Conclusão e Trabalhos Futuros

A construção de um CUBO OLAP é algo que exige um bom planejamento. Existem diversas variáveis que devem ser consideradas e analisadas. Primeiramente quais dados são importantes para a análise e que deverão ser migrados. Qual arquitetura será utilizada, uma arquitetura com resultado rápido, mas com o risco de falhar no futuro, ou uma arquitetura onde o resultado é lento, contudo com melhores resultados futuros. A forma com que o processo de ETL será executado também deve ser avaliado, considerado a janela de tempo disponível para realizar o processo, e a disponibilidade das fontes de dados, assim como a modelagem do esquema, e o tipo de SGBD que será utilizado.

Para a construção de um CUBO para análise de metas de sistemas baseados em *Balanced Scorecard*, deve-se também ter uma preocupação com quais dados e com qual nível de detalhamento os dados devem ser migrados pra o *Data Mart*. Neste projeto a decisão foi fazer a migração de todos os indicadores.

A modelagem utilizada foi uma modelagem multidimensional utilizando o esquema estrela. Essa técnica apresentou vantagens quando comparada ao esquema floco de neve na solução do problema de modelagem de dados do *Balanced Scorecard*.

O CUBO foi construído sob uma arquitetura *Bottom-Up*, pois essa se adapta melhor a situação de um CUBO para *Balanced Scorecard* onde existe apenas uma fonte de dados, e existe também apenas um *Data Mart* para consulta.

A ferramenta de ETL realiza uma extração total, contudo as inconsistências já são filtra-

das antes de serem extraídas. O método de carga utilizado também foi o método de carga total.

Como proposta para trabalhos futuros, recomenda-se a implementação das funcionalidades de extração parcial e carga parcial. Além do suporte a outros SGBD como fonte e destino dos dados. Outra recomendação é a criação de tabelas de agregados no *Data Mart*, melhorando a performance das ferramentas de consulta.

Capítulo 6

Anexo - Código Fonte

Listagem 6.1: Principal.java

```
package ufsc.ine;

import ufsc.ine.controle.Controlador;

public class Principal
{
    public static void main(String[] args)
    {
        new Controlador();
    }
}
```

Listagem 6.2: Controlador.java

```
/*
 * Binara Informática
 * Created on 21/02/2006
 */
package ufsc.ine.controle;

import ufsc.ine.dao.DAOFactory;
import ufsc.ine.dao.DAOMigradorDestino;
import ufsc.ine.dao.DAOMigradorOrigem;
import ufsc.ine.modelo.Auditoria;
```

```
import ufsc.ine.modelo.ModeloBSC;

public class Controlador {

    DAOMigradorOrigem daoOrigem = DAOFactory.getDAOMigradorOrigem();
    DAOMigradorDestino daoDestino = DAOFactory.getDAOMigradorDestino();

    Auditoria auditoria;

    public Controlador() {
        auditoria = new Auditoria();
        auditoria.iniciaLeitura();
        ModeloBSC modelo = daoOrigem.getModeloBSC();
        auditoria.encerraLeitura();
        daoDestino.setAuditoria(auditoria);
        daoDestino.gravaModeloBSC(modelo);
    }
}
```

Listagem 6.3: Conexao.java

```
/*
 * Binara Informática
 * Created on 01/02/2006
 */
package ufsc.ine.dao;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

import org.apache.log4j.Logger;

import ufsc.ine.log.GeradorLogger;

public class Conexao {
    Properties p;
```

```
    Connection conn;

20

    Logger log = GeradorLogger.getLogger(this.getClass().getName());

    public Conexao() {
        String file = "/ufsc/ine/resources/dao.oracle.properties";
25      try {
            p = new Properties();
            p.load(this.getClass().getResourceAsStream(file));
        } catch (IOException e) {
            log.error("Lendo arquivo de propriedades: " + file + "
               problema: "
30                  + e.toString());
        }

    }

35  protected void conectaOrigem() {
        try {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:thin:@"
                    + p.getProperty("oracle.origem.host") + ":"
40                  + p.getProperty("oracle.origem.port") + ":"
                    + p.getProperty("oracle.origem.sid"), p
                    .getProperty("oracle.origem.user"), p
                    .getProperty("oracle.origem.password"));
        } catch (SQLException e) {
45          String conect = "jdbc:oracle:thin:@"
                    + p.getProperty("oracle.origem.host") + ":"
                    + p.getProperty("oracle.origem.port") + ":"
                    + p.getProperty("oracle.origem.sid") + "(user:"
                    + p.getProperty("oracle.origem.user") + ")" + " [
                       excecao]:"
50                  + e.toString();
            log.error("Erro ao conectar no Banco de dados: " + conect);
            log.debug(e);
        }
    }
```

```
55
    protected void conectaDestino() {
        try {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            conn = DriverManager.getConnection("jdbc:oracle:thin:@"
60                  + p.getProperty("oracle.destino.host") + ":"
                    + p.getProperty("oracle.destino.port") + ":"
                    + p.getProperty("oracle.destino.sid"), p
                    .getProperty("oracle.destino.user"), p
                    .getProperty("oracle.destino.password"));
65      } catch (SQLException e) {
            String conect = "jdbc:oracle:thin:@"
                + p.getProperty("oracle.destino.host") + ":"
                + p.getProperty("oracle.destino.port") + ":"
                + p.getProperty("oracle.destino.sid") + "(user:"
70              + p.getProperty("oracle.destino.user") + ")" + " [excecao
                    ]:"
                + e.toString();
            log.error("Erro ao conectar no Banco de dados: " + conect);
            log.debug(e);
        }
75  }

    protected void desconecta() {
        try {
            conn.close();
80      } catch (SQLException e) {
            log.error("Erro ao desconectar do Banco de dados");
            log.debug(e);
        }
    }
85
}
```

Listagem 6.4: DAOFactory.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
```

```
 */
package ufsc.ine.dao;

import java.io.IOException;
import java.util.Properties;

public class DAOFactory {
    public static DAOMigradorOrigem getDAOMigradorOrigem() {
        Properties p = new Properties();
        try {
            p.load(DAOFactory.class
                    .getResourceAsStream("/ufsc/ine/resources/dao.
                        properties"));
        } catch (IOException e) {
            System.err.println("erro" + e);
        }
        DAOMigradorOrigem tmp = null;
        try {
            tmp = (DAOMigradorOrigem) Class.forName(
                    p.getProperty("dao.DAOMigradorOrigem")).newInstance()
                        ;
        } catch (IllegalAccessException e) {
            System.err.println("erro" + e);
        } catch (InstantiationException e) {
            System.err.println("erro" + e);
        } catch (ClassNotFoundException e) {
            System.err.println("erro:" + e);
        }
        return tmp;
    }

    public static DAOMigradorDestino getDAOMigradorDestino() {
        Properties p = new Properties();
        try {
            p.load(DAOFactory.class
                    .getResourceAsStream("/ufsc/ine/resources/dao.
                        properties"));
        } catch (IOException e) {
```

```
            System.err.println("erro" + e);
        }
        DAOMigradorDestino tmp = null;
        try {
            tmp = (DAOMigradorDestino) Class.forName(
                    p.getProperty("dao.DAOMigradorDestino")).newInstance
                        ();
        } catch (IllegalAccessException e) {
            System.err.println("erro" + e);
        } catch (InstantiationException e) {
            System.err.println("erro" + e);
        } catch (ClassNotFoundException e) {
            System.err.println("erro:" + e);
        }
        return tmp;
    }

}
```

Listagem 6.5: DAOMigradorDestino.java

```
/*
 * Binara Informática
 * Created on 22/02/2006
 */
package ufsc.ine.dao;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Enumeration;
import java.util.Hashtable;
import java.util.List;
import ufsc.ine.modelo.Area;
import ufsc.ine.modelo.Auditoria;
import ufsc.ine.modelo.Indicador;
import ufsc.ine.modelo.ModeloBSC;
import ufsc.ine.modelo.Objetivo;
```

```
import ufsc.ine.modelo.Perspectiva;
import ufsc.ine.modelo.Usuario;
import ufsc.ine.modelo.UsuarioArea;
import ufsc.ine.modelo.Valor;
public class DAOMigradorDestino extends Conexao
{
    ModeloBSC modelo;
    Auditoria auditoria;
    protected void gravaIndicadores()
    {
        Hashtable<Integer, Indicador> indicadores = modelo.getIndicadores
            ();
        Indicador indicador;
        String sql = "INSERT INTO INDICADOR (CD_INDICADOR, DS_INDICADOR,
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Indicador> e = indicadores.elements(); e.
            hasMoreElements();)
        {
            indicador = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, indicador.getCodigo());
                stmt.setString(2, indicador.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando indicador");
                log.debug(e1);
            }
        }
    }
    protected void gravaObjetivos()
    {
        Hashtable<Integer, Objetivo> objetivos = modelo.getObjetivos();
```

```
        Objetivo obj;
55      String sql = "INSERT INTO OBJETIVO (CD_OBJETIVO, DS_OBJETIVO,
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Objetivo> e = objetivos.elements(); e.
            hasMoreElements();)
        {
            obj = e.nextElement();
60          try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, obj.getCodigo());
                stmt.setString(2, obj.getNome());
65              stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
70              log.error("Erro gravando novo objetivo");
                log.debug(e1);
            }
        }
    }
75  protected void gravaPerspectivas()
    {
        Hashtable<Integer, Perspectiva> perspectivas = modelo.
            getPerspectivas();
        Perspectiva persp;
        String sqlInsert = "INSERT INTO PERSPECTIVA (CD_PERSPECTIVA,
            DS_PERSPECTIVA, CD_AUDITORIA) VALUES(?,?,?)";
80      PreparedStatement stmt = null;
        for (Enumeration<Perspectiva> e = perspectivas.elements(); e.
            hasMoreElements();)
        {
            persp = e.nextElement();
            try
85          {
                stmt = conn.prepareStatement(sqlInsert);
```

```
                stmt.setInt(1, persp.getCodigo());
                stmt.setString(2, persp.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova perspectiva");
                log.debug(e1);
            }
        }
    }
    protected void gravaAreas()
    {
        Hashtable<Integer, Area> areas = modelo.getAreas();
        Area area;
        String sqlInsert = "INSERT INTO AREA (CD_AREA, DS_AREA,
           CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Area> e = areas.elements(); e.hasMoreElements()
           ;)
        {
            area = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sqlInsert);
                stmt.setInt(1, area.getCodigo());
                stmt.setString(2, area.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova área");
                log.debug(e1);
            }
        }
    }
```

```
    protected void gravaUsuarios()
    {
125     Hashtable<Integer, Usuario> usrs = modelo.getUsuarios();
        Usuario usr;
        String sqlInsert = "INSERT INTO USUARIO (CD_USUARIO, DS_USUARIO,
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Usuario> e = usrs.elements(); e.hasMoreElements
            ();)
130     {
            usr = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sqlInsert);
135             stmt.setInt(1, usr.getCodigo());
                stmt.setString(2, usr.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
140         } catch (SQLException e1)
            {
                log.error("Erro gravando novo usuario");
                log.debug(e1);
            }
145     }
    }
    protected void gravaUsuariosArea()
    {
        Hashtable<Integer, UsuarioArea> usrs = modelo.getUsuariosAreas();
150     UsuarioArea usr;
        String sqlInsert = "INSERT INTO USUARIO_AREA (CD_USUARIO, CD_AREA
            , CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<UsuarioArea> e = usrs.elements(); e.
            hasMoreElements();)
        {
155         usr = e.nextElement();
            try
```

```
            {
                stmt = conn.prepareStatement(sqlInsert);
                stmt.setInt(1, usr.getUsuario().getCodigo());
                stmt.setInt(2, usr.getArea().getCodigo());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova vinculação USUARIO_AREA");
                log.debug(e1);
            }
        }
    }
    protected void gravaValores()
    {
        Hashtable<Integer, Valor> valores = modelo.getValores();
        Valor valor;
        String sql = "INSERT INTO FATO (CD_AREA, CD_PERSPECTIVA,
           CD_OBJETIVO, CD_INDICADOR, DATA, DS_CATEGORIA, DS_FORMULA,
           DS_FREQUENCIA_COLETA, DS_INDICADOR_MELHORIA, DS_UNIDADE_MEDIDA
           , VL_PADRAO_TENDENCIAL, VL_META_TENDENCIAL, VL_PADRAO_OTIMISTA
           , VL_META_OTIMISTA, VL_PADRAO_PESSIMISTA, VL_META_PESSIMISTA,
           VL_REALIZADO, CD_AUDITORIA) VALUES
           (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Valor> e = valores.elements(); e.hasMoreElements
           ();)
        {
            valor = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, valor.getIndicadorArea().getArea().
                   getCodigo());
                stmt.setInt(2, valor.getIndicadorArea().getPersp().
                   getCodigo());
                stmt.setInt(3, valor.getIndicadorArea().getObjetivo().
```

```
                    getCodigo());
                stmt.setInt(4, valor.getIndicadorArea().getIndicador().
                    getCodigo());
                stmt.setDate(5, new Date(valor.getData().getTime()));
                stmt.setString(6, valor.getIndicadorArea().getCategoria()
                    );
                stmt.setString(7, valor.getIndicadorArea().getFormula());
                stmt.setString(8, valor.getIndicadorArea().
                    getFrequenciaColeta().getNome());
                stmt.setString(9, valor.getIndicadorArea().
                    getIndicadorMelhoria());
                stmt.setString(10, valor.getIndicadorArea().
                    getUnidadeMedida().getNome());
                stmt.setFloat(11, valor.getPadraoTendencial());
                stmt.setFloat(12, valor.getMetaTendencial());
                stmt.setFloat(13, valor.getPadraoOtimista());
                stmt.setFloat(14, valor.getMetaOtimista());
                stmt.setFloat(15, valor.getPadraoPessimista());
                stmt.setFloat(16, valor.getMetaPessimista());
                stmt.setFloat(17, valor.getRealizado());
                stmt.setInt(18, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando Valores");
                log.debug(e1);
                e1.printStackTrace();
            }
        }
    }
    public void gravaModeloBSC(ModeloBSC modelo)
    {
        this.conecta();
        this.limpaBaseDestino();
        auditoria.setCodigo(this.getCodigoAuditoria());
        this.modelo = modelo;
        modelo.setAuditoria(auditoria);
```

```
        auditoria.iniciaEscrita();
        this.gravaAreas();
        this.gravaPerspectivas();
        this.gravaObjetivos();
        this.gravaIndicadores();
        this.gravaUsuarios();
        this.gravaUsuariosArea();
        this.gravaValores();
        auditoria.encerraEscrita();
        auditoria.setRegistrosLidos(modelo.getAreas().size() + modelo.
            getFrequenciasColeta().size() + modelo.getIndicadores().size()
             + modelo.getIndicadoresAreas().size() + modelo.getObjetivos()
            .size()
                + modelo.getObjetivosTemas().size() + modelo.
                    getPerspectivas().size() + modelo.getUnidadesMedida().
                    size() + modelo.getUsuarios().size() + modelo.
                    getUsuariosAreas().size() + modelo.getValores().size()
                    );
        this.desconecta();
    }
    public int getCodigoAuditoria()
    {
        String sqlInsert = "INSERT INTO AUDITORIA (CD_AUDITORIA) VALUES (
            SEQ_CD_AUDITORIA.NEXTVAL)";
        String sql = "SELECT SEQ_CD_AUDITORIA.CURRVAL AS CODIGO FROM DUAL
            ";
        int cod = -1;
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sqlInsert);
            stmt.executeUpdate();
            stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            while (rset.next())
                cod = rset.getInt("CODIGO");
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Codigo Auditoria");
```

```
            log.debug(e);
        }
        return cod;
    }
    public void setModeloBSC(ModeloBSC m)
    {
        this.modelo = m;
    }
    protected int getTotalLinhasTabela(String tabela)
    {
        int ret = -1;
        String sql = "SELECT COUNT(*) AS TOTAL FROM " + tabela;
        PreparedStatement stmt;
        try
        {
            stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            rset.next();
            ret = rset.getInt("TOTAL");
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Total de Linhas da tabela: " +
                tabela);
            log.debug(e);
        }
        return ret;
    }
    public void setAuditoria(Auditoria auditoria)
    {
        this.auditoria = auditoria;
    }
    public void conecta()
    {
        super.conectaDestino();
    }

    protected void limpaBaseDestino()
    {
```

```
        List<String> sqls = new ArrayList<String>();
        sqls.add("DELETE FROM FATO");
        sqls.add("DELETE FROM BR_INDICADOR");
        sqls.add("DELETE FROM INDICADOR");
        sqls.add("DELETE FROM OBJETIVO");
        sqls.add("DELETE FROM USUARIO_AREA");
        sqls.add("DELETE FROM USUARIO");
        sqls.add("DELETE FROM AREA");
        sqls.add("DELETE FROM CENARIO");
        sqls.add("DELETE FROM PERSPECTIVA");
        for (String sql : sqls)
        {
            try
            {
                conn.prepareStatement(sql).execute();
            } catch (SQLException e)
            {
                log.error("Apagando dados da tabela de fato");
                log.debug(e);
            }
        }
    }
}
```

Listagem 6.6: DAOFactory.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.dao;

import java.io.IOException;
import java.util.Properties;

public class DAOFactory {
    public static DAOMigradorOrigem getDAOMigradorOrigem() {
        Properties p = new Properties();
        try {
```

```
            p.load(DAOFactory.class
                    .getResourceAsStream("/ufsc/ine/resources/dao.
                        properties"));
        } catch (IOException e) {
            System.err.println("erro" + e);
        }
        DAOMigradorOrigem tmp = null;
        try {
            tmp = (DAOMigradorOrigem) Class.forName(
                    p.getProperty("dao.DAOMigradorOrigem")).newInstance()
                        ;
        } catch (IllegalAccessException e) {
            System.err.println("erro" + e);
        } catch (InstantiationException e) {
            System.err.println("erro" + e);
        } catch (ClassNotFoundException e) {
            System.err.println("erro:" + e);
        }
        return tmp;
    }

    public static DAOMigradorDestino getDAOMigradorDestino() {
        Properties p = new Properties();
        try {
            p.load(DAOFactory.class
                    .getResourceAsStream("/ufsc/ine/resources/dao.
                        properties"));
        } catch (IOException e) {
            System.err.println("erro" + e);
        }
        DAOMigradorDestino tmp = null;
        try {
            tmp = (DAOMigradorDestino) Class.forName(
                    p.getProperty("dao.DAOMigradorDestino")).newInstance
                        ();
        } catch (IllegalAccessException e) {
            System.err.println("erro" + e);
        } catch (InstantiationException e) {
```

```
            System.err.println("erro" + e);
        } catch (ClassNotFoundException e) {
            System.err.println("erro:" + e);
        }
        return tmp;
    }

}
```

Listagem 6.7: DAOMigradorDestinoOracle.java

```
/*
 * Binara Informática
 * Created on 21/02/2006
 */
package ufsc.ine.dao;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Enumeration;
import java.util.Hashtable;
import ufsc.ine.modelo.Area;
import ufsc.ine.modelo.Auditoria;
import ufsc.ine.modelo.Indicador;
import ufsc.ine.modelo.ModeloBSC;
import ufsc.ine.modelo.Objetivo;
import ufsc.ine.modelo.Perspectiva;
import ufsc.ine.modelo.Usuario;
import ufsc.ine.modelo.UsuarioArea;
import ufsc.ine.modelo.Valor;
public class DAOMigradorDestinoOracle extends DAOMigradorDestino
{
    protected void gravaIndicadores()
    {
        Hashtable<Integer, Indicador> indicadores = modelo.getIndicadores
            ();
        Indicador indicador;
        String sql = "INSERT INTO INDICADOR (CD_INDICADOR, DS_INDICADOR,
```

```
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Indicador> e = indicadores.elements(); e.
            hasMoreElements();)
        {
            indicador = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, indicador.getCodigo());
                stmt.setString(2, indicador.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando indicador");
                log.debug(e1);
            }
        }
    }
    protected void gravaObjetivos()
    {
        Hashtable<Integer, Objetivo> objetivos = modelo.getObjetivos();
        Objetivo obj;
        String sql = "INSERT INTO OBJETIVO (CD_OBJETIVO, DS_OBJETIVO,
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Objetivo> e = objetivos.elements(); e.
            hasMoreElements();)
        {
            obj = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, obj.getCodigo());
                stmt.setString(2, obj.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
```

```
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando novo objetivo");
                log.debug(e1);
            }
        }
    }
    protected void gravaPerspectivas()
    {
        Hashtable<Integer, Perspectiva> perspectivas = modelo.
            getPerspectivas();
        Perspectiva persp;
        String sqlInsert = "INSERT INTO PERSPECTIVA (CD_PERSPECTIVA,
            DS_PERSPECTIVA, CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Perspectiva> e = perspectivas.elements(); e.
            hasMoreElements();)
        {
            persp = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sqlInsert);
                stmt.setInt(1, persp.getCodigo());
                stmt.setString(2, persp.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova perspectiva");
                log.debug(e1);
            }
        }
    }
    protected void gravaAreas()
    {
```

```
        Hashtable<Integer, Area> areas = modelo.getAreas();
        Area area;
        String sqlInsert = "INSERT INTO AREA (CD_AREA, DS_AREA,
            CD_AUDITORIA) VALUES(?,?,?)";
100     PreparedStatement stmt = null;
        for (Enumeration<Area> e = areas.elements(); e.hasMoreElements()
            ;)
        {
            area = e.nextElement();
            try
105         {
                stmt = conn.prepareStatement(sqlInsert);
                stmt.setInt(1, area.getCodigo());
                stmt.setString(2, area.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
110             stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova área");
115             log.debug(e1);
            }
        }
    }
    protected void gravaUsuarios()
120 {
        Hashtable<Integer, Usuario> usrs = modelo.getUsuarios();
        Usuario usr;
        String sqlInsert = "INSERT INTO USUARIO (CD_USUARIO, DS_USUARIO,
            CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
125     for (Enumeration<Usuario> e = usrs.elements(); e.hasMoreElements
            ();)
        {
            usr = e.nextElement();
            try
            {
130             stmt = conn.prepareStatement(sqlInsert);
```

```
                stmt.setInt(1, usr.getCodigo());
                stmt.setString(2, usr.getNome());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando novo usuario");
                log.debug(e1);
            }
        }
    }
    protected void gravaUsuariosArea()
    {
        Hashtable<Integer, UsuarioArea> usrs = modelo.getUsuariosAreas();
        UsuarioArea usr;
        String sqlInsert = "INSERT INTO USUARIO_AREA (CD_USUARIO, CD_AREA
            , CD_AUDITORIA) VALUES(?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<UsuarioArea> e = usrs.elements(); e.
            hasMoreElements();)
        {
            usr = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sqlInsert);
                stmt.setInt(1, usr.getUsuario().getCodigo());
                stmt.setInt(2, usr.getArea().getCodigo());
                stmt.setInt(3, modelo.getAuditoria().getCodigo());
                stmt.execute();
                stmt.close();
            } catch (SQLException e1)
            {
                log.error("Erro gravando nova vinculação USUARIO_AREA");
                log.debug(e1);
            }
        }
    }
```

```
    protected void gravaValores()
    {
        Hashtable<Integer, Valor> valores = modelo.getValores();
        Valor valor;
        String sql = "INSERT INTO FATO (CD_AREA, CD_PERSPECTIVA,
            CD_OBJETIVO, CD_INDICADOR, DATA, DS_CATEGORIA, DS_FORMULA,
            DS_FREQUENCIA_COLETA, DS_INDICADOR_MELHORIA, DS_UNIDADE_MEDIDA
            , VL_PADRAO_TENDENCIAL, VL_META_TENDENCIAL, VL_PADRAO_OTIMISTA
            , VL_META_OTIMISTA, VL_PADRAO_PESSIMISTA, VL_META_PESSIMISTA,
            VL_REALIZADO, CD_AUDITORIA) VALUES
            (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
        PreparedStatement stmt = null;
        for (Enumeration<Valor> e = valores.elements(); e.hasMoreElements
            ();)
        {
            valor = e.nextElement();
            try
            {
                stmt = conn.prepareStatement(sql);
                stmt.setInt(1, valor.getIndicadorArea().getArea().
                    getCodigo());
                stmt.setInt(2, valor.getIndicadorArea().getPersp().
                    getCodigo());
                stmt.setInt(3, valor.getIndicadorArea().getObjetivo().
                    getCodigo());
                stmt.setInt(4, valor.getIndicadorArea().getIndicador().
                    getCodigo());
                stmt.setDate(5, new Date(valor.getData().getTime()));
                stmt.setString(6, valor.getIndicadorArea().getCategoria()
                    );
                stmt.setString(7, valor.getIndicadorArea().getFormula());
                stmt.setString(8, valor.getIndicadorArea().
                    getFrequenciaColeta().getNome());
                stmt.setString(9, valor.getIndicadorArea().
                    getIndicadorMelhoria());
                stmt.setString(10, valor.getIndicadorArea().
                    getUnidadeMedida().getNome());
                stmt.setFloat(11, valor.getPadraoTendencial());
```

```
190                 stmt.setFloat(12, valor.getMetaTendencial());
                    stmt.setFloat(13, valor.getPadraoOtimista());
                    stmt.setFloat(14, valor.getMetaOtimista());
                    stmt.setFloat(15, valor.getPadraoPessimista());
                    stmt.setFloat(16, valor.getMetaPessimista());
195                 stmt.setFloat(17, valor.getRealizado());
                    stmt.setInt(18, modelo.getAuditoria().getCodigo());
                    stmt.execute();
                    stmt.close();
                } catch (SQLException e1)
200             {
                    log.error("Erro gravando Valores");
                    log.debug(e1);
                    e1.printStackTrace();
                }
205         }
        }
        public void gravaModeloBSC(ModeloBSC modelo)
        {
            this.conecta();
210         this.limpaBaseDestino();
            auditoria.setCodigo(this.getCodigoAuditoria());
            this.modelo = modelo;
            modelo.setAuditoria(auditoria);
            auditoria.iniciaEscrita();
215         this.gravaAreas();
            this.gravaPerspectivas();
            this.gravaObjetivos();
            this.gravaIndicadores();
            this.gravaUsuarios();
220         this.gravaUsuariosArea();
            this.gravaValores();
            auditoria.encerraEscrita();
            auditoria.setRegistrosLidos(modelo.getAreas().size() + modelo.
                getCenarios().size() + modelo.getFrequenciasColeta().size() +
                modelo.getIndicadores().size() + modelo.getIndicadoresAreas().
                size() + modelo.getObjetivos().size()
                        + modelo.getObjetivosTemas().size() + modelo.
```

```
                    getPerspectivas().size() + modelo.getUnidadesMedida().
                    size() + modelo.getUsuarios().size() + modelo.
                    getUsuariosAreas().size() + modelo.getValores().size()
                    );
225         this.desconecta();
        }
        public int getCodigoAuditoria()
        {
            String sqlInsert = "INSERT INTO AUDITORIA (CD_AUDITORIA) VALUES (
                SEQ_CD_AUDITORIA.NEXTVAL)";
230         String sql = "SELECT SEQ_CD_AUDITORIA.CURRVAL AS CODIGO FROM DUAL
                ";
            int cod = -1;
            try
            {
                PreparedStatement stmt = conn.prepareStatement(sqlInsert);
235             stmt.executeUpdate();
                stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                while (rset.next())
                    cod = rset.getInt("CODIGO");
240         } catch (SQLException e)
            {
                log.error("Erro ao buscar Codigo Auditoria");
                log.debug(e);
            }
245         return cod;
        }
        public void setModeloBSC(ModeloBSC m)
        {
            this.modelo = m;
250     }
        protected int getTotalLinhasTabela(String tabela)
        {
            int ret = -1;
            String sql = "SELECT COUNT(*) AS TOTAL FROM " + tabela;
255         PreparedStatement stmt;
            try
```

```
        {
            stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            rset.next();
            ret = rset.getInt("TOTAL");
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Total de Linhas da tabela: " +
                tabela);
            log.debug(e);
        }
        return ret;
    }
    public void setAuditoria(Auditoria auditoria)
    {
        this.auditoria = auditoria;
    }
}
```

Listagem 6.8: DAOMigradorOrigem.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.dao;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Hashtable;
import org.apache.log4j.Logger;
import ufsc.ine.log.GeradorLogger;
import ufsc.ine.modelo.Area;
import ufsc.ine.modelo.FlagCubo;
import ufsc.ine.modelo.FrequenciaColeta;
import ufsc.ine.modelo.Indicador;
import ufsc.ine.modelo.IndicadorArea;
import ufsc.ine.modelo.ModeloBSC;
import ufsc.ine.modelo.Objetivo;
```

```
import ufsc.ine.modelo.ObjetivoTema;
import ufsc.ine.modelo.Perspectiva;
import ufsc.ine.modelo.UnidadeMedida;
import ufsc.ine.modelo.Usuario;
import ufsc.ine.modelo.UsuarioArea;
import ufsc.ine.modelo.Valor;
import ufsc.ine.util.GeradorChave;
public abstract class DAOMigradorOrigem extends Conexao
{
    Logger log = GeradorLogger.getLogger(this.getClass().getName());
    ModeloBSC modelo;
    GeradorChave gerador;
    protected Hashtable<Integer, Indicador> getIndicadores()
    {
        String sql = "SELECT CODIGOINDICADOR, NOME, FL_CUBO FROM
            INDICADORES";
        Hashtable<Integer, Indicador> indicadores = new Hashtable<Integer
            , Indicador>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            Indicador ind;
            while (rset.next())
            {
                ind = new Indicador(rset.getInt("CODIGOINDICADOR"), rset.
                    getString("NOME"), rset.getInt("FL_CUBO"));
                indicadores.put(new Integer(ind.getCodigo()), ind);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Indicadores");
            log.debug(e);
        }
        return indicadores;
    }
    protected Hashtable<Integer, Objetivo> getObjetivos()
    {
```

```
        String sql = "SELECT CODIGOOBJETIVO, OBJETIVO, FL_CUBO FROM
            OBJETIVOS";
55      Hashtable<Integer, Objetivo> objetivos = new Hashtable<Integer,
            Objetivo>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
60          Objetivo obj;
            while (rset.next())
            {
                obj = new Objetivo(rset.getInt("CODIGOOBJETIVO"), rset.
                    getString("OBJETIVO"), rset.getInt("FL_CUBO"));
                objetivos.put(new Integer(obj.getCodigo()), obj);
65          }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Objetivos");
            log.debug(e);
70      }
        return objetivos;
    }
    protected Hashtable<Integer, Perspectiva> getPerspectivas()
    {
75      String sql = "SELECT CODIGOPERSPECTIVA, NOME, ORDEM, FL_CUBO FROM
            PERSPECTIVAS";
        Hashtable<Integer, Perspectiva> perspectivas = new Hashtable<
            Integer, Perspectiva>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
80          ResultSet rset = stmt.executeQuery();
            Perspectiva persp;
            while (rset.next())
            {
                persp = new Perspectiva(rset.getInt("CODIGOPERSPECTIVA"),
                    rset.getString("NOME"), rset.getInt("ORDEM"), rset.
                    getInt("FL_CUBO"));
```

```
85                  perspectivas.put(new Integer(persp.getCodigo()), persp);
                }
            } catch (SQLException e)
            {
                log.error("Erro ao buscar Perspectivas");
90              log.debug(e);
            }
            return perspectivas;
        }
        protected Hashtable<Integer, Area> getAreas()
95      {
            String sql = "SELECT CODIGOAREA, NOME, MISSAO, VISAO, SIGLA,
               RESPONSAVEL, FL_CUBO " + " FROM AREAS";
            Hashtable<Integer, Area> areas = new Hashtable<Integer, Area>();
            try
            {
100             PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                Area area;
                while (rset.next())
                {
105                 area = new Area(rset.getInt("CODIGOAREA"), rset.getString
                       ("NOME"), rset.getString("MISSAO"), rset.getString("
                       VISAO"), rset.getString("SIGLA"), rset.getString("
                       RESPONSAVEL"), rset.getInt("FL_CUBO"));
                    areas.put(new Integer(area.getCodigo()), area);
                }
            } catch (SQLException e)
            {
110             log.error("Erro ao buscar Areas");
                log.debug(e);
            }
            return areas;
        }
115     protected Hashtable<Integer, FrequenciaColeta> getFrequenciasColeta()
        {
            String sql = "SELECT CODIGOFREQUENCIA, FREQUENCIA, FL_CUBO FROM
               FREQUENCIACOLETA";
```

```
        Hashtable<Integer, FrequenciaColeta> frequenciasColeta = new
            Hashtable<Integer, FrequenciaColeta>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            FrequenciaColeta fColeta;
            while (rset.next())
            {
                fColeta = new FrequenciaColeta(rset.getInt("
                    CODIGOFREQUENCIA"), rset.getString("FREQUENCIA"), rset
                    .getInt("FL_CUBO"));
                frequenciasColeta.put(new Integer(fColeta.getCodigo()),
                    fColeta);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar FrequenciasColeta");
            log.debug(e);
        }
        return frequenciasColeta;
    }
    protected Hashtable<Integer, UnidadeMedida> getUnidadesMedida()
    {
        String sql = "SELECT CODIGOUNIDADE, UNIDADEMEDIDA, SIGLA,
            PRECISAO, FL_CUBO FROM UNIDADEMEDIDA";
        Hashtable<Integer, UnidadeMedida> unidadesMedida = new Hashtable<
            Integer, UnidadeMedida>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            UnidadeMedida uMed;
            while (rset.next())
            {
                uMed = new UnidadeMedida(rset.getInt("CODIGOUNIDADE"),
                    rset.getString("UNIDADEMEDIDA"), rset.getString("SIGLA
                    "), rset.getInt("PRECISAO"), rset.getInt("FL_CUBO"));
```

```
                unidadesMedida.put(new Integer(uMed.getCodigo()), uMed);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar UnidadesMedida");
            log.debug(e);
        }
        return unidadesMedida;
    }
    protected Hashtable<Integer, Usuario> getUsuarios()
    {
        String sql = "SELECT CODIGOUSUARIO, USUARIO, SENHA, SUPERVISOR,
            GERENTE, CADASTROS, DADOS, RELATORIOS, NOME, FL_CUBO FROM
            USUARIOS ";
        Hashtable<Integer, Usuario> usrs = new Hashtable<Integer, Usuario
            >();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            Usuario usr;
            while (rset.next())
            {
                usr = new Usuario(rset.getInt("CODIGOUSUARIO"), rset.
                    getString("NOME"), rset.getString("USUARIO"), rset.
                    getString("SENHA"), rset.getInt("SUPERVISOR"), rset.
                    getInt("GERENTE"), rset.getInt("CADASTROS"), rset.
                    getInt("DADOS"), rset
                        .getInt("RELATORIOS"), rset.getInt("FL_CUBO"));
                usrs.put(new Integer(usr.getCodigo()), usr);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Usuários");
            log.debug(e);
        }
        return usrs;
    }
```

```
    protected Hashtable<Integer, IndicadorArea> getIndicadoresAreas()
180 {
        String sql = "SELECT IA.CODIGOAREA, IA.CODIGOPERSPECTIVA, IA.
            CODIGOOBJETIVO, " + "IA.CODIGOINDICADOR, IA.FORMULA, IA.
            CODIGOUNIDADE, IA.CODIGOFREQUENCIA, " + "IA.INDICADORMELHORIA,
             IA.DESCRICAOFORMULA, IA.CATEGORIA, IA.FL_CUBO "
                + "FROM INDICADORAREA IA, INDICADORES I, AREAS A,
                    PERSPECTIVAS P, OBJETIVOS O, " + "UNIDADEMEDIDA U,
                    FREQUENCIACOLETA F " + "WHERE IA.CODIGOAREA = A.
                    CODIGOAREA " + "AND IA.CODIGOPERSPECTIVA = P.
                    CODIGOPERSPECTIVA "
                + " AND IA.CODIGOINDICADOR = I.CODIGOINDICADOR "
                + "AND IA.CODIGOOBJETIVO = O.CODIGOOBJETIVO " + "AND IA.
                    CODIGOUNIDADE = U.CODIGOUNIDADE " + "AND IA.
                    CODIGOFREQUENCIA = F.CODIGOFREQUENCIA " + "AND IA.
                    FL_CUBO = " + FlagCubo.FlCubo.NOVO.ordinal();
185     Hashtable<Integer, IndicadorArea> indsAreas = new Hashtable<
            Integer, IndicadorArea>();
        try
        {
            PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
190         Area area;
            Perspectiva persp;
            Objetivo obj;
            Indicador ind;
            IndicadorArea indArea;
195         UnidadeMedida uMed;
            FrequenciaColeta fColeta;
            while (rset.next())
            {
                area = (Area) modelo.getAreas().get(new Integer(rset.
                    getInt("CODIGOAREA")));
200             persp = (Perspectiva) modelo.getPerspectivas().get(new
                    Integer(rset.getInt("CODIGOPERSPECTIVA")));
                obj = (Objetivo) modelo.getObjetivos().get(new Integer(
                    rset.getInt("CODIGOOBJETIVO")));
                ind = (Indicador) modelo.getIndicadores().get(new Integer
```

```
                (rset.getInt("CODIGOINDICADOR")));
            uMed = (UnidadeMedida) modelo.getUnidadesMedida().get(new
                Integer(rset.getInt("CODIGOUNIDADE")));
            fColeta = (FrequenciaColeta) modelo.getFrequenciasColeta
                ().get(new Integer(rset.getInt("CODIGOFREQUENCIA")));
            indArea = new IndicadorArea(area, persp, obj, ind, rset.
                getString("FORMULA"), uMed, fColeta, rset.getString("
                INDICADORMELHORIA"), rset.getString("DESCRICAOFORMULA"
                ), rset.getString("CATEGORIA"), rset.getInt("FL_CUBO")
                );
            if(indArea.getIndicador() == null) System.out.println("
                null");
            indsAreas.put(gerador.geraChave(indArea.getArea().
                getCodigo(), indArea.getPersp().getCodigo(), indArea.
                getObjetivo().getCodigo(), indArea.getIndicador().
                getCodigo()), indArea);
        }
    } catch (SQLException e)
    {
        log.error("Erro ao buscar IndicadoresAreas.");
        log.debug(e);
    }
    sql = "SELECT * FROM INDICADORVETOR WHERE APROVADO=1";
    try
    {
        PreparedStatement stmt = conn.prepareStatement(sql);
        ResultSet rset = stmt.executeQuery();
        IndicadorArea origem = null;
        IndicadorArea destino = null;
        while (rset.next())
        {
            origem = indsAreas.get(gerador.geraChave(rset.getInt("
                CODIGOAREA"), rset.getInt("CODIGOPERSPECTIVA"), rset.
                getInt("CODIGOOBJETIVO"), rset.getInt("CODIGOINDICADOR
                ")));
            destino = indsAreas.get(gerador.geraChave(rset.getInt("
                CODIGOAREADESTINO"), rset.getInt("
                CODIGOPERSPECTIVADESTINO"), rset.getInt("
```

```
                    CODIGOOBJETIVODESTINO"), rset.getInt("
                    CODIGOINDICADORDESTINO")));
225                 if (origem != null && destino != null)
                    {
                        origem.getFilhos().add(destino);
                        destino.setPai(origem);
                    }
230             }
            } catch (SQLException e)
            {
                log.error("Erro buscando vinculação de indicadores");
                log.debug(e);
235         }
            setIndicadoresResultado(indsAreas);
            return indsAreas;
        }
        protected Hashtable<Integer, ObjetivoTema> getObjetivosTemas()
240     {
            String sql = "SELECT OT.CODIGOAREA, OT.CODIGOPERSPECTIVA, OT.
                CODIGOOBJETIVO, " + "OT.DATAINICIO,  OT.DATAFIM, OT.
                DETALHAMENTO, OT.FL_CUBO " + "FROM OBJETIVOS_TEMAS OT, AREAS A
                , PERSPECTIVAS P, OBJETIVOS O "
                    + "WHERE OT.CODIGOAREA = A.CODIGOAREA " + "AND OT.
                        CODIGOPERSPECTIVA = P.CODIGOPERSPECTIVA " + "AND OT.
                        CODIGOOBJETIVO = O.CODIGOOBJETIVO " + "AND OT.FL_CUBO
                        = " + FlagCubo.FlCubo.NOVO.ordinal();
            Hashtable<Integer, ObjetivoTema> objTemas = new Hashtable<Integer
                , ObjetivoTema>();
            try
245         {
                PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                Area area;
                Perspectiva persp;
250             Objetivo obj;
                ObjetivoTema objTema;
                while (rset.next())
                {
```

```
                area = (Area) modelo.getAreas().get(new Integer(rset.
                    getInt("CODIGOAREA")));
255             persp = (Perspectiva) modelo.getPerspectivas().get(new
                    Integer(rset.getInt("CODIGOPERSPECTIVA")));
                obj = (Objetivo) modelo.getObjetivos().get(new Integer(
                    rset.getInt("CODIGOOBJETIVO")));
                objTema = new ObjetivoTema(area, persp, obj, rset.getDate
                    ("DATAINICIO"), rset.getDate("DATAFIM"), rset.
                    getString("DETALHAMENTO"));
                objTemas.put(gerador.geraChave(objTema.getArea().
                    getCodigo(), objTema.getPersp().getCodigo(), objTema.
                    getObjetivo().getCodigo()), objTema);
            }
260         } catch (SQLException e)
            {
                log.error("Erro ao pegar ObjetivosTemas.");
                log.debug(e);
            }
265         return objTemas;
        }
        protected Hashtable<Integer, UsuarioArea> getUsuariosArea()
        {
            String sql = "SELECT UA.CODIGOUSUARIO, UA.CODIGOAREA, UA.FL_CUBO
                " + "FROM USUARIOSAREAS UA, USUARIOS U, AREAS A " + "WHERE U.
                CODIGOUSUARIO = UA.CODIGOUSUARIO " + "AND A.CODIGOAREA = UA.
                CODIGOAREA";
270         Hashtable<Integer, UsuarioArea> usuarioArea = new Hashtable<
                Integer, UsuarioArea>();
            Area area;
            Usuario usr;
            try
            {
275             PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                UsuarioArea usrArea;
                while (rset.next())
                {
280                 area = (Area) modelo.getAreas().get(new Integer(rset.
```

```
                    getInt("CODIGOAREA")));
                usr = (Usuario) modelo.getUsuarios().get(new Integer(rset
                    .getInt("CODIGOUSUARIO")));
                usrArea = new UsuarioArea(area, usr, rset.getInt("FL_CUBO
                    "));
                usuarioArea.put(gerador.geraChave(usrArea.getUsuario().
                    getCodigo(), usrArea.getArea().getCodigo()), usrArea);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar UsuariosArea");
            log.debug(e);
        }
        return usuarioArea;
    }
    public ModeloBSC getModeloBSC()
    {
        conecta();
        modelo.setAreas(this.getAreas());
        modelo.setPerspectivas(this.getPerspectivas());
        modelo.setObjetivos(this.getObjetivos());
        modelo.setIndicadores(this.getIndicadores());
        modelo.setUsuarios(this.getUsuarios());
        modelo.setFrequenciasColeta(this.getFrequenciasColeta());
        modelo.setUnidadesMedida(this.getUnidadesMedida());
        modelo.setUsuariosAreas(this.getUsuariosArea());
        modelo.setObjetivosTemas(this.getObjetivosTemas());
        modelo.setIndicadoresAreas(this.getIndicadoresAreas());
        modelo.setValores(this.getValores());
        desconecta();
        return modelo;
    }
    protected abstract Hashtable<Integer, Valor> getValores();
    protected void setIndicadoresResultado(Hashtable<Integer,
        IndicadorArea> indsAreas)
    {
        String sql = "SELECT * FROM INDICADORAREA WHERE (CODIGOAREA,
            CODIGOPERSPECTIVA, CODIGOOBJETIVO, CODIGOINDICADOR) NOT IN (
```

```
            SELECT CODIGOAREADESTINO, CODIGOPERSPECTIVADESTINO,
            CODIGOOBJETIVODESTINO, CODIGOINDICADORDESTINO FROM
            INDICADORVETOR WHERE CODIGOAREA = CODIGOAREADESTINO AND
            CODIGOPERSPECTIVA = CODIGOPERSPECTIVADESTINO AND
            CODIGOOBJETIVO = CODIGOOBJETIVODESTINO)";
        try
        {
315         PreparedStatement stmt = conn.prepareStatement(sql);
            ResultSet rset = stmt.executeQuery();
            IndicadorArea indArea = null;
            while (rset.next())
            {
320             indArea = indsAreas.get(gerador.geraChave(rset.getInt("
                    CODIGOAREA"), rset.getInt("CODIGOPERSPECTIVA"), rset.
                    getInt("CODIGOOBJETIVO"), rset.getInt("CODIGOINDICADOR
                    ")));
                if (indArea != null)
                    indArea.setIndicadorResultado(true);
            }
        } catch (SQLException e)
        {
325         log.error("Erro buscando indicadores de resultado");
            log.debug(e);
        }
    }
330
    public void conecta()
    {
        super.conectaOrigem();
    }
335 }
```

Listagem 6.9: DAOMigradorOrigemMysql.java

```
package ufsc.ine.dao;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
5 import java.util.Hashtable;
```

```
import ufsc.ine.modelo.IndicadorArea;
import ufsc.ine.modelo.Valor;
public class DAOMigradorOrigemMysql extends DAOMigradorOrigem
{
    protected Hashtable<Integer, Valor> getValores()
    {
        String sql = "SELECT SUM(PREVISTO*(1-abs(sign(
            CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%M
            %' THEN 10 ELSE 1 END))))) AS META_TENDENCIAL, SUM(PREVISTO
            *(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
            UTILIZAFORMULA like '%P%' THEN 12 ELSE 9 END))))) AS
            PADRAO_TENDENCIAL, SUM(PREVISTO*(1-abs(sign(
            CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%M
            %' THEN 30 ELSE 13 END))))) AS META_OTIMISTA, SUM(PREVISTO*(1-
            abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA
             like '%P%' THEN 19 ELSE 16 END))))) AS PADRAO_OTIMISTA, SUM(
            PREVISTO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
            UTILIZAFORMULA like '%M%' THEN 31 ELSE 14 END))))) AS
            META_PESSIMISTA, SUM(PREVISTO*(1-abs(sign(
            CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%P
            %' THEN 18 ELSE 17 END))))) AS PADRAO_PESSIMISTA, SUM(
            REALIZADO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
            UTILIZAFORMULA like '%R%' THEN 11 ELSE 3 END))))) AS REALIZADO
            , AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
            CODIGOTEMAESTRATEGICO, AP.CODIGOOBJETIVO, AP.CODIGOINDICADOR,
            AP.DATA FROM ACOMPANHAMENTO_PR AP, INDICADORAREA IA WHERE AP.
            CODIGOAREA = IA.CODIGOAREA AND AP.CODIGOPERSPECTIVA = IA.
            CODIGOPERSPECTIVA AND AP.CODIGOTEMAESTRATEGICO = IA.
            CODIGOTEMAESTRATEGICO AND AP.CODIGOOBJETIVO = IA.
            CODIGOOBJETIVO AND AP.CODIGOINDICADOR = IA.CODIGOINDICADOR
            GROUP BY AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
            CODIGOTEMAESTRATEGICO, AP.CODIGOOBJETIVO, AP.CODIGOINDICADOR,
            AP.DATA having SUM(REALIZADO*(1-abs(sign(
            CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%R
            %' THEN 11 ELSE 3 END))))) is not null";
        Hashtable<Integer, Valor> valores = new Hashtable<Integer, Valor
            >();
        try
```

```
15          {
                PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                IndicadorArea indArea;
                Valor valor;
20              while (rset.next())
                {
                    indArea = modelo.getIndicadoresAreas().get(gerador.
                        geraChave(rset.getInt("CODIGOAREA"), rset.getInt("
                        CODIGOPERSPECTIVA"), rset.getInt("CODIGOOBJETIVO"),
                        rset.getInt("CODIGOINDICADOR")));
                    valor = new Valor();
                    valor.setIndicadorArea(indArea);
25                  valor.setData(rset.getDate("DATA"));
                    valor.setMetaTendencial(rset.getFloat("META_TENDENCIAL"))
                        ;
                    valor.setMetaOtimista(rset.getFloat("META_OTIMISTA"));
                    valor.setMetaPessimista(rset.getFloat("META_PESSIMISTA"))
                        ;
                    valor.setPadraoTendencial(rset.getFloat("
                        PADRAO_TENDENCIAL"));
30                  valor.setPadraoOtimista(rset.getFloat("PADRAO_OTIMISTA"))
                        ;
                    valor.setPadraoPessimista(rset.getFloat("
                        PADRAO_PESSIMISTA"));
                    valor.setRealizado(rset.getFloat("REALIZADO"));
                    valores.put(gerador.geraChave(indArea.getArea().getCodigo
                        (), indArea.getPersp().getCodigo(), indArea.
                        getObjetivo().getCodigo(), indArea.getIndicador().
                        getCodigo(), valor.getData().toString()), valor);
                }
35          } catch (SQLException e)
            {
                log.error("Erro ao buscar Valores");
                log.debug(e);
            }
40          return valores;
        }
```

```
}
```

Listagem 6.10: DAOMigradorOrigemOracle.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.dao;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Hashtable;
import ufsc.ine.modelo.IndicadorArea;
import ufsc.ine.modelo.ModeloBSC;
import ufsc.ine.modelo.Valor;
import ufsc.ine.util.GeradorChave;
public class DAOMigradorOrigemOracle extends DAOMigradorOrigem
{
    public DAOMigradorOrigemOracle()
    {
        gerador = new GeradorChave();
        modelo = new ModeloBSC();
    }
    protected Hashtable<Integer, Valor> getValores()
    {
        String sql = "SELECT * FROM ( SELECT SUM(PREVISTO*(1-abs(sign(
            CODIGOTIPOACOMPANHAMENTO-" + "(CASE WHEN IA.UTILIZAFORMULA
            like '%M%' THEN 10 ELSE 1 END))))) AS META_TENDENCIAL, " + "
            SUM(PREVISTO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-"
                + "(CASE WHEN IA.UTILIZAFORMULA like '%P%' THEN 12 ELSE 9
                    END))))) AS PADRAO_TENDENCIAL, " + "SUM(PREVISTO*(1-
                abs(sign(CODIGOTIPOACOMPANHAMENTO-" + "(CASE WHEN IA.
                UTILIZAFORMULA like '%M%' THEN 30 ELSE 13 END))))) AS
                META_OTIMISTA, "
                + "SUM(PREVISTO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-" +
                "(CASE WHEN IA.UTILIZAFORMULA like '%P%' THEN 19 ELSE
                16 END))))) AS PADRAO_OTIMISTA, " + "SUM(PREVISTO*(1-
                abs(sign(CODIGOTIPOACOMPANHAMENTO-"
```

```
                + "(CASE WHEN IA.UTILIZAFORMULA like '%M%' THEN 31 ELSE
                    14 END))))) AS META_PESSIMISTA, " + "SUM(PREVISTO*(1-
                    abs(sign(CODIGOTIPOACOMPANHAMENTO-"
                + "(CASE WHEN IA.UTILIZAFORMULA like '%P%' THEN 18 ELSE
                    17 END))))) AS PADRAO_PESSIMISTA, " + "SUM(REALIZADO
                    *(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-" + "(CASE WHEN
                    IA.UTILIZAFORMULA like '%R%' THEN 11 ELSE 3 END)))))
                    AS REALIZADO, "
                + "AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
                    CODIGOTEMAESTRATEGICO, AP.CODIGOOBJETIVO, " + "AP.
                    CODIGOINDICADOR, AP.DATA FROM ACOMPANHAMENTO_PR AP,
                    INDICADORAREA IA " + "WHERE AP.CODIGOAREA = IA.
                    CODIGOAREA "
                + "AND AP.CODIGOPERSPECTIVA = IA.CODIGOPERSPECTIVA " + "
                    AND AP.CODIGOTEMAESTRATEGICO = IA.
                    CODIGOTEMAESTRATEGICO " + "AND AP.CODIGOOBJETIVO = IA.
                    CODIGOOBJETIVO " + "AND AP.CODIGOINDICADOR = IA.
                    CODIGOINDICADOR "
                + "GROUP BY AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
                    CODIGOTEMAESTRATEGICO, " + "AP.CODIGOOBJETIVO, AP.
                    CODIGOINDICADOR, AP.DATA) T " + "WHERE T.REALIZADO IS
                    NOT NULL ";
            Hashtable<Integer, Valor> valores = new Hashtable<Integer, Valor
                >();
            try
            {
                PreparedStatement stmt = conn.prepareStatement(sql);
                ResultSet rset = stmt.executeQuery();
                IndicadorArea indArea;
                Valor valor;
                while (rset.next())
                {
                    indArea = modelo.getIndicadoresAreas().get(gerador.
                        geraChave(rset.getInt("CODIGOAREA"), rset.getInt("
                        CODIGOPERSPECTIVA"), rset.getInt("CODIGOOBJETIVO"),
                        rset.getInt("CODIGOINDICADOR")));
                    valor = new Valor();
                    valor.setIndicadorArea(indArea);
```

```
                valor.setData(rset.getDate("DATA"));
                valor.setMetaTendencial(rset.getFloat("META_TENDENCIAL"))
                    ;
                valor.setMetaOtimista(rset.getFloat("META_OTIMISTA"));
                valor.setMetaPessimista(rset.getFloat("META_PESSIMISTA"))
                    ;
                valor.setPadraoTendencial(rset.getFloat("
                    PADRAO_TENDENCIAL"));
                valor.setPadraoOtimista(rset.getFloat("PADRAO_OTIMISTA"))
                    ;
                valor.setPadraoPessimista(rset.getFloat("
                    PADRAO_PESSIMISTA"));
                valor.setRealizado(rset.getFloat("REALIZADO"));
                valores.put(gerador.geraChave(indArea.getArea().getCodigo
                    (), indArea.getPersp().getCodigo(), indArea.
                    getObjetivo().getCodigo(), indArea.getIndicador().
                    getCodigo(), valor.getData().toString()), valor);
            }
        } catch (SQLException e)
        {
            log.error("Erro ao buscar Valores");
            log.debug(e);
        }
        return valores;
    }
}
```

Listagem 6.11: DAOMigradorOrigemPgsql.java

```
package ufsc.ine.dao;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Hashtable;
import ufsc.ine.modelo.IndicadorArea;
import ufsc.ine.modelo.Valor;
public class DAOMigradorOrigemPgsql extends DAOMigradorOrigem
{
    @Override
```

```
protected Hashtable<Integer, Valor> getValores()
{
    String sql = "SELECT SUM(PREVISTO*(1-abs(sign(
        CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%M
        %' THEN 10 ELSE 1 END))))) AS META_TENDENCIAL, SUM(PREVISTO
        *(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
        UTILIZAFORMULA like '%P%' THEN 12 ELSE 9 END))))) AS
        PADRAO_TENDENCIAL, SUM(PREVISTO*(1-abs(sign(
        CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%M
        %' THEN 30 ELSE 13 END))))) AS META_OTIMISTA, SUM(PREVISTO*(1-
        abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA
         like '%P%' THEN 19 ELSE 16 END))))) AS PADRAO_OTIMISTA, SUM(
        PREVISTO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
        UTILIZAFORMULA like '%M%' THEN 31 ELSE 14 END))))) AS
        META_PESSIMISTA, SUM(PREVISTO*(1-abs(sign(
        CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%P
        %' THEN 18 ELSE 17 END))))) AS PADRAO_PESSIMISTA, SUM(
        REALIZADO*(1-abs(sign(CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.
        UTILIZAFORMULA like '%R%' THEN 11 ELSE 3 END))))) AS REALIZADO
        , AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
        CODIGOTEMAESTRATEGICO, AP.CODIGOOBJETIVO, AP.CODIGOINDICADOR,
        AP.DATA FROM ACOMPANHAMENTO_PR AP, INDICADORAREA IA WHERE AP.
        CODIGOAREA = IA.CODIGOAREA AND AP.CODIGOPERSPECTIVA = IA.
        CODIGOPERSPECTIVA AND AP.CODIGOTEMAESTRATEGICO = IA.
        CODIGOTEMAESTRATEGICO AND AP.CODIGOOBJETIVO = IA.
        CODIGOOBJETIVO AND AP.CODIGOINDICADOR = IA.CODIGOINDICADOR
        GROUP BY AP.CODIGOAREA, AP.CODIGOPERSPECTIVA, AP.
        CODIGOTEMAESTRATEGICO, AP.CODIGOOBJETIVO, AP.CODIGOINDICADOR,
        AP.DATA having SUM(REALIZADO*(1-abs(sign(
        CODIGOTIPOACOMPANHAMENTO-(CASE WHEN IA.UTILIZAFORMULA like '%R
        %' THEN 11 ELSE 3 END))))) is not null";
    Hashtable<Integer, Valor> valores = new Hashtable<Integer, Valor
        >();
15  try
    {
        PreparedStatement stmt = conn.prepareStatement(sql);
        ResultSet rset = stmt.executeQuery();
        IndicadorArea indArea;
```

```
20              Valor valor;
                while (rset.next())
                {
                    indArea = modelo.getIndicadoresAreas().get(gerador.
                        geraChave(rset.getInt("CODIGOAREA"), rset.getInt("
                        CODIGOPERSPECTIVA"), rset.getInt("CODIGOOBJETIVO"),
                        rset.getInt("CODIGOINDICADOR")));
                    valor = new Valor();
25                  valor.setIndicadorArea(indArea);
                    valor.setData(rset.getDate("DATA"));
                    valor.setMetaTendencial(rset.getFloat("META_TENDENCIAL"))
                        ;
                    valor.setMetaOtimista(rset.getFloat("META_OTIMISTA"));
                    valor.setMetaPessimista(rset.getFloat("META_PESSIMISTA"))
                        ;
30                  valor.setPadraoTendencial(rset.getFloat("
                        PADRAO_TENDENCIAL"));
                    valor.setPadraoOtimista(rset.getFloat("PADRAO_OTIMISTA"))
                        ;
                    valor.setPadraoPessimista(rset.getFloat("
                        PADRAO_PESSIMISTA"));
                    valor.setRealizado(rset.getFloat("REALIZADO"));
                    valores.put(gerador.geraChave(indArea.getArea().getCodigo
                        (), indArea.getPersp().getCodigo(), indArea.
                        getObjetivo().getCodigo(), indArea.getIndicador().
                        getCodigo(), valor.getData().toString()), valor);
35              }
            } catch (SQLException e)
            {
                log.error("Erro ao buscar Valores");
                log.debug(e);
40          }
            return valores;
        }
    }
```

Listagem 6.12: GeradorLogger.java

```
/*
```

```
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.log;

import java.io.InputStream;
import java.util.Properties;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

public class GeradorLogger {
    public static Logger getLogger(String classname) {
        Logger log = Logger.getLogger(classname);
        try {
            Properties p = new Properties();
            InputStream ip = GeradorLogger.class
                    .getResourceAsStream("/ufsc/ine/log/log4j.properties"
                        );
            p.load(ip);
            PropertyConfigurator.configure(p);
        } catch (Exception e) {
            System.out.println("Error initializing  logger for CLASS("
                    + classname + "):" + e);
            e.printStackTrace();
        }
        return log;
    }

}
```

Listagem 6.13: log4j.properties

```
# Set root logger level to DEBUG and its only appender to A1.
log4j.rootLogger=DEBUG, A1
# Console Appender
# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender
# A1 uses PatternLayout.
```

```
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d{HH:mm}(%-4r) [%t] %-5p %c
    {1} %x - %m%n
```

Listagem 6.14: Area.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

// missao, visao, sigla, responsavel
public class Area extends Informacao {

    private String missao;

    private String visao;

    private String sigla;

    private String responsavel;

    /**
     * Contrutor para facililtar a criação, passando todos os parâmetros
     *
     * @param cd
     * @param nm
     * @param missao
     * @param visao
     * @param sigla
     * @param resp
     */
    public Area(int cd, String nm, String missao, String visao, String
        sigla,
            String resp, int flCubo) {
        this.codigo = cd;
        this.nome = nm;
        this.missao = missao;
```

```
        this.visao = visao;
        this.sigla = sigla;
        this.responsavel = resp;
        this.flCubo = flCubo;
    }

    public String getMissao() {
        return missao;
    }

    public void setMissao(String missao) {
        this.missao = missao;
    }

    public String getResponsavel() {
        return responsavel;
    }

    public void setResponsavel(String responsavel) {
        this.responsavel = responsavel;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
    }

    public String getVisao() {
        return visao;
    }

    public void setVisao(String visao) {
        this.visao = visao;
    }
}
```

Listagem 6.15: Auditoria.java

```
/*
 * Binara Informática
 * Created on 21/02/2006
 */
package ufsc.ine.modelo;

import java.sql.Timestamp;
import java.util.GregorianCalendar;

public class Auditoria {
    private int codigo;

    private Timestamp inicioLeitura;

    private Timestamp fimLeitura;

    private Timestamp inicioEscrita;

    private Timestamp fimEscrita;

    private int registrosLidos;

    private int registrosNovosLidos;

    public int getCodigo() {
        return codigo;
    }

    public void iniciaLeitura() {
        this.inicioLeitura = new Timestamp(new GregorianCalendar()
                .getTimeInMillis());
    }

    public void encerraLeitura() {
        this.fimLeitura = new Timestamp(new GregorianCalendar()
                .getTimeInMillis());
```

```
    }

    public void iniciaEscrita() {
        this.inicioEscrita = new Timestamp(new GregorianCalendar()
                .getTimeInMillis());

    }

    public void encerraEscrita() {
        this.fimEscrita = new Timestamp(new GregorianCalendar()
                .getTimeInMillis());

    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public Timestamp getFimEscrita() {
        return fimEscrita;
    }

    public void setFimEscrita(Timestamp fimEscrita) {
        this.fimEscrita = fimEscrita;
    }

    public Timestamp getFimLeitura() {
        return fimLeitura;
    }

    public void setFimLeitura(Timestamp fimLeitura) {
        this.fimLeitura = fimLeitura;
    }

    public Timestamp getInicioEscrita() {
        return inicioEscrita;
    }

```

```
    public void setInicioEscrita(Timestamp inicioEscrita) {
        this.inicioEscrita = inicioEscrita;
    }

    public Timestamp getInicioLeitura() {
        return inicioLeitura;
    }

    public void setInicioLeitura(Timestamp inicioLeitura) {
        this.inicioLeitura = inicioLeitura;
    }

    public int getRegistrosLidos() {
        return registrosLidos;
    }

    public void setRegistrosLidos(int registrosLidos) {
        this.registrosLidos = registrosLidos;
    }

    public int getRegistrosNovosLidos() {
        return registrosNovosLidos;
    }

    public void setRegistrosNovosLidos(int registrosNovosLidos) {
        this.registrosNovosLidos = registrosNovosLidos;
    }
}
```

Listagem 6.16: Cenario.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

public class Cenario extends Informacao {
    public Cenario(int cd, String nm, int flCubo) {
```

```
        this.codigo = cd;
        this.nome = nm;
        this.flCubo = flCubo;
    }
}
```

Listagem 6.17: FlagCubo.java

```
/*
 * Binara Informática
 * Created on 27/10/2005
 */
package ufsc.ine.modelo;

public class FlagCubo
{
    public static enum FlCubo{MIGRADO, NOVO, ATUALIZADO};
}
```

Listagem 6.18: FrequenciaColeta.java

```
/*
 * Binara Informática
 * Created on 27/10/2005
 */
package ufsc.ine.modelo;

public class FrequenciaColeta extends Informacao{

    public FrequenciaColeta(int cd, String nm, int flCubo) {
        this.codigo = cd;
        this.nome = nm;
        this.flCubo = flCubo;
    }

}
```

Listagem 6.19: Indicador.java

```
/*
 * Binara Informática
```

```
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

public class Indicador extends Informacao {

    public Indicador(int cd, String nm, int flCubo) {
        this.codigo = cd;
        this.nome = nm;
        this.flCubo = flCubo;
    }

}
```

Listagem 6.20: IndicadorArea.java

```
/*
 * Binara Informática
 * Created on 25/10/2005
 */
package ufsc.ine.modelo;

import java.util.LinkedList;

// informações de indicadorarea
// codigo area, perspectiva, objetivo, boolean(existeFormula),
   unidademedida,
// frequenciacoleta
// indicadorDeMelhoria, descricaoFormula, categoria
public class IndicadorArea {
    private Area area;

    private Perspectiva persp;

    private Objetivo objetivo;

    private boolean existeFormula;

    private boolean indicadorResultado;
```

```
    private String formula;

    private UnidadeMedida unidadeMedida;

    private FrequenciaColeta frequenciaColeta;

    private String indicadorMelhoria;

    private String descFormula;

    private String categoria;

    private String desc;

    private int flCubo;

    private LinkedList<IndicadorArea> filhos;

    private IndicadorArea pai;

    private Indicador indicador;

    public IndicadorArea(Area area, Perspectiva persp, Objetivo obj,
            Indicador ind, String form, UnidadeMedida med,
            FrequenciaColeta coleta, String indM, String descFormula,
            String cat, int flCubo) {
        this.area = area;
        this.persp = persp;
        this.objetivo = obj;
        this.indicador = ind;
        this.formula = form;
        this.unidadeMedida = med;
        this.frequenciaColeta = coleta;
        this.indicadorMelhoria = indM;
        this.descFormula = descFormula;
        this.categoria = cat;
        this.flCubo = flCubo;
```

```
        this.filhos = new LinkedList<IndicadorArea>();
        this.indicadorResultado = false;
    }

    public Area getArea() {
        return area;
    }

    public void setArea(Area area) {
        this.area = area;
    }

    public String getCategoria() {
        return categoria;
    }

    public void setCategoria(String categoria) {
        this.categoria = categoria;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public String getDescFormula() {
        return descFormula;
    }

    public void setDescFormula(String descFormula) {
        this.descFormula = descFormula;
    }

    public boolean isExisteFormula() {
        return existeFormula;
```

```
    }

100

    public void setExisteFormula(boolean existeFormula) {
        this.existeFormula = existeFormula;
    }

105 public String getIndicadorMelhoria() {
        return indicadorMelhoria;
    }

    public void setIndicadorMelhoria(String indicadorMelhoria) {
110     this.indicadorMelhoria = indicadorMelhoria;
    }

    public Perspectiva getPersp() {
        return persp;
115 }

    public void setPersp(Perspectiva persp) {
        this.persp = persp;
    }

120

    public int getFlCubo() {
        return flCubo;
    }

125 public void setFlCubo(int flCubo) {
        this.flCubo = flCubo;
    }

    public String getFormula() {
130     return formula;
    }

    public void setFormula(String formula) {
        this.formula = formula;
135 }
```

```
    public Indicador getIndicador() {
        return indicador;
    }

140

    public void setIndicador(Indicador indicador) {
        this.indicador = indicador;
    }

    public Objetivo getObjetivo() {
145     return objetivo;
    }

    public void setObjetivo(Objetivo objetivo) {
        this.objetivo = objetivo;
150 }

    public FrequenciaColeta getFrequenciaColeta() {
        return frequenciaColeta;
    }
155

    public void setFrequenciaColeta(FrequenciaColeta frequenciaColeta) {
        this.frequenciaColeta = frequenciaColeta;
    }

160

    public UnidadeMedida getUnidadeMedida() {
        return unidadeMedida;
    }

    public void setUnidadeMedida(UnidadeMedida unidadeMedida) {
165     this.unidadeMedida = unidadeMedida;
    }

    public boolean getIndicadorResultado()
    {
170     return indicadorResultado;
    }

    public void setIndicadorResultado(boolean indicadorResultado)
```

```
175     {
            this.indicadorResultado = indicadorResultado;
        }

        public LinkedList<IndicadorArea> getFilhos()
180     {
            return filhos;
        }

        public void setFilhos(LinkedList<IndicadorArea> filhos)
185     {
            this.filhos = filhos;
        }

        public IndicadorArea getPai()
190     {
            return pai;
        }

        public void setPai(IndicadorArea pai)
195     {
            this.pai = pai;
        }
    }
```

Listagem 6.21: Informacao.java

```
    /*
     * Binara Informática
     * Created on 13/10/2005
     */
5   package ufsc.ine.modelo;

    public abstract class Informacao {
        protected int codigo;
        protected int flCubo;
10      protected String nome;
        public Informacao(){
```

```
    }
    public Informacao(int cd, String nm) {
        this.codigo = cd;
        this.nome = nm;
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getFlCubo() {
        return flCubo;
    }
    public void setFlCubo(int fl_cubo) {
        this.flCubo = fl_cubo;
    }
}
```

Listagem 6.22: ModeloBSC.java

```
/*
 * Binara Informática
 * Created on 20/10/2005
 */
package ufsc.ine.modelo;

import java.util.Hashtable;
```

```
public class ModeloBSC {

    private Hashtable<Integer,Area> areas;

    private Hashtable<Integer,Cenario> cenarios;

    private Hashtable<Integer,UnidadeMedida> unidadesMedida;

    private Hashtable<Integer,Indicador> indicadores;

    private Hashtable<Integer,Objetivo> objetivos;

    private Hashtable<Integer,Perspectiva> perspectivas;

    private Hashtable<Integer,FrequenciaColeta> frequenciasColeta;

    private Hashtable<Integer,Usuario> usuarios;

    private Hashtable<Integer,UsuarioArea> usuariosAreas;

    private Hashtable<Integer,IndicadorArea> indicadoresAreas;

    private Hashtable<Integer,ObjetivoTema> objetivosTemas;

    private Hashtable<Integer,Valor> valores;

    private Auditoria auditoria;

    public Auditoria getAuditoria()
    {
        return auditoria;
    }

    public void setAuditoria(Auditoria auditoria)
    {
        this.auditoria = auditoria;
    }
```

```
    public Hashtable<Integer,Indicador> getIndicadores() {
        return indicadores;
    }

50

    public void setIndicadores(Hashtable<Integer,Indicador> indicadores)
        {
        this.indicadores = indicadores;
    }

55  public Hashtable<Integer,IndicadorArea> getIndicadoresAreas() {
        return indicadoresAreas;
    }

    public void setIndicadoresAreas(Hashtable<Integer,IndicadorArea>
        indicaresAreas) {
60      this.indicadoresAreas = indicaresAreas;
    }

    public Hashtable<Integer,ObjetivoTema> getObjetivosTemas() {
        return objetivosTemas;
65  }

    public void setObjetivosTemas(Hashtable<Integer,ObjetivoTema>
        objetivosTemas) {
        this.objetivosTemas = objetivosTemas;
    }

70

    public Hashtable<Integer,Area> getAreas() {
        return areas;
    }

75  public void setAreas(Hashtable<Integer,Area> areas) {
        this.areas = areas;
    }

    public Hashtable<Integer,Cenario> getCenarios() {
80      return cenarios;
    }
```

```
    public void setCenarios(Hashtable<Integer,Cenario> cenarios) {
        this.cenarios = cenarios;
    }

    public Hashtable<Integer,Objetivo> getObjetivos() {
        return objetivos;
    }

    public void setObjetivos(Hashtable<Integer,Objetivo> objetivos) {
        this.objetivos = objetivos;
    }

    public Hashtable<Integer,Perspectiva> getPerspectivas() {
        return perspectivas;
    }

    public void setPerspectivas(Hashtable<Integer,Perspectiva>
        perspectivas) {
        this.perspectivas = perspectivas;
    }

    public Hashtable<Integer,Usuario> getUsuarios() {
        return usuarios;
    }

    public void setUsuarios(Hashtable<Integer,Usuario> usuarios) {
        this.usuarios = usuarios;
    }

    public Hashtable<Integer,UsuarioArea> getUsuariosAreas() {
        return usuariosAreas;
    }

    public void setUsuariosAreas(Hashtable<Integer,UsuarioArea>
        usuariosAreas) {
        this.usuariosAreas = usuariosAreas;
    }
```

```
    public Hashtable<Integer,UnidadeMedida> getUnidadesMedida() {
        return unidadesMedida;
    }

    public void setUnidadesMedida(Hashtable<Integer,UnidadeMedida>
        unidadesMedida) {
        this.unidadesMedida = unidadesMedida;
    }

    public Hashtable<Integer,FrequenciaColeta> getFrequenciasColeta() {
        return frequenciasColeta;
    }

    public void setFrequenciasColeta(Hashtable<Integer,FrequenciaColeta>
        frequenciasColeta) {
        this.frequenciasColeta = frequenciasColeta;
    }

    public void setValores(Hashtable<Integer,Valor> valores) {
        this.valores = valores;

    }

    public Hashtable<Integer,Valor> getValores() {
        return valores;
    }

}
```

Listagem 6.23: Objetivo.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

// pegar d eobjetivotema
```

```
// codigo area, perspectiva, dataInicio, dataFim, detalhamento
public class Objetivo extends Informacao {
    public Objetivo(int cd, String nm, int flCubo){
        this.codigo = cd;
        this.nome = nm;
        this.flCubo = flCubo;
    }
}
```

Listagem 6.24: ObjetivoTema.java

```
/*
 * Binara Informática
 * Created on 25/10/2005
 */
package ufsc.ine.modelo;

import java.util.Date;

public class ObjetivoTema{
    private Area area;

    private Perspectiva persp;

    private Date dataInicio;

    private Date dataFim;

    private String detalhamento;

    private Objetivo objetivo;

    public ObjetivoTema(Area area, Perspectiva persp, Objetivo obj, java.
        sql.Date dataI, java.sql.Date dataF, String det) {
        this.area = area;
        this.persp = persp;
        this.objetivo = obj;
        this.dataInicio = dataI;
        this.dataFim = dataF;
```

```
        this.detalhamento = det;
    }

    public Area getArea() {
        return area;
    }

    public void setArea(Area area) {
        this.area = area;
    }

    public Date getDataFim() {
        return dataFim;
    }

    public void setDataFim(Date dataFim) {
        this.dataFim = dataFim;
    }

    public Date getDataInicio() {
        return dataInicio;
    }

    public void setDataInicio(Date dataInicio) {
        this.dataInicio = dataInicio;
    }

    public String getDetalhamento() {
        return detalhamento;
    }

    public void setDetalhamento(String detalhamento) {
        this.detalhamento = detalhamento;
    }

    public Perspectiva getPersp() {
        return persp;
    }
```

```
    public void setPersp(Perspectiva persp) {
        this.persp = persp;
    }

    public Objetivo getObjetivo() {
        return objetivo;
    }

    public void setObjetivo(Objetivo obj) {
        this.objetivo = obj;
    }
}
```

Listagem 6.25: Perspectiva.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

public class Perspectiva extends Informacao {
    private int ordem;

    public Perspectiva(int cd, String nm, int ordem, int flCubo) {
        this.codigo = cd;
        this.nome = nm;
        this.ordem = ordem;
        this.flCubo = flCubo;
    }

    public int getOrdem() {
        return ordem;
    }

    public void setOrdem(int ordem) {
        this.ordem = ordem;
    }
```

```
}
```

Listagem 6.26: UnidadeMedida.java

```
/*
 * Binara Informática
 * Created on 27/10/2005
 */
package ufsc.ine.modelo;

public class UnidadeMedida extends Informacao {

    private String sigla;

    private int precisao;

    public UnidadeMedida(int cd, String nm, String sigla, int prec, int
        flCubo) {
        this.codigo = cd;
        this.nome = nm;
        this.flCubo = flCubo;
        this.sigla = sigla;
        this.precisao = prec;
    }

    public int getPrecisao() {
        return precisao;
    }

    public void setPrecisao(int precisao) {
        this.precisao = precisao;
    }

    public String getSigla() {
        return sigla;
    }

    public void setSigla(String sigla) {
        this.sigla = sigla;
```

```
35      }
}
```

Listagem 6.27: Usuario.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

public class Usuario extends Informacao {
//     CODIGOUSUARIO, USUARIO, SENHA, SUPERVISOR, GERENTE, CADASTROS,
    DADOS, RELATORIOS, NOME from usuarios t
     private String usuario;
     private String senha;
     private int supervisor;
     private int gerente;
     private int cadastros;
     private int dados;
     private int relatorios;

     public Usuario(int cd, String nome, String usr, String pwd, int sup,
         int ger, int cad, int dados, int rel, int flCubo){
          this.codigo = cd;
          this.nome = nome;
          this.usuario = usr;
          this.supervisor = sup;
          this.gerente = ger;
          this.cadastros = cad;
          this.dados = dados;
          this.relatorios = rel;
          this.flCubo = flCubo;
     }

     public int getCadastros() {
          return cadastros;
     }
```

```
    public void setCadastros(int cadastros) {
        this.cadastros = cadastros;
35  }

    public int getDados() {
        return dados;
    }
40
    public void setDados(int dados) {
        this.dados = dados;
    }

45  public int getGerente() {
        return gerente;
    }

    public void setGerente(int gerente) {
50      this.gerente = gerente;
    }

    public int getRelatorios() {
        return relatorios;
55  }

    public void setRelatorios(int relatorios) {
        this.relatorios = relatorios;
    }
60
    public String getSenha() {
        return senha;
    }

65  public void setSenha(String senha) {
        this.senha = senha;
    }

    public int getSupervisor() {
70      return supervisor;
```

```
    }

    public void setSupervisor(int supervisor) {
        this.supervisor = supervisor;
    }

    public String getUsuario() {
        return usuario;
    }

    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }
}
```

Listagem 6.28: UsuarioArea.java

```
/*
 * Binara Informática
 * Created on 13/10/2005
 */
package ufsc.ine.modelo;

public class UsuarioArea {
    private Usuario usuario;

    private Area area;

    private int flCubo;

    public UsuarioArea(Area area, Usuario usr, int flCubo) {
        this.area = area;
        this.usuario = usr;
        this.flCubo = flCubo;
    }

    public Area getArea() {
        return area;
    }
```

```
    public void setArea(Area area) {
        this.area = area;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public int getFlCubo() {
        return flCubo;
    }

    public void setFlCubo(int flCubo) {
        this.flCubo = flCubo;
    }
}
```

Listagem 6.29: Valor.java

```
/*
 * Binara Informática
 * Created on 03/11/2005
 */
package ufsc.ine.modelo;

import java.util.Date;

public class Valor {
    private IndicadorArea indicadorArea;

    private Date data;

    private float metaTendencial;

```

```
    private float metaOtimista;

    private float metaPessimista;

    private float padraoTendencial;

    private float padraoOtimista;

    private float padraoPessimista;

    private float realizado;

    public Date getData() {
        return data;
    }

    public void setData(Date data) {
        this.data = data;
    }

    public IndicadorArea getIndicadorArea() {
        return indicadorArea;
    }

    public void setIndicadorArea(IndicadorArea indicadorArea) {
        this.indicadorArea = indicadorArea;
    }

    public float getRealizado() {
        return realizado;
    }

    public void setRealizado(float realizado) {
        this.realizado = realizado;
    }

    public float getMetaOtimista() {
        return metaOtimista;
```

```
    }

    public void setMetaOtimista(float metaOtimista) {
        this.metaOtimista = metaOtimista;
    }

    public float getMetaPessimista() {
        return metaPessimista;
    }

    public void setMetaPessimista(float metaPessimista) {
        this.metaPessimista = metaPessimista;
    }

    public float getMetaTendencial() {
        return metaTendencial;
    }

    public void setMetaTendencial(float metaTendencial) {
        this.metaTendencial = metaTendencial;
    }

    public float getPadraoOtimista() {
        return padraoOtimista;
    }

    public void setPadraoOtimista(float padraoOtimista) {
        this.padraoOtimista = padraoOtimista;
    }

    public float getPadraoPessimista() {
        return padraoPessimista;
    }

    public void setPadraoPessimista(float padraoPessimista) {
        this.padraoPessimista = padraoPessimista;
    }
```

```
    public float getPadraoTendencial() {
        return padraoTendencial;
    }

95

    public void setPadraoTendencial(float padraoTendencial) {
        this.padraoTendencial = padraoTendencial;
    }
}
```

Listagem 6.30: GeradorChave.java

```
/*
 * Binara Informática
 * Created on 01/02/2006
 */
5 package ufsc.ine.util;


public class GeradorChave {

10      public Integer geraChave(Integer a){
            return a.hashCode();
        }

        public Integer geraChave(Integer a, Integer b){
15          String chave = a.toString() + "X" + b.toString();
            return chave.hashCode();
        }

        public Integer geraChave(Integer a, Integer b, Integer c){
20          String chave = a.toString() + "X" + b.toString() + "X" + c.
                toString();
            return chave.hashCode();
        }

        public Integer geraChave(Integer a, Integer b, Integer c, Integer d){
25          String chave = a.toString() + "X" + b.toString() + "X" + c.
                toString() + "X" + d.toString();
            return chave.hashCode();
```

```
    }

    public Integer geraChave(Integer a, Integer b, Integer c, Integer d,
        Integer e){
        String chave = a.toString() + "X" + b.toString() + "X" + c.
            toString() + "X" + d.toString() + "X" + e.toString();
        return chave.hashCode();
    }

    public Integer geraChave(Integer a, Integer b, Integer c, Integer d,
        String f){
        String chave = a.toString() + "X" + b.toString() + "X" + c.
            toString() + "X" + d.toString() + "X" + f;
        return chave.hashCode();
    }

}
```

Listagem 6.31: ControladorTest.java

```
/*
 * Binara Informática
 * Created on 22/02/2006
 */
package ufsc.ine.controle;

import junit.framework.TestCase;

public class ControladorTest extends TestCase {
    Controlador controlador;

    public void setUp(){
        controlador = new Controlador();
    }

    public void testTimerLeitura(){
        assertEquals(-1, controlador.auditoria.getInicioLeitura().
            compareTo(controlador.auditoria.getFimLeitura()));
    }
```

```
    public void testTotalRegistros(){
        assertEquals(175,controlador.auditoria.getRegistrosLidos());
    }

    public void testTotalRegistrosNovosLidos(){
        assertEquals(140, controlador.auditoria.getRegistrosNovosLidos())
            ;
    }
}
```

Listagem 6.32: DAOMigradorOracleDestinoTest.java

```
/*
 * Binara Informática
 * Created on 22/02/2006
 */
package ufsc.ine.dao;
import java.sql.SQLException;
import junit.framework.TestCase;
import org.apache.log4j.Logger;
import ufsc.ine.log.GeradorLogger;
import ufsc.ine.modelo.Auditoria;
import ufsc.ine.modelo.ModeloBSC;
public class DAOMigradorOracleDestinoTest extends TestCase
{
    Logger log = GeradorLogger.getLogger(this.getClass().getName());
    DAOMigradorDestino dao = DAOFactory.getDAOMigradorDestino();
    DAOMigradorOrigem daoOrigem = DAOFactory.getDAOMigradorOrigem();
    ModeloBSC modelo;
    Auditoria auditoria;
    protected void setUp()
    {
        dao.conecta();
        try
        {
            dao.conn.setAutoCommit(false);
        } catch (SQLException e)
        {
```

```
            e.printStackTrace();
        }
        modelo = daoOrigem.getModeloBSC();
        auditoria = new Auditoria();
        auditoria.setCodigo(dao.getCodigoAuditoria());
        modelo.setAuditoria(auditoria);
        dao.setModeloBSC(modelo);
    }
    protected void tearDown()
    {
        try
        {
            dao.conn.rollback();
        } catch (SQLException e)
        {
            e.printStackTrace();
        }
        dao.desconecta();
    }

    public void testGravaAreas()
    {
        dao.gravaAreas();
        assertEquals("Total de áreas no destino: ", 4, dao.
            getTotalLinhasTabela("AREA"));
    }
    public void testGravaPerspectivas(){
        dao.gravaPerspectivas();
        assertEquals("Total de Perspectivas",4,dao.getTotalLinhasTabela("
            PERSPECTIVA"));
    }
    public void testGravaObjetivo(){
        dao.gravaObjetivos();
        assertEquals("Total de objetivos",13,dao.getTotalLinhasTabela("
            OBJETIVO"));
    }
    public void testGravaUsuario(){
        dao.gravaUsuarios();
```

```
        assertEquals("Total de usuarios na base:", 2,dao.
            getTotalLinhasTabela("USUARIO"));
    }
    public void testGravaUsuariosArea(){
        dao.gravaAreas();
        dao.gravaUsuarios();
        dao.gravaUsuariosArea();
        assertEquals("Total de USUARIO_AREA",2,dao.getTotalLinhasTabela("
            USUARIO_AREA"));
    }
    public void testGetCodigoAuditoria()
    {
        System.out.println(dao.getCodigoAuditoria());
        assertTrue(dao.getCodigoAuditoria() > 0);
    }
}
```

Listagem 6.33: DAOMigradorOracleOrigemTest.java

```
package ufsc.ine.dao;
import java.util.Enumeration;
import java.util.Hashtable;
import junit.framework.TestCase;
import org.apache.log4j.Logger;
import ufsc.ine.log.GeradorLogger;
import ufsc.ine.modelo.Area;
import ufsc.ine.modelo.FrequenciaColeta;
import ufsc.ine.modelo.Indicador;
import ufsc.ine.modelo.IndicadorArea;
import ufsc.ine.modelo.ModeloBSC;
import ufsc.ine.modelo.Objetivo;
import ufsc.ine.modelo.ObjetivoTema;
import ufsc.ine.modelo.Perspectiva;
import ufsc.ine.modelo.UnidadeMedida;
import ufsc.ine.modelo.Usuario;
import ufsc.ine.modelo.UsuarioArea;
import ufsc.ine.modelo.Valor;
import ufsc.ine.util.GeradorChave;
public class DAOMigradorOracleOrigemTest extends TestCase
```

```
{
    Logger log = GeradorLogger.getLogger(this.getClass().getName());
    DAOMigradorOrigem dao = DAOFactory.getDAOMigradorOrigem();
    GeradorChave gerador;
    protected void setUp()
    {
        dao.conecta();
        gerador = new GeradorChave();
    }
    protected void tearDown()
    {
        dao.desconecta();
    }
    public void testGetAreas()
    {
        Hashtable<Integer, Area> areas = dao.getAreas();
        assertEquals("Número de áreas: ", 4, areas.size());
        assertEquals("Area Binara:", "Binara", areas.get(51).getNome());
        assertEquals("Area Desenvolvimento de Software:", "
            Desenvolvimento de Software", areas.get(52).getNome());
        assertEquals("Area Qualidade:", "Qualidade", areas.get(70).
            getNome());
        assertEquals("Area Planejamento:", "Planejamento", areas.get(71).
            getNome());
    }
    public void testGetFrequencias()
    {
        Hashtable<Integer, FrequenciaColeta> freqs = dao.
            getFrequenciasColeta();
        assertEquals("Número de frequencias", 9, freqs.size());
        assertEquals("Frequencia Anual", "Anual", freqs.get(1).getNome())
            ;
        assertEquals("Frequencia Semestral", "Semestral", freqs.get(5).
            getNome());
        assertEquals("Frequencia Quadrimestral", "Quadrimestral", freqs.
            get(8).getNome());
    }
    public void testGetIndicadores()
```

```
    {
        Hashtable<Integer, Indicador> inds = dao.getIndicadores();
        assertEquals("Número de Indicadores", 38, inds.size());
        assertEquals("Indicador Clientes Satisfeitos", "Clientes
            Satisfeitos", inds.get(83).getNome());
        assertEquals("Indicador Op inião Neutra do Público", "Opinião
            Neutra do Público", inds.get(95).getNome());
        assertEquals("Indicador Integração de Sistemas", "Integração de
            Sistemas", inds.get(50).getNome());
    }
    public void testGetObjetivos()
    {
        Hashtable<Integer, Objetivo> objs = dao.getObjetivos();
        assertEquals("Número de objetivos", 13, objs.size());
        assertEquals("Objetivo Aumentar Receitas", "Aumentar Receitas",
            objs.get(70).getNome());
        assertEquals("Objetivo Aumentar Qualidade do Produto", "Aumentar
            Qualidade do Produto", objs.get(79).getNome());
        assertEquals("Objetivo Tornar a Marca Binara Referência", "Tornar
             a Marca Binara Referência", objs.get(81).getNome());
        assertEquals("Objetivo Aumentar Satisfação de Clientes", "
            Aumentar Satisfação de Clientes", objs.get(73).getNome());
    }
    public void testGetPerspectivas()
    {
        Hashtable<Integer, Perspectiva> persp = dao.getPerspectivas();
        assertEquals("Número de perspectivas", 4, persp.size());
        assertEquals("Perspectiva Aprendizado e Crescimento", "
            Aprendizado e Crescimento", persp.get(50).getNome());
        assertEquals("Ordem 4", 4, persp.get(50).getOrdem());
        assertEquals("Perspectiva Financeira", "Financeira", persp.get
            (51).getNome());
        assertEquals("Ordem 1", 1, persp.get(51).getOrdem());
        assertEquals("Perspectiva Processos Internos", "Processos
            Internos", persp.get(70).getNome());
        assertEquals("Ordem ", 3, persp.get(70).getOrdem());
        assertEquals("Perspectiva Clientes", "Clientes", persp.get(71).
            getNome());
```

```
        assertEquals("Ordem ", 2, persp.get(71).getOrdem());
    }
    public void testGetUnidadesMedida()
    {
        Hashtable<Integer, UnidadeMedida> unds = dao.getUnidadesMedida();
        assertEquals("Número de unidades de medida", 10, unds.size());
        assertEquals("Unidade Conceito", "Conceito", unds.get(1).getNome
            ());
        assertEquals("Precisao 0 ", 0, unds.get(1).getPrecisao());
        assertEquals("Sigla Con ", "Con", unds.get(1).getSigla());
        assertEquals("Unidade Anos", "Anos", unds.get(7).getNome());
        assertEquals("Precisao 2", 2, unds.get(7).getPrecisao());
        assertEquals("Sigla a", "a", unds.get(7).getSigla());
    }
    public void testGetUsuarios()
    {
        Hashtable<Integer, Usuario> usrs = dao.getUsuarios();
        assertEquals("Número de usuarios", 2, usrs.size());
        assertEquals("Usuario supervisor ", "supervisor", usrs.get(1).
            getUsuario());
    }
    public void testGetModeloBSC()
    {
        ModeloBSC modelo = dao.getModeloBSC();
        assertNotNull(modelo);
        assertEquals("Número de áreas: ", 4, modelo.getAreas().size());
        assertEquals("Número de cenários", 25, modelo.getCenarios().size
            ());
        assertEquals("Número de frequencias", 9, modelo.
            getFrequenciasColeta().size());
        assertEquals("Número de Indicadores", 38, modelo.getIndicadores()
            .size());
        assertEquals("Número de objetivos", 13, modelo.getObjetivos().
            size());
        assertEquals("Número de perspectivas", 4, modelo.getPerspectivas
            ().size());
        assertEquals("Número de unidades de medida", 10, modelo.
            getUnidadesMedida().size());
```

```
        assertEquals("Número de usuarios", 2, modelo.getUsuarios().size()
            );
    }
    public void testGetIndicadoresAreas()
    {
        ModeloBSC modelo = dao.getModeloBSC();
        Hashtable<Integer, IndicadorArea> inds = modelo.
            getIndicadoresAreas();
        assertEquals("Número de IndicadorArea", 39, inds.size());
        assertNotNull(inds.get(gerador.geraChave(52, 50, 75, 102)));
        assertEquals(5, inds.get(gerador.geraChave(52, 50, 75, 102)).
            getFrequenciaColeta().getCodigo());
        assertEquals(5, inds.get(gerador.geraChave(52, 50, 75, 102)).
            getUnidadeMedida().getCodigo());
        assertEquals("bsc", inds.get(gerador.geraChave(52, 50, 75, 102)).
            getCategoria());
        assertEquals(5, inds.get(gerador.geraChave(52, 71, 73, 83)).
            getFrequenciaColeta().getCodigo());
        assertEquals(3, inds.get(gerador.geraChave(52, 71, 73, 83)).
            getUnidadeMedida().getCodigo());
        assertEquals("bsc", inds.get(gerador.geraChave(52, 71, 73, 83)).
            getCategoria());
        int totalIndicadorResultado = 0;
        for (Enumeration<IndicadorArea> e = inds.elements(); e.
            hasMoreElements();)
        {
            if (e.nextElement().getIndicadorResultado())
            {
                totalIndicadorResultado++;
            }
        }
        assertEquals("Total de indicadores de resultado", 18,
            totalIndicadorResultado);
    }
    public void testGetUsuariosArea()
    {
        ModeloBSC modelo = dao.getModeloBSC();
        Hashtable<Integer, UsuarioArea> usrsArea = modelo.
```

```
            getUsuariosAreas();
        assertEquals("Número de UsuariosArea", 2, usrsArea.size());
        assertNotNull(usrsArea.get(gerador.geraChave(2, 51)));
        assertNotNull(usrsArea.get(gerador.geraChave(2, 70)));
    }
    public void testGetObjetivosTemas()
    {
        ModeloBSC modelo = dao.getModeloBSC();
        Hashtable<Integer, ObjetivoTema> objTemas = modelo.
            getObjetivosTemas();
        assertEquals("Quantidade de objetivos temas", 14, objTemas.size()
            );
        assertNotNull(objTemas.get(gerador.geraChave(52, 51, 70)));
        assertNotNull(objTemas.get(gerador.geraChave(52, 51, 72)));
        assertNotNull(objTemas.get(gerador.geraChave(52, 71, 78)));
        assertNotNull(objTemas.get(gerador.geraChave(52, 70, 82)));
        assertEquals("Data Inicio 1/1/2006", "2006-01-01", objTemas.get(
            gerador.geraChave(52, 50, 75)).getDataInicio().toString());
        assertEquals("Data Fim 31/12/2008", "2008-12-31", objTemas.get(
            gerador.geraChave(52, 50, 75)).getDataFim().toString());
    }
    public void testGetValores()
    {
        ModeloBSC modelo = dao.getModeloBSC();
        Hashtable<Integer, Valor> valores = modelo.getValores();
        assertNotNull(valores);
        assertEquals("Total de valores", 15, valores.size());
        assertEquals("Realizado número clientes abc3", 4.0f, valores.get(
            gerador.geraChave(52, 71, 78, 81, "2006-01-01")).getRealizado
            ());
        assertEquals("Meta Tendencial número clientes abc3", 10.0f,
            valores.get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
            getMetaTendencial());
        assertEquals("Meta Pessimista número clientes abc3", 8.0f,
            valores.get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
            getMetaPessimista());
        assertEquals("Meta Otimista número clientes abc3", 15.0f, valores
            .get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
```

```
            getMetaOtimista());
        assertEquals("Padrao Tendencial número clientes abc3", 5.0f,
            valores.get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
            getPadraoTendencial());
        assertEquals("Padrao Pessimista número clientes abc3", 3.0f,
            valores.get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
            getPadraoPessimista());
        assertEquals("Padrao Otimista número clientes abc3", 8.0f,
            valores.get(gerador.geraChave(52, 71, 78, 81, "2006-01-01")).
            getPadraoOtimista());
        assertEquals("Realizado Lucratividade por Cliente", 82.0f,
            valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")).
            getRealizado());
        assertEquals("Meta Tendencial  Lucratividade por Cliente", 90.0f,
             valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")).
            getMetaTendencial());
        assertEquals("Meta Pessimista  Lucratividade por Cliente", 0.0f,
            valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")).
            getMetaPessimista());
        assertEquals("Meta Otimista  Lucratividade por Cliente", 0.0f,
            valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")).
            getMetaOtimista());
        assertEquals("Padrao Tendencial  Lucratividade por Cliente", 80.0
            f, valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")
            ).getPadraoTendencial());
        assertEquals("Padrao Pessimista  Lucratividade por Cliente", 0.0f
            , valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01"))
            .getPadraoPessimista());
        assertEquals("Padrao Otimista  Lucratividade por Cliente", 0.0f,
            valores.get(gerador.geraChave(52, 51, 72, 78, "2006-01-01")).
            getPadraoOtimista());
    }
}
```

Listagem 6.34: AuditoriaTest.java

```
/*
 * Binara Informática
 * Created on 21/02/2006
```

```
 */
package ufsc.ine.modelo;

import java.sql.Timestamp;
import java.util.GregorianCalendar;

import junit.framework.TestCase;

public class AuditoriaTest extends TestCase {

    Auditoria auditoria;
    public void setUp(){
        auditoria = new Auditoria();
    }
    public void testTimers(){
        auditoria.setInicioLeitura(new Timestamp(new GregorianCalendar().
            getTimeInMillis()));
        auditoria.setInicioEscrita(new Timestamp(new GregorianCalendar().
            getTimeInMillis()));
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        auditoria.setFimLeitura(new Timestamp(new GregorianCalendar().
            getTimeInMillis()));
        auditoria.setFimEscrita(new Timestamp(new GregorianCalendar().
            getTimeInMillis()));
        assertEquals(-1, auditoria.getInicioLeitura().compareTo(auditoria
            .getFimLeitura()));
        assertEquals(-1, auditoria.getInicioEscrita().compareTo(auditoria
            .getFimEscrita()));
    }
}
```

Listagem 6.35: InformacaoTest.java

```
/*
 * Binara Informática
```

```
 * Created on 01/03/2006
 */
package ufsc.ine.modelo;
import junit.framework.TestCase;
public class InformacaoTest extends TestCase
{
    public void testEnum()
    {
        assertEquals(0,FlagCubo.FlCubo.MIGRADO.ordinal());
        assertEquals(1,FlagCubo.FlCubo.NOVO.ordinal());
        assertEquals(2,FlagCubo.FlCubo.ATUALIZADO.ordinal());
    }
}
```

Listagem 6.36: GeradorChaveTest.java

```
/*
 * Binara Informática
 * Created on 01/02/2006
 */
package ufsc.ine.util;

import junit.framework.TestCase;

public class GeradorChaveTest extends TestCase {

    public void testGeradorChave(){
        GeradorChave gerador = new GeradorChave();
        assertFalse(gerador.geraChave(11,2).equals(gerador.geraChave
            (1,12)));
        assertTrue(gerador.geraChave(11,2).equals(gerador.geraChave(11,2)
            ));
    }

}
```