

CENTRO UNIVERSITÁRIO DE BELO HORIZONTE
MOISÉS HENRIQUE RAMOS PEREIRA

**APLICAÇÃO DE METODOLOGIA PARA
IMPLEMENTAÇÃO DE MODELOS DIMENSIONAIS EM
BANCO DE DADOS ORIENTADO A OBJETOS**

Belo Horizonte
2009

MOISÉS HENRIQUE RAMOS PEREIRA

**APLICAÇÃO DE METODOLOGIA PARA
IMPLEMENTAÇÃO DE MODELOS DIMENSIONAIS EM
BANCO DE DADOS ORIENTADO A OBJETOS**

Área de concentração: Banco de Dados.

Orientador: Professor Antonio da Mota
Moura Junior.

**Belo Horizonte
2009**

Dedico à minha namorada Luanda, grande incentivadora para a realização deste trabalho. À minha família, pessoas muito especiais na minha vida que contribuíram, passo a passo, na formação do meu caráter. Ao meu orientador pelos conselhos e por me orientar exemplarmente em meus interesses acadêmicos. Aos meus amigos que conquistei na faculdade pela convivência e pelo apoio constante.

RESUMO

Este trabalho consiste em aplicar parte da metodologia proposta por Borba (2006) para implantação de modelos dimensionais de ambientes de Data Warehouses (DW) em um ambiente de desenvolvimento integrado e orientado a objetos. A metodologia em que esse trabalho se baseia apresenta técnicas para definir o modelo de negócio da organização, gerar o modelo dimensional, representar o modelo gerado em um diagrama UML e, por fim, efetivar o mapeamento deste em um banco orientado a objetos pelo padrão ODMG. Conforme as adaptações realizadas para o estudo de caso de um modelo de negócio real, este trabalho desenvolve o mapeamento de um modelo dimensional já existente em diagrama de classes UML, implementando o banco de forma direta através de um framework de persistência de objetos integrado a uma linguagem orientada a objetos comercialmente difundida seguindo a padronização estabelecida pela OMG.

Palavras-chaves: data warehouse, modelo dimensional, modelagem, bancos de dados, metodologia, orientação a objetos, UML, BDOO.

LISTA DE ILUSTRAÇÕES

Figura 3.1.1	Exemplo de um negócio modelado.....	19
Figura 3.1.2	Exemplo de um modelo ER.....	20
Figura 3.2.1	A integração dos dados para uniformizar e facilitar o acesso.....	22
Figura 3.2.2	Diagrama de ciclo de vida dimensional de negócio - Modelo de Kimball.....	24
Figura 3.2.3	Cubo de dados para um modelo dimensional.....	25
Figura 3.2.4	Esboço de um modelo dimensional.....	27
Figura 3.3.1	Especialização das classes Barco e Carro em superclasse Veículo.....	31
Figura 3.3.2	Alguns tipos de relacionamentos entre classes na UML.....	33
Figura 3.3.3	Diagrama de classes baseado no modelo ER da figura 3.1.2.....	36
Figura 4.1	Resumo da metodologia de Borba e ordem de execução das etapas.....	39
Figura 4.1.1	Forma de armazenamento de objetos de um SGBD relacional e o db4o.....	41
Figura 4.2.1	Modelo dimensional CSB.....	42
Figura 4.3.1	Diagrama de classes CSB.....	46
Figura 4.4.1	Código Java parcial para a classe de dimensão CDAgrupamento.....	48
Figura 4.5.1	Esboço do modelo ER da base operacional de teste.....	53
Figura 4.5.2	Conjunto de resposta gerado no teste de execução.....	54

LISTA DE TABELAS

Tabela 3.2.1	Comparação entre modelo ER e modelo Dimensional.....	28
--------------	--	----

LISTA DE SIGLAS

API	Application Programming Interface / Interface de Programação de Aplicativos
BDOO	Banco de Dados Orientado a Objetos
BI	Business Intelligence / Inteligência de Negócio
DW	Data Warehouse / Armazém de Dados
ER	Entidade-Relacionamento
ETL	Extract-Transformation-Load / Extração, Transformação e Carga
JDBC	Java Database Connectivity / Conectividade de Banco de Dados via Java
OCL	Object Constraint Language / Linguagem de Restrição de Objetos
OID	Object Identified / Identificador de Objeto
OLAP	On-Line Analytical Processing / Processamento Analítico On-line
OO	Orientação a Objetos
RUP	Rational Unified Process / Processo Unificado da Rational
SGBD	Sistema de Gerenciamento de Bancos de Dados
SGBDOO	Sistema de Gerenciamento de Bancos de Dados Orientados a Objetos
SGBDOR	Sistema de Gerenciamento de Bancos de dados Objeto-Relacionais
SQL	Structured Query Language / Linguagem de Consulta Estruturada
UML	Unified Modeling Language / Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO	11
1.1 Contextualização do problema	11
1.2 Estrutura do trabalho	13
2 REVISÃO DA LITERATURA	14
3 MODELOS DE DADOS E A ORIENTAÇÃO A OBJETOS	17
3.1 Modelo de Dados Entidade-Relacionamento.....	18
3.2 Data Warehouse e o Modelo Dimensional.....	21
3.3 UML e a Modelagem Orientada a Objetos.....	28
4 MODELO DIMENSIONAL EM AMBIENTE DE PERSISTÊNCIA DE OBJETOS	38
4.1 As ferramentas utilizadas.....	40
4.2 Modelo dimensional utilizado	41
4.3 Mapeamento do modelo dimensional em UML.....	43
4.4 Implementação do diagrama UML no BDOO	47
4.5 Implementação e Execução de Consultas.....	50
5 CONCLUSÃO.....	56
REFERÊNCIAS.....	59
APÊNDICES.....	63

1 INTRODUÇÃO

Neste trabalho são descritas e aplicadas algumas etapas da metodologia proposta no ano de 2006 por Sueli de Fátima Poppi Borba, na Universidade Federal de Santa Catarina, em tese de doutorado referente à modelagem e implementação de ambientes de Data Warehouse sob os conceitos do paradigma da Orientação a Objetos, adaptando-a em um modelo dimensional conhecido.

1.1 Contextualização do problema

Na era da sociedade do conhecimento, o sucesso das organizações está diretamente ligado ao acesso privilegiado à informação em tempo e em quantidade satisfatórios dentro da dinâmica de negócio (BORBA, 2006). Com a crescente necessidade de gerenciar e analisar o desempenho organizacional, favorecendo o processo de tomada de decisão, percebe-se no mercado diversas soluções de Data Warehouses (DW) que disponibilizam, de forma mais interessante para a empresa, as informações contidas em grandes volumes de dados distribuídos em diferentes sistemas. A construção do DW deve possibilitar acesso fácil às informações, apresentá-las de maneira consistente, ser adaptável e flexível a mudanças, promover segurança em relação à proteção das informações e funcionar como uma base para suporte a decisões (INMON, 2005).

Da mesma forma que as organizações precisam de agilidade para dispor das informações estratégicas existentes nos sistemas, é necessário que o processo de implementação do DW seja bem definido, rápido e com baixos custos, o que pode ser determinante dentro da conjuntura comercial. Esse é o problema apontado por Klein, Campos & Tanaka (1999), pois existe a carência por uma metodologia bem definida para a implantação de ambientes DW. Assim, alguns aspectos devem ser discutidos como, por exemplo, o suporte na manipulação de dados e custos de implementação, pois em ambientes DW é necessário modelar todos os agrupamentos de informações da organização, gerando assim grande quantidade de

dados e maior complexidade dos mesmos para o armazenamento físico. Neste momento, com o intuito de que essa implantação seja menos impactante, surge a necessidade do uso de bancos de dados orientados a objetos (BDOO).

A motivação para este trabalho, além daquela observada pela metodologia de Borba (2006), vem do pressuposto que a OO poderia promover agilidade de desenvolvimento de bancos de dados OO como ocorre na realidade do desenvolvimento de software atualmente. Dessa forma, seria possível implementar soluções de DW seguindo uma metodologia ou um processo de desenvolvimento que embarcasse este paradigma, aproveitando-se das tecnologias oferecidas por aplicações de modelagem OO para melhoria da documentação do DW como, por exemplo, a abstração oferecida pela UML frente ao cliente.

As etapas descritas, como toda a metodologia, estão baseadas na modelagem e extração de informações comerciais relevantes sob o conjunto de processos chamado de Data Warehouse (DW), atualmente usado o termo Business Intelligence (BI), e pelo paradigma conhecido como Orientação a Objetos (OO) para o desenvolvimento de sistemas. O conceito de DW foi proposto em meados dos anos 80 pelos idealizadores Willian Inmon e Ralph Kimball que descrevem os processos de análise de negócio para suprir as necessidades empresariais através de sistemas de apoio à tomada de decisão, permitindo que informações relevantes e estratégicas sejam obtidas dos próprios dados existentes nas organizações. A teoria OO criada na década de 60 por Kristen Nygaard e Ole-Johan Dahl tinha como objetivo oferecer uma nova visão de modelagem e programação, pois era notável que as pessoas pensavam em entidades e responsabilidades de negócio como um todo estruturado e encapsulado, inferindo o conceito de objeto.

Dessa forma, durante toda a história da filosofia computacional e do pensamento científico para a resolução de problemas de forma mais objetiva e clara, diversos modelos foram definidos para documentar implementações de sistemas e metodologias de desenvolvimento, considerando as incertezas do pensamento humano e as abstrações de uma realidade para consenso de todos. Sobre as necessidades de diferencial competitivo promovido pelas soluções de DW e as facilidades geradas pelo paradigma OO no desenvolvimento de aplicações, percebe-se a relevância em estudar formas de agregar os dois conceitos para extrair os resultados mais eficazes desta associação frente às necessidades de melhor custo-

benefício nas análises de negócio.

O interesse deste trabalho é aplicar a metodologia de Borba (2006) para mapear e implementar um determinado modelo dimensional com o uso de tecnologias altamente difundidas no mercado que utilizam o paradigma OO e, dessa forma, relatar altos ganhos de desempenho, versatilidade e baixos custos de desenvolvimento com o intuito de apontar as vantagens no uso de bancos de dados OO comercialmente em soluções BI.

1.2 Estrutura do trabalho

Este trabalho está dividido em cinco capítulos, incluindo o primeiro capítulo de introdução. O segundo capítulo aborda uma revisão da literatura, refletindo, de forma sucinta, as soluções e problemas existentes quanto a modelagem e implementação de banco de dados, mais especificamente ambientes de DW e o histórico da OO. O capítulo mostra, também, o cenário atual com alguns trabalhos relacionados à implantação de DW e modelos propostos.

O terceiro capítulo apresenta, de maneira mais abrangente e conceitual, os modelos de dados conhecidos, desde a abordagem ER para a modelagem conceitual de bancos de dados relacionais até os modelos envolvendo a OO, abordagem mais recente para ambientes de persistência. Além disso, técnicas e componentes da modelagem dimensional são descritos para exemplificar algumas metodologias propostas para o desenvolvimento de soluções DW.

Já o capítulo 4 é dedicado à descrição da metodologia de Borba (2006) aplicada, aprofundando as etapas de mapeamento do modelo dimensional utilizado para UML e sua implementação física por uma linguagem de programação OO. Este capítulo apresenta as empresas que disponibilizaram o modelo dimensional para o estudo de caso, as ferramentas utilizadas no trabalho e os conceitos definidos no próprio modelo dimensional fornecido.

No capítulo final existem algumas reflexões e conclusões extraídas deste trabalho, sugerindo assim algumas linhas de pesquisa para trabalhos futuros.

2 REVISÃO DA LITERATURA

Nas décadas de 80 e 90, o banco de dados era definido como uma única fonte de dados para todo o processamento das aplicações existentes no meio científico. Com a evolução tecnológica, o computador tornou-se extremamente aplicável no mercado comercial, suprimindo as instituições empresariais com dados que favoreciam o ambiente de negócio, transformando os recursos computacionais de simples instrumentos de cálculo para verdadeiras ferramentas de análise de negócio com alto poder competitivo (VIDOTTI, 2001). Neste momento, a comunidade científica e as organizações empresariais passam a perceber a necessidade por alguma metodologia bem definida para a modelagem de bancos de dados por causa dos problemas causados pela implementação direta e falta de planejamento. Proposta por Peter Chen (1976), o modelo e abordagem ER passa a vigorar, como ocorre até nos dias de hoje, como o padrão de modelagem e desenvolvimento de bancos de dados relacionais.

Borba (2006) lembra que, desde os anos 90, a necessidade de obtenção de dados relevantes, principalmente em tempo satisfatório imposto pela dinâmica da globalização, vêm se apresentando cada vez mais importante e determinante para o sucesso das organizações. Com a crescente necessidade de gerenciar e analisar o desempenho organizacional, favorecendo o processo de tomada de decisão, percebe-se no mercado diversas soluções de Data Warehouses (DW) que disponibilizam, de forma mais interessante para a empresa, as informações contidas em grandes volumes de dados (BORBA, 2006).

Em 1992, Inmon definia um DW como uma coleção de dados orientada por assunto, integrada, variável no tempo e não-volátil, usada no apoio aos processos de tomada de decisão gerenciais. Para ele, o DW é uma tecnologia voltada para o estudo de técnicas e ferramentas utilizadas em aplicações que envolvam análise intensiva de dados e integração de fontes de dados heterogêneas, de forma a prover flexibilidade e agilidade na gerência, manutenção e acesso a estes dados (INMON, 2005). De uma forma geral, o DW é um ambiente que obtém um tratamento mais adequado da informação e facilita o processo de tomada de decisão, pois proporciona consultas de dados históricos que teriam alto custo de

desempenho se realizadas por sistemas tradicionais de armazenamento.

Como qualquer solução tecnológica adotada, inicialmente não se dava uma importância emergencial à definição de um processo bem fundamentado para o desenvolvimento de DW. Conforme Klein, Campos & Tanaka (1999), não existe uma metodologia sólida para implantação de ambientes DW, tendo somente guias que direcionam os arquitetos para obtenção do melhor resultado alcançável, indicando então um dos principais problemas para o seu desenvolvimento. Para facilitar a implementação deste tipo de abordagem, Kimball (1998), como poucos autores que aprofundam em um modelo lógico, propõe nove passos simples para nortear a implementação de um DW, tendo como foco o modelo estrela. Ele descreve que a construção efetiva de um DW se inicia pela modelagem dimensional, uma técnica de projeto lógico de banco de dados que busca apresentar os dados em um formato que seja intuitivo e com alto desempenho para o usuário final.

Diferentemente na área de desenvolvimento de sistemas, onde o paradigma da orientação a objetos (OO) se firma cada vez mais desde esta época, é importante comentar que não existem muitas pesquisas do uso desse paradigma na implementação de bases DW.

Esse enfoque para a orientação a objetos, paradigma de modelagem e programação criada nos anos 60, baseia-se no fato de que os objetos oferecem uma abstração maior para a modelagem de dados extremamente complexos, pois eles representam melhor as entidades do mundo real incorporando atributos e responsabilidades daquela entidade (BOSCARIOLI et al, 2006). Como ocorre também na modelagem dimensional, é necessário encontrar todos os objetos que compõem o DW, sem contar que os requisitos do negócio podem mudar dentro da dinâmica de competição do mercado. Dessa forma, os objetos podem ser estendidos para atender às novas demandas de mercado. Segundo Freitas Júnior et al (2002), esses objetos podem ser atributos de uma dimensão, podem ser as próprias dimensões, as métricas ou até mesmo as tabelas fato do DW. Outra vantagem do modelo orientado a objetos percebe-se na possibilidade de utilizar o mesmo modelo no projeto conceitual, onde os objetos são construídos; no projeto lógico, estendendo-se esses objetos conforme a tecnologia; e no físico, quando detalhes da ferramenta utilizada na implementação são incorporados ao modelo utilizado.

Borba (2006) comenta que cada objeto possui um único identificador OID

na base de dados inteira em um banco de dados orientado a objetos (BDOO), enquanto que em base de dados relacionais, uma chave primária formada por um ou mais atributos identifica somente uma tupla em uma relação. O estado atual de um determinado objeto consiste no valor das suas propriedades, sejam seus próprios atributos ou os relacionamentos com outros objetos. Como os objetos representam entidades do mundo real mais fielmente, os valores de seus atributos podem ser muito complexos, podendo representar outros objetos. Já em bancos de dados relacionais ocorrem algumas incompatibilidades referentes a esse tipo de complexidade, pois existe uma grande separação entre os dados e suas operações ou responsabilidades (BOSCARIOLI et al, 2006).

Dessa forma, apesar do sucesso dos sistemas de gerenciamento de dados relacionais, Klein, Campos & Tanaka (1999) afirmam que a manipulação de informações complexas proporcionada pelas diversas aplicações, inclusive a tecnologia de DW, exigiram novas soluções de gerenciamento de dados como os sistemas de gerenciamento de banco de dados orientados a objetos (SGBDOO) e objeto-relacionais (SGBDOR).

Dentro da modelagem de um DW, o esquema estrela se projeta como a principal forma de representação da dimensionalidade de negócio e, ao introduzir as abstrações OO, esses sistemas de gerenciamento permitem que as tabelas fato e as dimensões sejam melhor representadas. Neste contexto surge a proposta de modelagem OO por Borba (2006) para bancos DW que aplica integralmente os conceitos da OO sem nenhum tipo de tecnologia intermediária.

Nos últimos anos ocorreram alguns usos de BDOO no mercado, como relata Rosemberg (2008). A Indra Sistemas, grupo espanhol líder em tecnologia, foi contratada recentemente para desenvolver um centro de controle do sistema de trens bala AVE da Espanha (ROSEMBERG, 2008). Neste projeto, um BDOO foi utilizado como a base de dados de tempo real para controlar o tráfego. O BDOO usado foi o db4o, um framework de persistência de objetos criado por volta do ano 2000, gerando uma capacidade de processamento em torno de 200 mil objetos por segundo no sistema da Indra com pouco uso de memória. De acordo com José Miguel Rubio Sánchez, gerente técnico da Indra, o maior benefício observado se mostrou na facilidade em trabalhar diretamente com objetos em consultas sem transformar os dados nos projetos complexos em Java.

3 MODELOS DE DADOS E A ORIENTAÇÃO A OBJETOS

Nas últimas décadas, a computação vem evoluindo no que se refere às metodologias e tecnologias de modelagem e armazenamento de dados. Uma dessas tecnologias que contribuíram para essa evolução são os diversos tipos de bancos de dados que armazenam grande quantidade de dados em um curto espaço de tempo, mas esta realidade passou a dificultar aos envolvidos identificar e analisar informações relevantes ali presentes para o negócio da organização. Dessa forma, o modelo de dados deve fornecer uma visão precisa e objetiva de como os dados serão armazenados para favorecer o entendimento de conceitos, especificações e regras durante o projeto de banco de dados (BORBA, 2006).

Conforme Vieira (2001), um modelo de dados é uma coleção de conceitos que podem ser usados para descrever um conjunto de dados e operações para manipular esses dados. Para Elmasri & Navathe (2002), os modelos de dados podem ser classificados em modelo conceitual, lógico e físico, conforme a etapa de desenvolvimento do projeto do banco em que o modelo é utilizado.

O modelo conceitual representa as entidades e seus relacionamentos conforme observadas e expostas no mundo real durante a fase de análise entre os envolvidos da organização, desconsiderando detalhes impostos pela tecnologia, metodologias ou dispositivos físicos. No modelo lógico, construído a partir do modelo conceitual, as entidades mapeadas são representadas conforme um padrão mais técnico e formal, considerando limitações tecnológicas e decisões de projeto conforme a visão do usuário do SGBD, mas ainda se abstém do ambiente físico onde os dados serão armazenados no computador. Já no modelo físico, detalha-se os tipos de campos, o acesso à memória e demais itens para o SGBD, pois os dados são representados conforme o ambiente físico para implementar as estruturas de armazenamento das ocorrências ou instâncias das entidades (COUGO, 1997).

Dentro de uma abordagem tradicional, um tipo de modelo muito usado na especificação de um banco de dados é o modelo Entidade-Relacionamento, geralmente utilizado no projeto conceitual de aplicações de um banco de dados. (ELMASRI & NAVATHE, 2005).

3.1 Modelo de Dados Entidade-Relacionamento

Proposto por Peter Chen em 1976, o modelo Entidade-Relacionamento (ER) é uma extensão do modelo conceitual que estuda os dados e descrições da realidade em um esquema de conceitos entendíveis pelo usuário final. Dessa forma, o modelo ER representa, sob uma forma estrutural simétrica, os dados em um diagrama que mapeia as entidades do negócio e os relacionamentos entre elas de forma a remover a redundância de dados (KIMBAL, 1998).

Para implementar um modelo ER sobre um determinado cenário de dados, é necessário identificar os tipos de entidades e seus relacionamentos, desenhar um diagrama ER a partir destes, identificar os atributos e valores, traduzir o diagrama ER em um diagrama estruturado de dados e projetar os formatos dos registros (CHEN *apud* BORBA, 2006).

No estudo de modelagens de dados, é necessário conhecer os conceitos de entidade, relacionamento, atributo e identificadores, elementos presentes na construção dos diagramas e importantes na construção de um banco de dados.

Chen (1976) define entidade como “(...) uma ‘coisa’ que pode ser claramente identificada. Uma determinada pessoa, empresa ou evento é um exemplo de uma entidade”, ou seja, entidade é todo objeto relevante encontrado na descrição do negócio modelado, no qual objeto aqui é um termo genérico e ainda não está associado à orientação a objeto, assunto que será tratado mais à frente.

Graficamente no diagrama ER, as entidades modeladas são representadas por um retângulo que, numa abordagem geral, significa todos os objetos referentes àquela entidade dos quais deseja-se armazenar informações. Caso seja preciso referenciar um objeto em particular, ou seja, um determinado conjunto de dados reais existentes no ambiente modelado, denota-se uma *ocorrência* ou *instância* de uma entidade (HEUSER, 1998).

Depois de identificadas todas as entidades, é necessário representar as informações que caracterizam essas entidades dentro do cenário de negócio. Conforme Heuser (1999), uma dessas características que pode ser armazenada é o relacionamento entre entidades no ambiente real. Percebe-se que relacionamento é uma determinada associação entre duas ou mais entidades como, por exemplo,

entre as entidades Professor e Disciplina de um determinado minimundo, podemos encontrar o relacionamento Leciona, pois a associação existente é de que Professor Leciona Disciplina. Na figura 3.1.1 temos um exemplo de um determinado negócio modelado onde foram identificadas três entidades (Cliente, Pedido e Produto) e dois relacionamentos (Faz e Contém) entre elas. As imagens de objetos e pessoas representam as respectivas instâncias das entidades.



Figura 3.1.1 – Exemplo de um negócio modelado.
Fonte: MACHADO & ABREU, 2004.

Os relacionamentos podem ser classificados em três tipos:

- um-para-um (1:1), onde uma instância de uma entidade está associada a uma instância da outra entidade;
- um-para-muitos (1:n), onde uma instância da primeira entidade pode estar associada a uma ou várias instâncias da segunda entidade;
- muitos-para-muitos (n:m), onde cada instância de uma entidade pode estar associada a várias instâncias da outra entidade e vice-versa, ou seja, pode ser entendido como um relacionamento 1:n bilateral pelo fato de que em ambos os sentidos de leitura existe um grau de um-para-muitos (MACHADO & ABREU, 2004).

No estudo de relacionamentos pelo modelo ER, encontramos também os conceitos de totalidade e parcialidade que exprimem, respectivamente, a obrigatoriedade e a não-obrigatoriedade da existência de pelo menos uma associação entre duas entidades. No diagrama ER os relacionamentos são

representados por um losango e os conceitos acima aparecem como uma linha de ligação dupla (totalidade) e uma linha simples (parcialidade) ligadas às entidades. Outra forma muito utilizada para se representar esse tipo de obrigatoriedade é descrever a cardinalidade mínima e máxima das entidades no relacionamento, colocando o número 0 para cardinalidade mínima opcional e o número 1 para ocorrências obrigatórias (HEUSER, 1998).

Outra propriedade das entidades que devem ser modeladas e armazenadas são os seus atributos, valores que definem e caracterizam uma determinada entidade. Para Chen (1976), “As informações sobre uma entidade ou um relacionamento é obtido através da observação e medição, expressas por um conjunto de pares atributo-valor”, ou seja, as entidades e relacionamentos podem ser compostos por atributos que, por sua vez, possuem valores em suas instâncias.

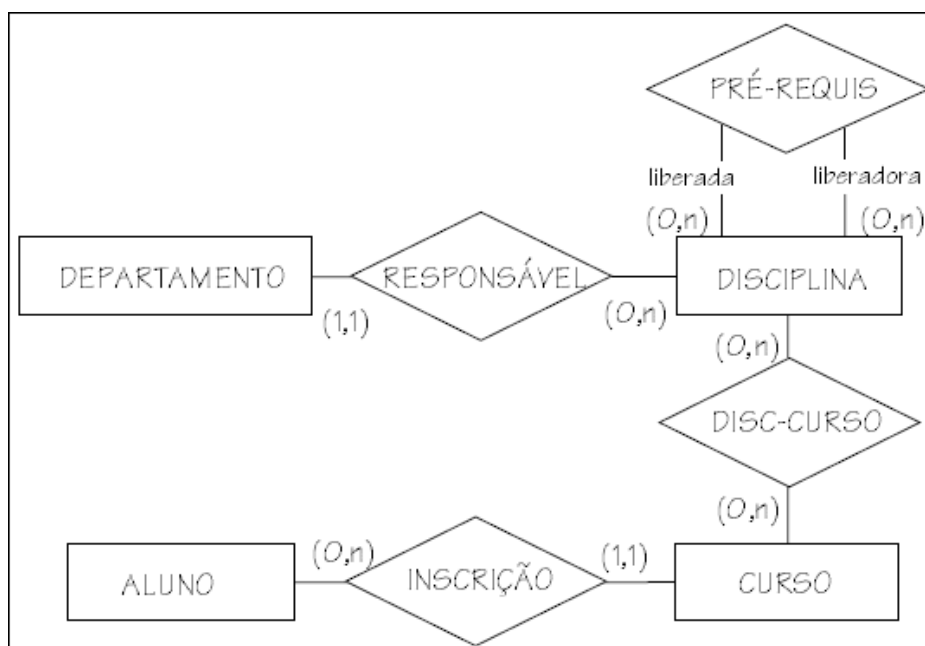


Figura 3.1.2 – Exemplo de um modelo ER.
Fonte: HEUSER, 1998.

Finalizando essa abordagem superficial dos conceitos envolvidos no modelo ER, encontramos os identificadores, atributos especiais existentes nas entidades modeladas, possibilitando que cada instância ou tupla seja única nas tabelas que representam as entidades. Esses identificadores são conhecidos como chaves primárias de banco de dados relacionais e relacionamentos n:m são identificados pelos atributos-chave das entidades envolvidas (BORBA, 2006).

3.2 Data Warehouse e o Modelo Dimensional

Nesta seção, serão apresentadas algumas definições referentes a Data Warehouse (DW) e alguns elementos importantes do modelo dimensional para uma discussão das vantagens de um mapeamento orientado a objetos.

Encontra-se na literatura a caracterização de DW como “(...) uma coleção de dados orientada a assuntos, integrada, não volátil e variável no tempo em suporte às decisões gerenciais” (INMON, 2005: 29). Borba (2006) complementa que um DW é uma importante ferramenta para análise e acesso mais global aos dados advindos de diversas bases de dados autônomas e heterogêneas, pois o DW possui uma cópia dos dados extraídos de um ou mais sistemas de produção na fase de carregamento (KIMBALL, 1998). O conceito de Data Mart é parecido com as definições acerca de um DW, mas focado sobre um determinado assunto, ou seja, Data Mart é um DW departamental que possui um assunto definido.

A implementação de um DW deve ser considerada um projeto de desenvolvimento e deve ser mantido pela organização que deseja melhorar o seu processo de decisão (PINHEIRO, 2002). Na abordagem de Kimball (2002), Data Marts são criados e, posteriormente, o DW é construído a partir da correlação entre eles, mas para Inmon (2005), o DW é construído para depois os assuntos serem identificados para Data Marts. Como não é foco de análise deste trabalho, o detalhamento dessas implementações não será abordado.

Como visto na sessão anterior, o modelo ER é amplamente usado na construção de bancos de dados relacionais que suportam a maioria das bases de dados operacionais das organizações atualmente. Essas bases de dados operacionais são baseadas nas aplicações da empresa, para obtenção de melhor desempenho conforme o sistema ao qual está associada (VIDOTI, 2001). Diferentemente, um DW é baseado em assunto, ou seja, os dados são organizados conforme os principais assuntos do negócio da empresa.

Os dados no DW não são deletados e nem atualizados livremente como nas bases operacionais, salvo na administração do processo de carga do DW, quando os dados dos aplicativos em produção são inseridos e, a partir daí, serão apenas consultados durante as demandas de negócio. Conforme Kimball (1998), o

primeiro estágio do DW onde estes dados são carregados para serem tratados em processos ETL (extract-transformation-load) é conhecida como *data staging area* e esse controle de carga (inserção) é feito fora do expediente das consultas analíticas. Essa propriedade de que os dados não são atualizados e deletados, inferindo a não-volatilidade ao DW, permite melhor performance do mesmo, pois o ambiente executará as transações analíticas sem perder tempo de processamento no controle de concorrência como acontece nos bancos de dados operacionais acessados por diversos usuários que podem executar diferentes tarefas sobre os mesmos dados.

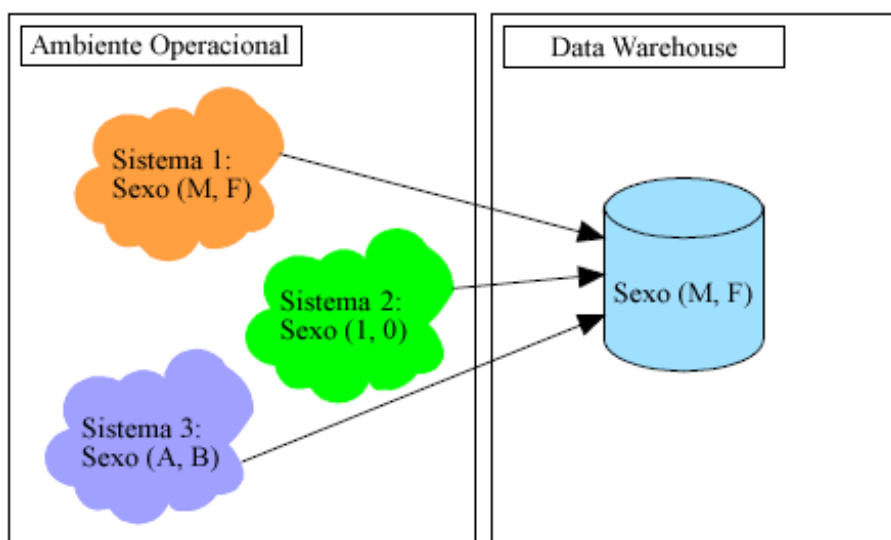


Figura 3.2.1 – A integração dos dados para uniformizar e facilitar o acesso.
Fonte: VIDOTTI, 2001.

Conforme a Figura 3.2.1 acima, a integridade dos dados é outra importante característica de um ambiente DW, pois os dados advindos das diversas fontes operacionais podem possuir diferentes formatos e valores, em conformidade às suas respectivas aplicações. Como não existe a exigência de uma padronização das informações no ambiente operacional, adota-se um padrão para os dados no DW, mantendo os mesmos integrados sob um único formato (VIDOTTI, 2001). Para Inmon (2005), a integração é o processo mais importante na manutenção de um DW e como os dados são alimentados a partir de múltiplas fontes, eles devem ser manipulados a fim de possuírem uma única imagem corporativa. Com isso, “os dados são inseridos no data warehouse de tal forma que as muitas incoerências no nível do aplicativo são desfeitas” (INMON, 2005: 31).

O DW também é variante no tempo, ou seja, os dados nesse ambiente representam uma faixa extensa de tempo, podendo armazenar, por exemplo, as

realidades e instâncias de um determinado dado concebido há vários anos. Já as bases operacionais de dados, frequentemente, armazenam somente a atual instância dos dados e, como podem ser alterados ou até mesmo deletados, quando uma mudança ocorre, perde-se a referência do antigo *status* em que o dado se encontrava. Conforme Vidotti (2001), essa característica do DW é garantida pela implementação de uma dimensão de tempo que, no momento em que os dados são inseridos no DW, receberá em seu atributo-chave o valor da data desta atualização, permitindo assim ter redundância dos dados que estão diferenciados somente pela data de alteração dos mesmos no ambiente operacional. O conceito de dimensão será apresentado durante a definição de modelagem dimensional.

O acesso ao DW fornece alto desempenho na obtenção de resposta em sistemas caros com alta transitividade de dados. Dessa forma, o DW é um tipo de projeto que enfoca “(...) a implementação de um processo, ferramentas e facilidades para se gerenciar e gerar informação completa, oportuna e compreensível para o processo decisório nas organizações” (BORBA, 2006).

Na modelagem de DW não é utilizado plenamente o conceito de projetos normalizados, como existente na abordagem ER (BORBA, 2006). Dessa forma, no projeto de DW, é modelada uma tabela central (fato) conectada a diversas outras tabelas que armazenam as dimensões do negócio dando um caráter estrela ao modelo. Geralmente, somente a tabela de fatos é normalizada, não necessariamente todas as tabelas como no modelo ER. Borba (2006) afirma que a comunidade científica ainda não direcionou um estudo concreto para a padronização de modelos conceituais para DW, incluindo a construção de modelos dimensionais.

Conforme Barbieri (2001), os tipos de modelos existentes para estruturas transacionais, como o modelo ER, são incompatíveis com as novas necessidades empresariais de competitividade, diferenciais de negócio e tomadas de decisão estratégicas, sendo necessária a formulação de novos modelos que abordem uma visão mais focada para o negócio das organizações e que, dessa forma, facilitem no desenvolvimento de ambientes DW. Como descrito no capítulo 2, ao longo dos anos, em consonância com as necessidades de mercado, foram publicadas diversas abordagens para implementação destes ambientes analíticos. A seguir são descritas duas metodologias que possuem muitos elementos em comum, mas seguem arquiteturas e paradigmas diferentes em diversas fases: o modelo de Kimball e o

modelo proposto por Borba em 2006.

Frente à realidade do mercado no gerenciamento de informações estratégicas, Kimball et al (1998) promoveu, desde meados dos anos 80, um modelo contendo as principais diretrizes de desenvolvimento do DW, distribuídas em 9 fases. Esse modelo afirma que os projetos do DW devem incidir sobre as necessidades do negócio e que os dados devem ser unidimensionais quando apresentados aos clientes. Cada projeto no DW deverá ter um ciclo finito com início e fim bem definidos. A sua primeira fase consiste do Planejamento de Projeto, seguida pelas fases de Definição dos Requisitos de Negócio, Design Técnico de Arquitetura, Seleção e Instalação de Produtos, Modelagem Dimensional, Design Físico, Design e Desenvolvimento da *Data Staging Area*, Especificação e Implementação da Aplicação Analítica, Implantação, Manutenção e Crescimento (KIMBALL, 2002). Este modelo foca o desenvolvimento do DW para um ambiente de bancos de dados relacionais.

A Figura 3.2.2 apresenta o diagrama de ciclo de vida do modelo de Kimball, indicando a distribuição das fases seqüenciais, dependências e fases concorrentes em um projeto de desenvolvimento de DW. Kimball & Ross (2002) alertam que o diagrama não reflete uma cronologia absoluta entre as fases e nem o verdadeiro tempo de duração de cada uma delas. É um modelo que pode ser considerado como um simples roteiro que facilita a equipe do projeto no desenvolvimento e que serve tanto para os projetos-assunto de Data Mart como para todo o ambiente DW.

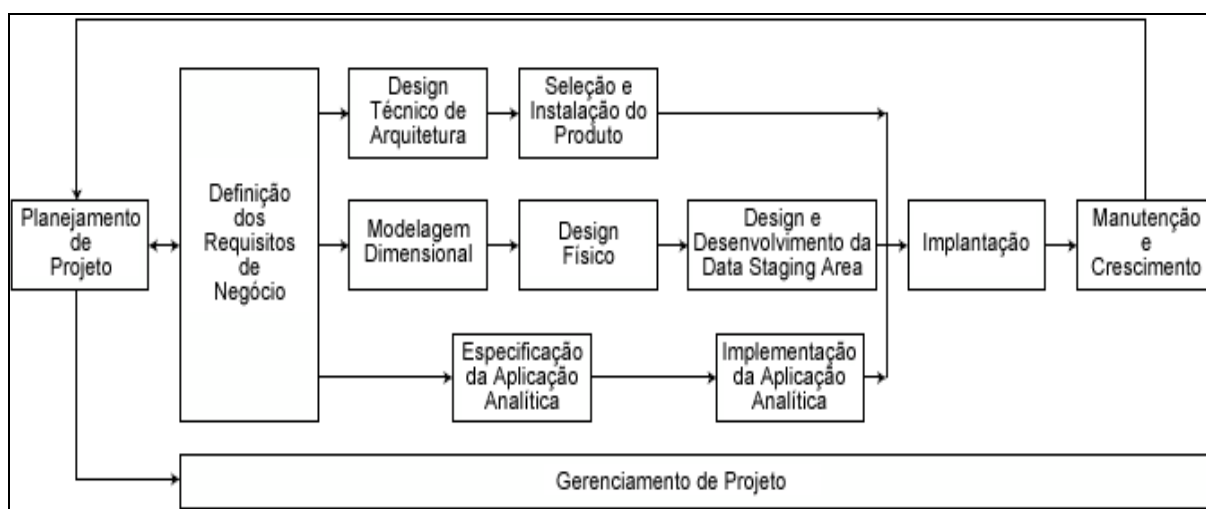


Figura 3.2.2 – Diagrama de ciclo de vida dimensional de negócio – Modelo de Kimball.
Fonte: KIMBALL, 2002: 332.

O modelo proposto por Borba (2006) promove a implementação do DW em BDOO, trabalhando com os conceitos do paradigma OO em várias de suas fases. Estes conceitos, quando representados pela UML, oferecem uma modelagem adequada das características de um DW, desde a fase de levantamento de requisitos até a implementação. Conforme Borba & Morales (2006), as 5 etapas que compõe este modelo são, nesta ordem: a Definição do Modelo do Negócio; Geração do Modelo Dimensional; Geração do Modelo Dimensional Representado pela UML; Mapeamento do Modelo UML para BDOO. Este modelo será devidamente aprofundado no próximo capítulo, pois trata-se de uma parte fundamental para a realização deste trabalho, principalmente no que se refere às fases de mapeamento do modelo dimensional em diagrama UML e deste para um ambiente OO.

Uma das técnicas de modelagem mais utilizadas atualmente para a implementação de um DW é a modelagem dimensional, um tipo de modelo lógico que reflete os requisitos de negócio em uma estrutura de cubo de dados, como mostrado na figura 3.2.3, um cubo de dados envolvendo as dimensões Região, Produto e Tempo. Conforme Kimball & Ross (2002), para construir o modelo dimensional é necessário identificar o processo de negócio a ser modelado, descrever o nível de granularidade e mapear as dimensões e fatos do negócio.

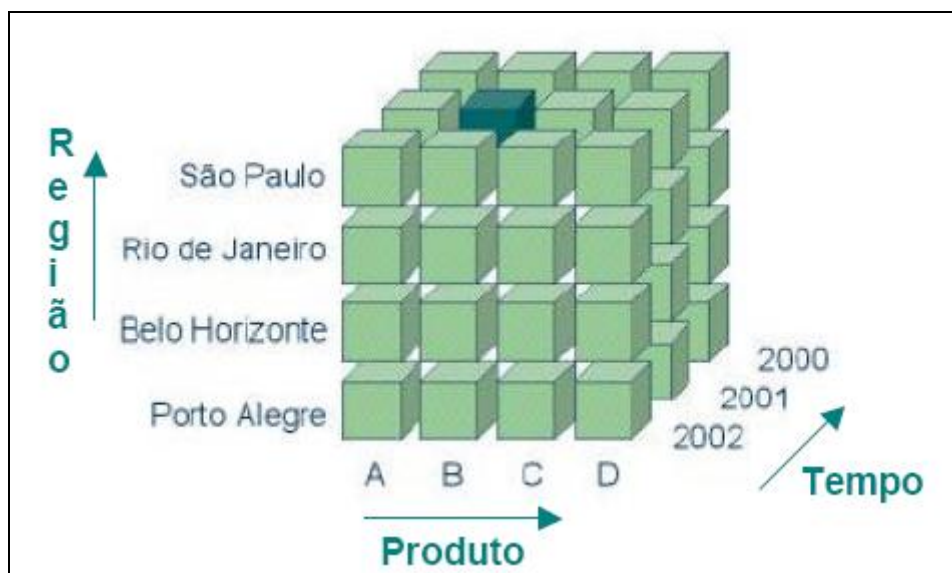


Figura 3.2.3 – Cubo de dados para um modelo dimensional.
Fonte: PINHEIRO, 2002.

Barbieri (2001) complementa que a modelagem dimensional representa a informação como interseções das várias dimensões do negócio para que o usuário

veja os dados conforme o entendimento que ele tem dessa realidade. O modelo dimensional possui uma estrutura mais assimétrica, diferentemente do modelo ER, mapeando uma tabela central chamada de tabela fato com diversos *joins* com outras tabelas secundárias, as dimensões (KIMBALL, 1998). No estudo da modelagem dimensional, mostra-se necessário entender os conceitos de tabelas fatos, dimensões, métricas e demais elementos que integram este tipo de modelagem.

Lopes & Oliveira (2007) definem tabelas fatos, também conhecidas como cubos de decisão ou tabelas de fatos, como representantes das transações ou ocorrências existentes no negócio. De forma mais específica, Kimball (1998) afirma que as tabelas de fatos armazenam as medições numéricas de desempenho extraídas dos fatos relevantes do negócio e as combinações entre as dimensões ali presentes. Para Borba (2006), em muitos casos, os fatos não são conhecidos e, geralmente, representam valores numéricos, também chamados de métricas, quantificados em medidas conforme uma realidade que é determinada pelos níveis de dimensão. Um exemplo de tabela fato está no modelo dimensional da figura 3.2.4, cuja tabela de fatos Vendas, conectada às dimensões Tempo, Produto e Loja, possui os totais diários de todos os produtos vendidos em uma loja, produzindo instâncias diferentes para diferentes combinações das dimensões (KIMBALL, 1998).

As dimensões representam os diversos tipos de visões que os usuários poderão ter do negócio. Este tipo de tabela armazena as descrições das dimensões de negócio modeladas, de preferência que sejam implementadas em diversos campos textuais, e são utilizadas como elementos condicionais que restringem ou formam os cabeçalhos das consultas analíticas do usuário (KIMBALL, 1998). Na figura 3.2.4, as tabelas Produto, Loja e Tempo são tabelas dimensionais e cada registro existente em cada uma delas representa, respectivamente, um produto, uma loja e tempo específicos. No caso da dimensão Tempo, como comentado anteriormente, é típica em um modelo dimensional para garantir a variância temporal do DW e armazenar as instâncias ao longo do tempo para consultas históricas.

No modelo dimensional, o mapeamento dos relacionamentos é feito sob uma lógica diferente da aplicada ao modelo ER. As dimensões fazem relacionamentos de 1:n com a tabela de fatos, cujo lado 1 está na dimensão e o lado n na tabela fato, representando que uma instância de uma determinada dimensão pode estar em uma ou mais instâncias dos fatos ocorridos no negócio. Em muitos

modelos dimensionais o lado n de relacionamentos é descrito por um asterisco, notação muito encontrada também em modelos orientados a objetos. Conforme Borba (2006), enquanto os relacionamentos entre as entidades são mapeados de forma explícita no modelo ER, no modelo dimensional os relacionamentos representam, implicitamente, as interseções entre a tabela de fatos e suas dimensões, dentro do ideal imposto pelo cubo de dados.

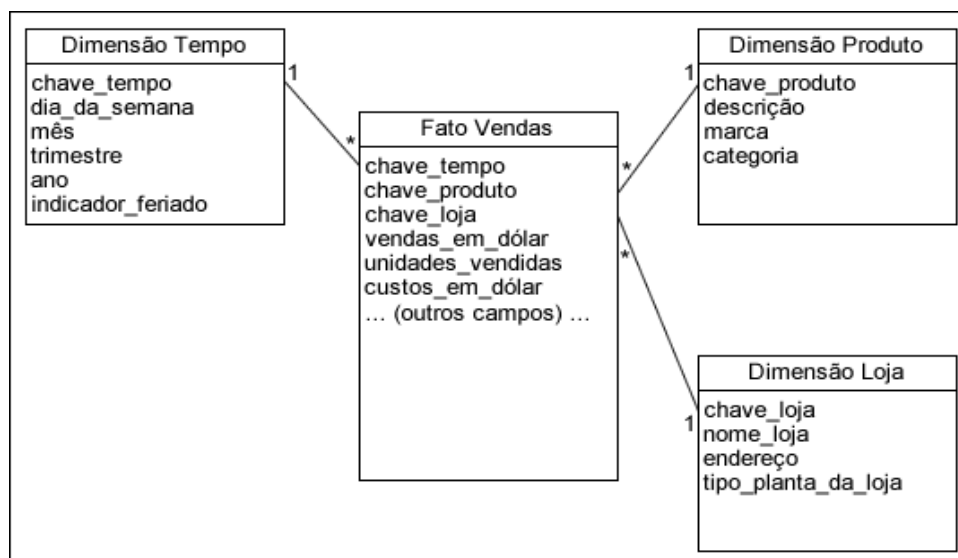


Figura 3.2.4 – Esboço de um modelo dimensional.
 Fonte: Adaptado de KIMBALL, 1998: 10.

Os campos das tabelas desenhadas para o modelo dimensional são extremamente importantes, pois, dependendo de como são classificados no modelo, eles refletirão alguma característica e informação importantes para os processos analíticos de tomada de decisão, seja para armazenar um fato do negócio ou uma descrição textual de uma dimensão. De acordo com Kimball (1998), essa classificação é denominada granularidade da tabela. Toda tabela de fatos possui como atributo-chave a combinação das chaves primárias das tabelas dimensionais as quais está ligada. Como na modelagem de bancos de dados relacionais, todo relacionamento n:m deve ser mapeado como uma nova tabela e, em ambientes DW, ela será uma tabela de fatos (KIMBALL, 1998).

A tabela 3.2.1 abaixo aponta as principais diferenças, sob um ponto de vista conceitual, entre os modelos ER e dimensional. Borba (2006) analisa que, mesmo com características análogas, estas duas modelagens focam em semânticas diferentes na abordagem e modelagem dos dados, como visto até agora. Enquanto

o modelo ER objetiva a modelagem das entidades existentes no negócio e como se relacionam entre si, a modelagem dimensional modela as dimensões de negócio envolvidas e os fatos ocorridos que as combinam, medindo o desempenho sob a análise de métricas.

Tabela 3.2.1: Comparação entre modelo ER e modelo Dimensional.

Fonte: Adaptado de BARBIERI, 2001.

Modelo ER	Modelo Dimensional
Modelo mais complexo	Padrão de estrutura mais fácil e intuitiva
Tabelas (Entidades) representam os dados e seus relacionamentos	Tabelas Fato e tabelas Dimensão
Todas as tabelas são normalizadas	Tabelas de Fatos são núcleos (normalizadas)
Criado para remover redundância	Mantém a redundância
As tabelas são indistintamente acessadas	Tabelas Dimensão são os pontos de entrada
Maior número de tabelas, dificultando os <i>joins</i>	Menos tabelas, facilitando os <i>joins</i>
Dificuldade de leitura pelos usuários finais	Leitura mais fácil pelos usuários finais

Existem diversos modelos dimensionais, mas os mais utilizados academicamente e comercialmente são o modelo Estrela e o modelo Floco de Neve, conhecidos na literatura, respectivamente, por *Star Schema* e *Snowflake Schema*. A estrutura do modelo Estrela consiste em diversas tabelas dimensionais conectadas exclusivamente a uma ou mais tabelas de fatos que se encontram no centro, possuindo um formato estrelar. Já no modelo Floco de Neve existem tabelas de dimensão conectadas a outras dimensões. A metodologia discutida no capítulo 4 aborda detalhes do mapeamento para cada um destes modelos, enfatizando mais o modelo Estrela, pois trata-se do formato do modelo dimensional utilizado.

3.3 UML e a Modelagem Orientada a Objetos

Criado na década de 60, o paradigma da Orientação a Objetos (OO) vem sendo cada vez mais difundido nos processos de desenvolvimento de software. Ele representa os dados em estruturas conhecidas como classes de objetos que, sob um ponto de vista de análise de projetos, implementam atributos e responsabilidades inerentes ao tipo de entidade modelada. Esse tipo de estrutura favorece a representação de dados muito complexos. Conforme Borba (2006), a modelagem

OO promove maior abstração dos dados a serem representados usando estruturas denominadas classes de objetos, encapsulando as diversas operações ou métodos que podem ser aplicadas a um determinado objeto.

Sob o âmbito do processo de construção de bancos de dados, a abordagem orientada a objetos é utilizada na modelagem conceitual de banco de dados orientados a objetos (BDOO), pois existe mapeamento direto, até mesmo sob a concepção do usuário, de um modelo de dados OO para sua implementação em um BDOO (BORBA, 2006). Para que a modelagem orientada a objetos fique bem fundamentada e mais fácil de ser entendida, é necessário conhecer os conceitos e objetivos do paradigma OO e os seus elementos essenciais, englobando as definições de classe, objeto, associação, herança, polimorfismo, encapsulamento, mensagens e os diagramas produzidos.

De forma análoga, Freitas Júnior et al (2002) define objeto como uma entidade no modelo ER que, além dos seus atributos, possuem também métodos. Os atributos de um objeto carregam mais características do que aqueles mapeados nas entidades ER. Um atributo, além de seu nome, também possui grau de visibilidade e o tipo de dado que ele armazena, dependendo do nível em que o projeto se encontra. Os métodos dos objetos também informam essas características em suas assinaturas, trechos para descrever se o método retorna valores, qual o tipo de retorno, nome da operação e parâmetros necessários. Um objeto, sob uma definição mais técnica, é uma instância de uma classe modelada para um sistema ou para um BDOO.

Conforme Elmasri & Navathe (2002), na implementação de sistemas, dados estruturados são denominados objetos transientes quando não são mantidos quando o programa termina sua execução, enquanto que, quando são armazenados permanentemente no BDOO, eles são chamados de objetos persistentes. Todo objeto possui um estado, comportamentos e um OID, que é um identificador único de um objeto em todo o ambiente físico onde o mesmo se encontra modelado.

Conforme Boscarioli (2006), o estado de um objeto reflete o valor da instância deste objeto, mais especificamente, o valor das suas propriedades. Fisicamente, o estado é formado pelas instâncias das variáveis que podem armazenar os dados e, por elas, pode ser modificado ao longo da existência do objeto no ambiente (BORBA, 2006). O comportamento de um objeto é definido pelo

conjunto de métodos definidos e que são executados quando invocados por operações de outros objetos. Essas operações podem alterar um ou mais atributos do objeto, alterando, assim, o seu estado.

A identidade é uma das principais propriedades dos objetos. Mesmo convergindo para o conceito de identificação única proposto pela chave primária às entidades do modelo ER, Cattell et al (2000) aponta algumas diferenças entre a abordagem ER e OO, pois enquanto a chave primária identifica uma única tupla em uma relação entre entidades, o OID identifica um objeto único em todo o banco de dados. É interessante que os OIDs sejam associados uma única vez a um objeto, ou seja, que os identificadores de objetos removidos não sejam reaproveitados. Dessa forma, alguns bancos de dados OO utilizam métodos *hashing* para que a recuperação de objetos seja feita de forma mais eficiente (VIEIRA, 2001).

As classes, conforme Elmasri & Navathe (2002), definem um tipo de objeto e as operações inerentes àquele tipo, ou seja, as classes são estruturas que implementam os mesmos atributos e operações para um grupo de objetos que serão instâncias destas classes. Além das estruturas pré-estabelecidas pelos sistemas de gerenciamento de BDOO e pelas linguagens OO, dependendo da aplicação, outras classes personalizadas podem ser construídas, segundo a necessidade da modelagem (BOSCARIOLI et al, 2006). As classes são estruturas abstratas de dados extremamente importantes na modelagem OO, pois através delas é possível mapear diversos tipos de associações e características como vistos na realidade por meio de mecanismos semânticos que a OO oferece como, por exemplo, os conceitos de herança, polimorfismo e encapsulamento.

A herança entre classes permite representar hierarquias entre as entidades identificadas na realidade do negócio. Este mecanismo permite o mapeamento de especializações e generalizações de objetos, ou seja, uma classe, chamada de subclasse, pode ser definida a partir de outra classe, denominada de superclasse, herdando as instâncias dos atributos e os métodos encontrados na superclasse. Segundo Boscarioli et al (2006), trata-se de um mecanismo muito útil quando são identificados no domínio do problema diversos objetos que possuam diversas características em comum e outras características singulares a certas entidades, criando uma classe geral englobando os atributos e métodos comuns e mantendo as diversas subclasses com suas próprias implementações. Este tipo de

mapeamento é denominado de generalização. A modelagem da situação contrária também acontece (especialização). Elmasri & Navathe (2002) apresentam esses mesmos conceitos no modelo ERE, uma extensão do modelo ER, mas esse se difere dos princípios propostos pela OO, pois a herança é implementada como relacionamentos entre as entidades, existindo forte comunicação entre tabelas e não objetos independentes, como no modelo OO. A figura 3.3.1 apresenta um exemplo de relacionamento de generalização, utilizando a UML. Como mostra a figura, os relacionamentos que representam as generalizações possuem a notação de uma seta de ponta vazia que vai da subclasse para a superclasse.

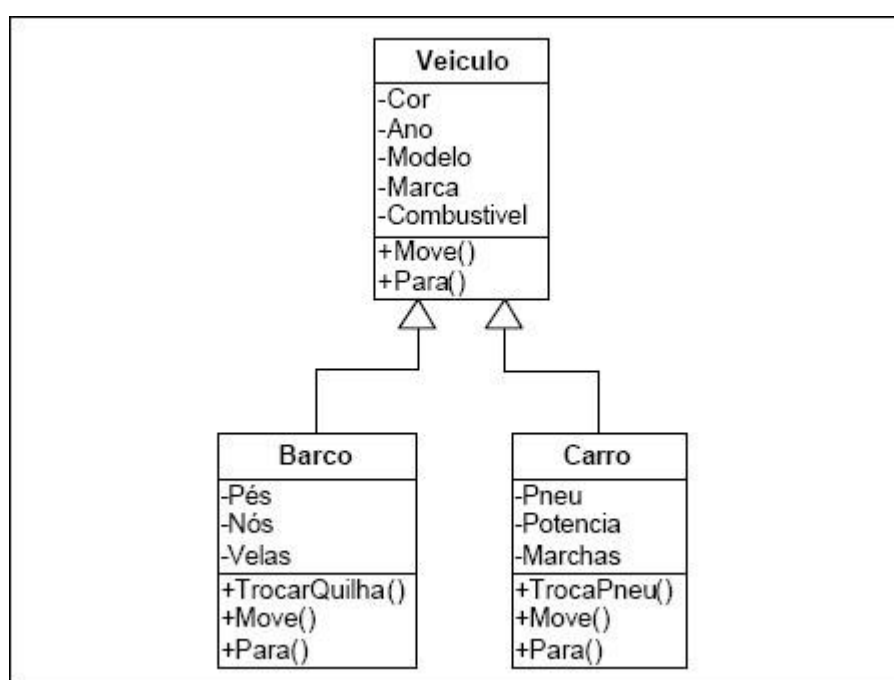


Figura 3.3.1 – Generalização das classes Barco e Carro em superclasse Veículo.
Fonte: BOSCAROLI, 2006.

Outro mecanismo relevante nas implementações de sistemas e modelagem de classes é o polimorfismo. Este tipo de mecanismo, também conhecido como sobrecarga de operador (*operator overloading*), permite que diversas classes implementem uma mesma operação de forma diferente. Definindo mais especificamente, através do polimorfismo, a OO permite que diferentes classes possuam uma mesma assinatura de método e que seu conteúdo possa ser desenvolvido de forma diferente. Boscaroli et al (2006) afirma que a decisão de qual classe executará o método em um determinado instante será feita pelo sistema de gerenciamento, tanto do banco como do compilador da linguagem, em tempo de

execução. Algumas linguagens de programação, como o Java, por exemplo, implementam, automaticamente, uma generalização constante das classes criadas pelo desenvolvedor para uma mesma superclasse pré-estabelecida. Dessa forma, toda classe definida pelo usuário pode utilizar o polimorfismo, pois são especializadas de uma mesma superclasse.

Para Elmasri & Navathe (2002), o encapsulamento é uma das principais características da OO e está relacionado à ocultação de informação, conceito existente nas linguagens de programação que seguem este paradigma. O conceito de encapsulamento pode ser percebido na estrutura de formação dos objetos, pois os dados do objeto são protegidos pelos métodos através de restrições de visibilidade e as implementações destas operações, como de todo o objeto, também não são conhecidas pelos usuários que as utilizam. Conforme Vieira (2001), o encapsulamento determina que apenas os métodos sobre os objetos sejam visíveis e toda sua estrutura fique oculta. De uma forma geral, o modelo OO está baseado no encapsulamento de dados e as operações e relacionamentos sobre estes dados.

A comunicação entre objetos é realizada através de mensagens. Uma mensagem é um estímulo provocado por um objeto sobre outro objeto que, dessa forma, executará um determinado método conforme a mensagem recebida. Esta mensagem pode alterar o estado do objeto passivo ou apenas obter alguma informação dele (BORBA, 2006).

Além de representar a estrutura de atributos e operações dos objetos através de classes, o modelo OO também representa os relacionamentos entre eles. Os relacionamentos aqui possuem a mesma semântica daquela apresentada para os relacionamentos no modelo ER. Contudo, mais tipos de relacionamentos poderão ser mapeados na UML (Unified Modeling Language), obtendo maior fidelidade à semântica da realidade modelada, como os conceitos de dependência, associação, generalização e, derivando da associação, a agregação e a composição. Na UML, os relacionamentos muitos-para-muitos, quando possuem atributos, geram novas classes, chamadas de classes associativas.

Os relacionamentos que representam dependências indicam que uma determinada classe depende dos métodos ou dados de outra classe, indicando “que a alteração na especificação de um elemento pode afetar outro elemento que a usa, mas não necessariamente o oposto” (SILVA & VIDEIRA, 2001: 169). Já as

associações representam ligações estruturais entre objetos e podem apresentar restrições a serem implementadas.

As associações podem ser mapeadas seguindo dois formatos: agregação e composição. A agregação, como também a composição, trata-se de associações do tipo *todo-parte*, cujo lado *parte* é chamado de *objetos-parte* que compõem o lado *todo* representado pelo *objeto-todo*. Para Bezerra (2003), na agregação, os objetos *parte* podem pertencer a diversos objetos *todo* e são independentes da existência deste, ou seja, excluir um objeto *todo* não implica em destruir objetos *parte* que podem ser criados e manipulados independentemente disso. Uma situação diferente é representada na composição, na qual os objetos *parte* pertencem a um único objeto *todo*. Dessa forma, quando este é excluído, todos os objetos que o compõe são destruídos. A agregação tem na UML a notação gráfica de um diamante branco no lado *todo* e a composição apresenta um diamante negro (BEZERRA, 2003).

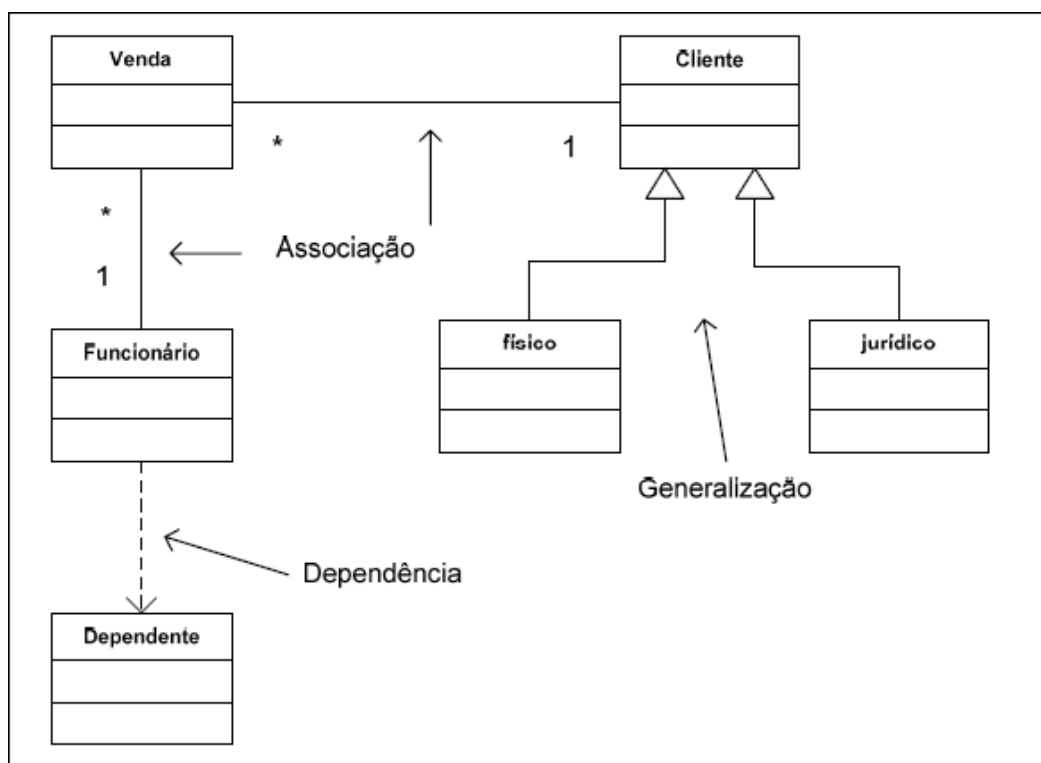


Figura 3.3.2 – Alguns tipos de relacionamentos entre classes na UML.
Fonte: BORBA, 2006.

Proposta por Booch, Rumbaugh e Jacobson em 1995 e adotada como padrão pela OMG (Object Management Group) em 1997, a UML é uma linguagem visual para modelar sistemas sob os conceitos da OO, cujos elementos gráficos possuem sintaxe e regras bem definidas para construção de diagramas que

representam todas as perspectivas dos sistemas modelados. Para Rumbaugh, Jacobson & Booch (2004), uma característica muito importante da UML é que se trata de um padrão de modelagem totalmente independente da tecnologia utilizada, favorecendo o entendimento do cliente e sua participação em projetos conceituais e lógicos de modelagem. Processos de desenvolvimento necessitam e produzem diversos documentos textuais ou gráficos, chamados de artefatos, quando utilizam a UML como suporte à modelagem (BEZERRA, 2003). A UML oferece diversos elementos, distribuídos em 9 diagramas, para modelar todas as visões do sistema e os artefatos gráficos necessários no desenvolvimento. Dentre estes diagramas, será conceituado neste trabalho apenas o diagrama de classes, que servirá para representar um modelo dimensional sob os conceitos da OO no modelo de classes.

O diagrama de classes da UML representa a estrutura estática dos objetos e seus relacionamentos, desde a análise até a especificação do modelo. No modelo de classes, de forma análoga à modelagem ER considerando os projetos conceitual, lógico e físico, existem também projetos que modelam os diferentes níveis do processo de desenvolvimento denominados de Modelo de Classes de Domínio, Modelo de Classes de Especificação e Modelo de Classes de Implementação. Como no nível conceitual, o modelo de classes de domínio representa as classes existentes no domínio do negócio modelado, sem considerar detalhes da tecnologia que será utilizada. Extensão deste primeiro, o modelo de classes de especificação complementa detalhes do ambiente de software a ser utilizado, mas ainda está em um nível alto de abstração. Já o último modelo corresponde a este segundo modelo implementado em uma linguagem de programação OO com detalhes mais profundos de solução técnica sob o ponto de vista dos analistas. Nesta fase, decide-se em qual linguagem de programação, geralmente do tipo OO, as classes serão implementadas (BEZERRA, 2003).

A fim de modelar como os objetos se encontram estruturados no sistema, o diagrama de classes apresenta diversas regras muito bem definidas para representar as classes e seus relacionamentos. Segundo Bezerra (2003), a representação deve possuir, no máximo, três grupos de informações e cada um deles pode exibir diversos elementos. O primeiro grupo de uma classe no diagrama deve ser o seu nome e pode apresentar, também, um estereótipo informando o tipo de domínio que aquela classe representa. Em bancos de dados OO e em sistemas

que devem manter os dados de um objeto armazenados, mapeia-se o estereótipo *entity* ou persistente. Silva & Videira (2001) ainda complementam que a classe pode ter um nome mais completo, tendo seu nome simples precedido pelo nome do pacote a qual ela pertence separado pelos caracteres '::'. A segunda seção que pode ser exibida é a lista de atributos da classe, local onde os nomes dos atributos são apresentados, podendo exibir mais elementos como grau de visibilidade, geralmente visibilidade privada denotada por um sinal de menos, e o tipo do atributo. E a terceira parte é a lista de operações que, também, apresentam seus nomes e podem apresentar parâmetros, além do grau de visibilidade e o tipo de retorno.

Rumbaugh, Jacobson & Booch (2008), apresentam, ainda, uma quarta subdivisão na representação UML para classes, contendo as responsabilidades, descritas como texto em formato livre.

Além das regras para representar classes, o diagrama de classes possui um conjunto de notações para a semântica dos tipos de relacionamentos entre os objetos mapeados no modelo. De uma forma geral, neste diagrama, os relacionamentos são chamados de associações. As associações são exibidas através de uma linha que liga as classes correspondentes aos objetos que se relacionam e podem apresentar outros elementos opcionais que são utilizados quando se deseja um diagrama mais fiel à realidade.

Conforme Rumbaugh, Jacobson & Booch (2004), a multiplicidade é a propriedade mais importante de uma associação e exibe a quantidade de instâncias de objetos a que um determinado objeto pode estar associado, ou seja, representa a cardinalidade, como descrito nos modelos anteriores, entre os objetos existentes no sistema. A notação utilizada para os relacionamentos com conectividade um-para-um e um-para-muitos, respectivamente, é do tipo $x..1$ e $x..*$, onde x é igual a 0 ou 1, representando participação opcional ou obrigatória de uma instância no relacionamento. O asterisco representa a quantidade máxima de instâncias, sem limite, que o objeto do lado x está associado. Bezerra (2003) menciona que podem existir intervalos específicos em uma determinada multiplicidade para expressar uma quantidade definida e obrigatória de participação de um objeto.

As associações podem apresentar outros adornos para agregar mais significado ao relacionamento, podendo ser nomeadas, apresentar restrições e navegabilidade, indicar sentido de leitura e visualizar papéis. Toda associação, por

padrão, possui caráter bidirecional.

A navegabilidade, representada por uma seta em um dos extremos da associação, define qual classe na associação pode acessar informações de uma ou mais instâncias da outra classe.

As restrições representam condições de modelagem ou técnicas as quais ela esteja submetida. Segundo Silva & Videira (2001) e Bezerra (2003), as restrições são mecanismos que permitem estender ou alterar a semântica do diagrama de forma mais consistente, mas recomenda-se que estas sejam incluídas de forma textual. Caso não seja possível, existe na UML uma linguagem específica para modelar restrições, a OCL (Object Constraint Language).

Já o sentido de leitura é um elemento que informa, simplesmente, de qual classe para qual classe os envolvidos devem ler o nome da associação a fim de deixar mais claro o modelo.

Segundo Bezerra (2003), também é possível informar os papéis que as classes possuem na associação, pois toda classe pode possuir um papel diferente em toda associação. Esse tipo de recurso é pouco usado, mas muito interessante quando se trata de uma associação reflexiva, quando uma classe está associada a ela mesma, para o melhor entendimento do usuário. Sempre quando for difícil representar uma determinada associação, é recomendável representar os papéis das classes envolvidas e, no mínimo, nomear tal associação, pois assim melhora a legibilidade do modelo (BEZERRA, 2003).

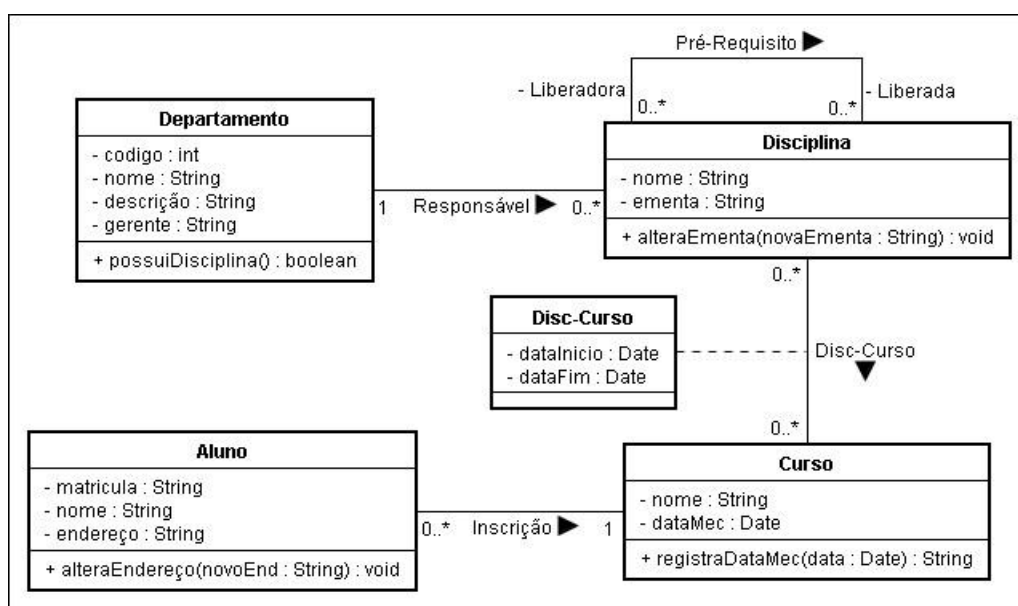


Figura 3.3.3 – Diagrama de classes baseado no modelo ER da figura 3.1.2.

A figura 3.3.3 acima mostra um diagrama de classes simples baseado no diagrama do modelo ER da seção 3.1 deste trabalho. Este diagrama de classes possui a maioria dos elementos definidos até agora. As classes da figura apresentam as seções, de cima para baixo, que informam seu nome, a lista de atributos e a lista de operações. As associações apresentam nomes, sentidos de leitura e multiplicidades. Pode-se perceber a presença de uma classe-associativa de nome *Disc-Curso* e a existência da associação reflexiva de nome *Pré-Requisito* que apresenta os distintos papéis que a classe *Disciplina* possui nesta associação, melhorando a legibilidade do modelo e informando que uma determinada *Disciplina Liberadora* é pré-requisito de uma determinada *Disciplina Liberada*.

Como já abordado, a UML é um padrão que oferece diversos elementos e notações bem definidas que modelam as características essenciais de uma realidade. Sob este padrão, o modelo de classes é uma das partes mais importantes da modelagem OO e reflete a visão estática do sistema a ser desenvolvido. Segundo Bezerra (2003), o modelo de classes é composto pelo diagrama de classes, componente utilizado para representar a estrutura estática modelada, e por uma descrição textual associada. Estes conceitos se mostram muito importantes para o desenvolvimento deste trabalho, pois Borba (2006) informa na terceira etapa do seu modelo que existe uma conversão conceitual direta entre o modelo dimensional e o diagrama de classes UML, representando o modelo estático dimensional segundo o paradigma OO. Este modelo será aprofundado no capítulo a seguir.

4 MODELO DIMENSIONAL EM AMBIENTE DE PERSISTÊNCIA DE OBJETOS

Este capítulo descreve a metodologia proposta por Borba (2006), aprofundando nas partes utilizadas para o modelo de desenvolvimento deste trabalho. Além disso, apresenta o modelo referente ao Crédito Sem Barreiras (CSB), mantido pela empresa Core Synesis, pois o foco é a modelagem para um ambiente de persistência de objetos e não as áreas da análise de requisitos e gestão de negócio, pré-requisitos da modelagem dimensional. Dessa forma, a estrutura do capítulo inicia-se com a abordagem do modelo de Borba (2006), apresentando sucintamente as ferramentas utilizadas e a empresa Core Synesis. Descreve também o modelo dimensional do CSB e aprofunda no mapeamento deste em diagrama de classes UML e sua implementação em um ambiente OO integrado. Ainda são descritos alguns testes e análises de resultados, principalmente para os exemplos de processo de carga criados, de uma base operacional para o ambiente de persistência OO.

Partindo da análise entre as diversas metodologias divulgadas ao longo dos anos para implementar o modelo dimensional na construção de um DW, Borba (2006) visa atender a todos os requisitos analisados em seu modelo. Como citado no capítulo anterior, este modelo possui 5 etapas que vão desde a definição do modelo de negócio até o mapeamento do modelo dimensional para um BDOO, seguindo as notações da UML e ODMG, utilizando a linguagem de definição de objetos ODL (BORBA, 2006). Neste trabalho, a implementação foi realizada, utilizando a linguagem orientada a objetos Java, associada ao framework de persistência de objetos db4o.

A primeira etapa proposta por Borba (2006) consiste na definição do modelo de negócio, na análise de requisitos junto ao cliente, a fim de contemplar as decisões das áreas a serem atendidas pela solução de DW modelada. Conforme Borba & Morales (2006), neste momento, os detalhes do negócio que serão base no desenvolvimento do projeto são analisados e traduzidos para o ambiente operacional adotado através de um modelo ER, documentando a finalidade do projeto, o escopo da realidade analisada e os recursos técnicos que serão utilizados.

Logo a seguir, é realizada a geração do modelo multidimensional, ou seja, o modelo de negócio identificado. Nesta etapa são identificados os sistemas e bases de dados operacionais, inclusive suas funcionalidades dentro deste ambiente, para mapear o modelo dimensional. Borba (2006) enfatiza que o modelo ER desenhado na etapa anterior deve ser minuciosamente analisado para que possam ser extraídas as dimensões que fazem parte do negócio, podendo indicar dimensões implícitas a serem modeladas. A granularidade das entidades também merece atenção, pois a partir desse estudo serão mapeados os atributos das dimensões e os fatos do negócio a fim de definir tabelas de fatos e métricas de derivação. A estrutura do modelo dimensional gerado poderá ser de quaisquer tipos como, os mais conhecidos, *Star Schema* e *Snowflake Schema*. É necessário, durante a análise, definir uma dimensão Tempo, pois o DW promove a equivalência de tempo sob a visão do negócio, agregando dados por um determinado tipo de período, conforme a necessidade da disposição dos dados (BORBA, 2006).

As próximas etapas da metodologia abordam a representação do modelo dimensional em diagrama de classes UML; o mapeamento deste diagrama para um BDOO; e, finalmente, a sua implementação. A figura 4.1 demonstra a ordem de execução destas etapas e um esboço de toda a metodologia de Borba (2006).

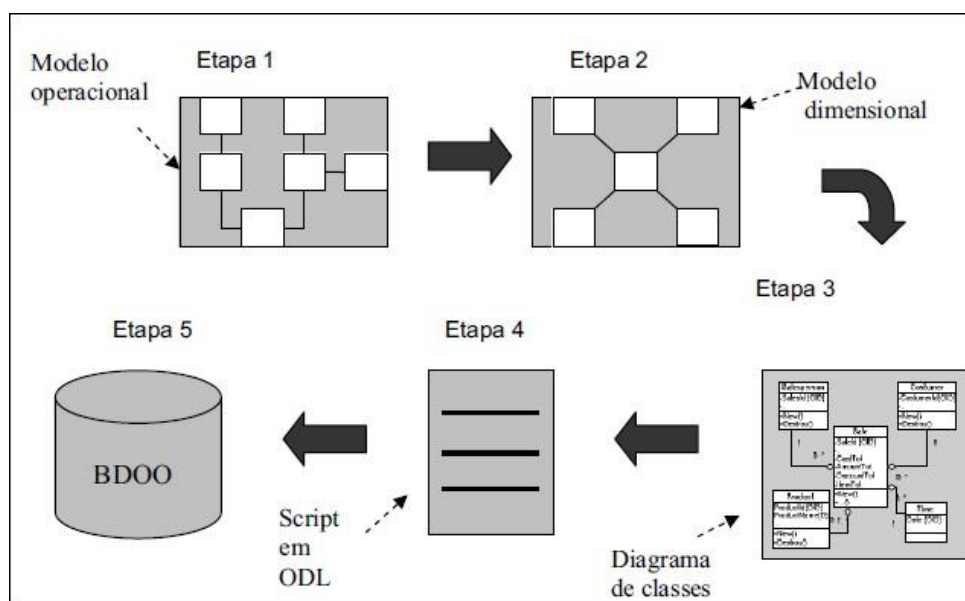


Figura 4.1 – Resumo da metodologia de Borba e ordem de execução das etapas.
Fonte: BORBA & MORALES (2006)

Estas etapas são descritas nas próximas seções, cada uma em seção própria, utilizando um modelo dimensional real já definido e mantido pela Core

Synesis. Segue, antes disso, uma breve apresentação da empresa citada e das ferramentas utilizadas para, além de fundamentar, exemplificar a aplicação da metodologia de Borba (2006) sobre modelos de dados reais.

4.1 As ferramentas utilizadas

Para representar o modelo dimensional em diagrama de classes UML foi utilizado o software Jude Community 3.2.1, disponibilizado para download no *site* oficial <http://jude.change-vision.com/jude-web/index.html>, sendo necessário se cadastrar gratuitamente. A codificação das classes mapeadas nesta fase foi realizada em linguagem de programação Java, utilizando-se o ambiente de desenvolvimento Eclipse, versão 3.2, integrado ao framework de persistência de objetos db4o na versão 7.4, disponíveis para download, respectivamente, em seus endereços <http://www.eclipse.org> e <http://developer.db4o.com>. É necessário instalar plug-ins do db4o no Eclipse para a integração. Estes podem ser baixados no *site* oficial citado do db4o. Para implementar o teste de execução realizado e descrito na última seção, foi utilizado o Oracle Express 10g para criar um ambiente operacional.

Excetuando o uso do ambiente Oracle para a implementação do banco operacional, a escolha dos softwares utilizados ateu-se pelo fato de que estes implementam os conceitos do paradigma OO nos níveis em que este trabalho se baseia, desde a análise e mapeamento de modelos até a fase de implementação para a persistência de dados.

Conforme Paterson et al (2006), o db4o é um tipo de BDOO de código aberto que permite aos desenvolvedores de linguagens de programação OO reduzir consideravelmente os custos envolvidos no desenvolvimento, alcançando melhorias de performance superiores aos bancos de dados relacionais. Segundo seu *site* oficial, o db4o executa consultas até 44 vezes mais rápido que algumas soluções altamente conhecidas no mercado, como o hibernate, sendo que este apenas oferece um mapeamento de aplicações OO para bancos de dados relacionais e não persistência direta de objetos, como o BDOO aqui discutido (DB4OBJECTS, 2008).

A figura 4.1.1 abaixo ilustra um comparativo entre as técnicas em bancos

de dados relacionais e o uso do db4o para a implementação dos objetos modelados. Observa-se que existem persistência e mapeamento diretos dos objetos no BDOO, facilitando a consulta a esses objetos, diferentemente de um SGBD relacional associado às técnicas de mapeamento objeto-relacional, como o hibernate, que implementam fisicamente os objetos modelados em diversas tabelas, projetando maior complexidade na manipulação dos mesmos.



Figura 4.1.1 – Forma de armazenamento de objetos de um SGBD relacional e o db4o.
Fonte: DB4OBJECTS, 2008.

Dessa forma, além da aplicação dos conceitos do paradigma OO, a escolha do db4o para a implementação de um modelo dimensional também se fundamenta pelo alto desempenho conseguido nas consultas aos objetos, principalmente por aplicações que trabalham com linguagens OO.

4.2 Modelo dimensional utilizado

O modelo utilizado para a aplicação da metodologia de Borba (2006) trata-se do modelo dimensional do serviço Créditos Sem Barreiras (CSB), disponibilizado pela operadora Vivo em Minas Gerais, e encontra-se implementado no Data Mart de Vendas. Este Data Mart fornece informações referentes ao gerenciamento de vendas da Vivo-MG e armazena os dados das ativações de contratos de celulares de planos pré e pós-pagos, das vendas de produtos como aparelhos e chips, das ativações de cartões e o acompanhamento entre estas vendas e suas metas. Estes dados são disponibilizados diariamente para acesso do usuário por meio de ferramentas OLAP.

O serviço de Crédito sem Barreiras é uma forma de venda de recarga programada da Vivo-MG. O sistema origem permite que o cliente com um celular de

plano pós-pago agende a compra de créditos pré-pagos para outro celular designado por ele. Os usuários do DW acompanham a movimentação da carteira, adesões e cancelamentos referentes ao serviço. Existem análises por clientes pré-pagos distintos que aderiram ao CSB e também por agendamentos. Informações mais detalhadas como os números de celulares pós e pré-pagos, valores de crédito, canal de compra e funcionário que efetuou a venda são extraídas diretamente da base de ETL2 do DW, onde os dados são tratados e manipulados conforme as demandas de negócio.

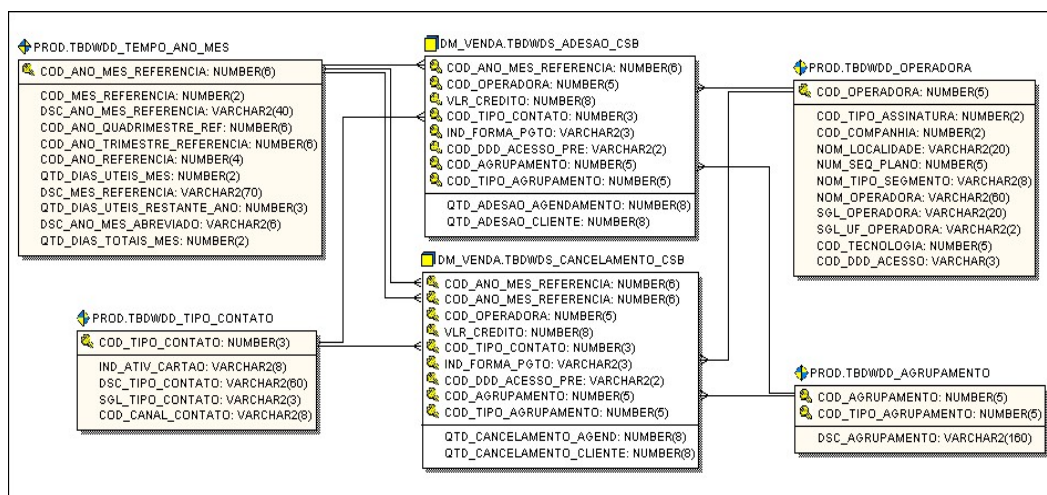


Figura 4.2.1 – Modelo dimensional CSB.

Fonte: Adaptado de Core Synesis, 2008.

O modelo dimensional CSB utilizado possui duas tabelas de fatos conectadas a outras quatro tabelas dimensionais, conforme a figura 4.2.1. Este modelo dimensional agrupa três modelos dimensionais representados pelas suas tabelas de fatos ADESAO_CSBB e CANCELAMENTO_CSBB, que armazenam informações sobre as adesões dos clientes ao serviço e os cancelamentos realizados, respectivamente. As outras quatro tabelas TEMPO_ANO_MES, TIPO_CONTATO, OPERADORA e AGRUPAMENTO representam, assim, a dimensão de tempo no negócio, com uso muito importante em ambientes DW e consultas analíticas OLAP; por qual tipo de contato o cliente solicitou sua adesão ou cancelamento do serviço CSB; a qual operadora o cliente solicitante ou beneficiado pertence; e os agrupamentos de clientes por alguma característica, serviço ou planos em comum mapeados na base de dados. Os atributos-chave estão representados por um desenho de uma chave no modelo dimensional da figura.

Como descrito na seção referente à fundamentação dos modelos de dados, mais especificamente o modelo dimensional, os atributos-chave das tabelas de fatos são compostos por referências ou *Foreign Keys* das chaves primárias das tabelas dimensionais. Quando necessário, como neste modelo, outros atributos relevantes de um fato de negócio podem também fazer parte do grupo de atributos-chave das tabelas fato a fim de melhorar a qualidade de resposta nas consultas analíticas. Conforme Machado & Abreu (2004), os relacionamentos entre as tabelas estão representados logicamente por estas *Foreign Keys* e, como todo modelo que possui configuração baseada no ER, para cada valor atômico presente nestes campos chave das tabelas fatos existe correspondente de mesmo valor no atributo-chave daquela dimensão, permitindo assim buscar as devidas associações no conjunto de resposta solicitado pelo usuário.

Sob uma análise mais geral e melhor entendimento, o modelo dimensional representado acima agrupa as semânticas de negócio referentes às informações de adesões e cancelamentos do CSB solicitados por um cliente de uma determinada operadora através de um tipo de contato para um determinado período de tempo. A tabela ADESAO_CSB reflete os fatos referentes às quantidades de adesões por agendamento e por solicitações avulsas pelos clientes. Já da tabela de fatos CANCELAMENTO_CSB extrai-se as quantidades de cancelamentos de serviços agendados e de solicitações avulsas.

4.3 Mapeamento do modelo dimensional em UML

Com o modelo dimensional definido por meio de um esquema estrutural como, por exemplo, o *Star Schema* ou *Snowflake Schema*, basta classificar as entidades e suas hierarquias conforme as regras do negócio a fim de mapeá-las sob a perspectiva orientada a objetos da UML, permitindo assim que os dados representados sejam posteriormente agregados (BORBA, 2006). Dessa forma, como visto nas definições acerca de classes, as tabelas dimensionais são modeladas como classes dimensionais no diagrama correspondente da UML e as tabelas de fatos são representadas por classes de fatos. Além de mapear as classes e seus

atributos para representar as tabelas do modelo, faz-se necessário também traçar os relacionamentos e as multiplicidades entre as classes fato e suas dimensões.

No mapeamento das tabelas do modelo dimensional em classes, além de seguir as notações de modelagem da UML, OMG e ODMG, deve-se aplicar os padrões usados no DW pela organização. Para auxiliar as equipes envolvidas na administração e desenvolvimento do DW, é interessante que a padronização utilizada para identificar, explicitamente, as tabelas de fatos e as dimensões nas consultas OLAP e em rotinas ETL seja modelada no diagrama de classes, refletindo, conseqüentemente, sobre a codificação da mesma. Segundo Bezerra (2003), o padrão UML propõe que os nomes de classes e relacionamentos comecem pela primeira letra maiúscula e o restante das letras em minúsculo para cada palavra que compõe o identificador, sem nenhum espaço ou caractere separador entre elas, inclusive o padrão da organização que será concatenado ao nome. Somente as siglas que permanecem inalteradas com todos os caracteres em maiúsculo. Os atributos e métodos da classe seguem este padrão com exceção da primeira palavra que terá todos os caracteres em minúsculo.

Conforme Borba (2006), uma das formas mais apropriadas para representar os relacionamentos entre as classes de fatos e as classes das dimensões é por meio de agregações compartilhadas, pois, pela semântica dos conceitos de BI em modelos dimensionais, a tabela fato agrega as dimensões. A agregação representa que um fato de negócio é constituído de diversas partes, as dimensões, ou seja, as dimensões do negócio fazem parte de um fato de negócio. Somente para exemplificar, pois em ambientes DW não existe formalmente exclusão e alteração de dados, a exclusão de um fato não implica que as dimensões que o constituíam devam ser excluídas, demonstrando que as informações referentes às dimensões são independentes dos fatos ocorridos.

Após incluir as agregações, as multiplicidades devem ser mapeadas para detalhar mais o diagrama. Nos relacionamentos entre as dimensões e a fato, a extremidade referente às dimensões possui multiplicidade 0..1, pois estas armazenam os dados em mais baixo nível; e a extremidade junto à fato possui multiplicidade 1..*, pois as dimensões estão em diversas instâncias de fatos (BORBA, 2006). Para Borba (2006), o mapeamento das multiplicidades é muito importante quando o modelo dimensional tratado possui a estrutura de Floco de

Neve, pois a cardinalidade entre dimensões deve ser devidamente representada, modelada pelo diagrama de estrutura composta. No caso deste trabalho, pelo fato do modelo dimensional CSB apresentar-se por um esquema Estrela, este tipo especial de diagrama não será utilizado.

Para aplicar esta etapa da metodologia, os mapeamentos acima serão descritos logo a seguir. Tendo como base o modelo dimensional CSB e os padrões considerados, foram modeladas as classes das dimensões TempoAnoMes, TipoContato, Operadora e Agrupamento que correspondem, respectivamente, às tabelas dimensionais TEMPO_ANO_MES, TIPO_CONTATO, OPERADORA e AGRUPAMENTO. As tabelas de fatos ADESAO_CSB e CANCELAMENTO_CSB moldaram as correspondentes classes fato AdesaoCSB e CancelamentoCSB. Pode-se perceber no modelo dimensional utilizado que existem algumas notações de negócio como os termos DM_VENDA, PROD, TBDWDS e TBDWDD. Estes padrões, utilizados pela Core Synesis, possuem os seguintes significados:

- DM_VENDA e PROD são *owners* implementados para organizar esquemas proprietários sobre um grupo de tabelas no banco de dados que correspondem aos ambientes do Data Mart de Vendas e as tabelas de produção em ETL2.
- TBDWDS e TBDWDD, abreviações de Tabela do DW para Dados Sumarizados e Tabela do DW para Dados de Dimensão, são apenas nomenclaturas incluídas nos nomes das tabelas para identificar, neste caso, as tabelas fatos sumarizadas e as dimensões, respectivamente.

Na construção do diagrama, para implementar o último tipo de notação acima, as abreviações CS e CD foram concatenadas aos nomes das classes para identificar, respectivamente, as classes de fatos sumarizadas e as dimensões na manipulação dos dados. As classes de fatos do diagrama dimensional, agora nomeadas de CSAdesaoCSB e CSCancelamentoCSB, foram generalizadas para uma classe CSCSB, pois elas apresentam atributos em comum, além de possuírem dados referentes ao mesmo tipo de serviço, no caso o Crédito Sem Barreiras.

Conforme Cattell et al (2000), todo objeto em uma base de dados possui um identificador único OID e se mostra interessante que este seja modelado nos diagramas de dados. Borba (2006) confirma que os identificadores devem ser representados de forma explícita, mas somente para classes de dimensão,

colocando-se a restrição {OID} no atributo definido para cada classe. Já os descritores são os atributos representativos de cada classe dimensional e devem ser, preferencialmente, textuais por serem campos de referência para o usuário, modelados com a restrição {D} no diagrama de classes UML (BORBA, 2006). Um padrão sugerido neste trabalho para nomear os OIDs é concatenar o termo OID e o nome da classe correspondente sem carregar a sigla referente ao padrão CD para reconhecimento de dimensões. Conforme as análises e padrões descritos, o diagrama de classes CSB é apresentado na figura 4.3.1.

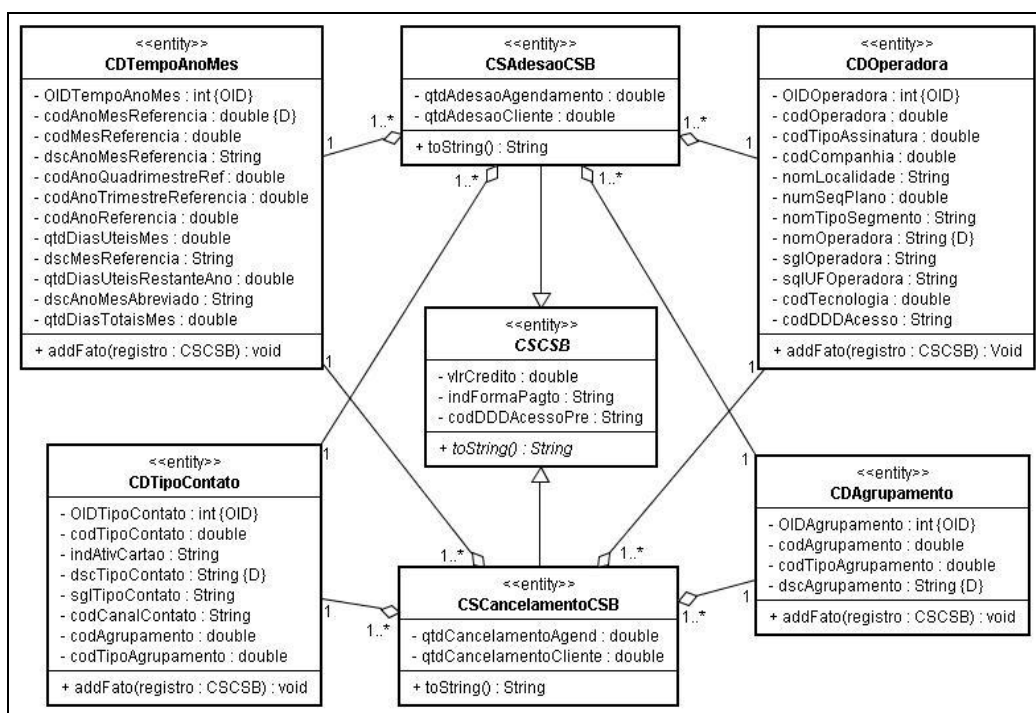


Figura 4.3.1 – Diagrama de classes CSB.

Percebe-se na figura acima que as classes fato **CSAdesaoCSB** e **CSCancelamentoCSB** estão conectadas diretamente às classes de dimensões **CDTempoAnoMes**, **CDTipoContato**, **CDOperadora** e **CDAgrupamento**, assim representando os mesmos dois modelos dimensionais referentes às adesões e cancelamentos do serviço CSB. Os descritores escolhidos no diagrama são, justamente, os atributos mais significativos e que possuem o tipo *String* modelado por serem atributos textuais.

Analisando Rumbaugh, Jacobson & Booch (2004), classe pode possuir métodos que retornam e alteram diretamente os dados de seus atributos. Estes métodos são representados pelas operações *gets* e *sets*, mas, por padrão, não é

necessário mapeá-los explicitamente no diagrama. Foram representadas somente as operações *addFato* nas dimensões e *toString* para as classes fato que correspondem, respectivamente, aos métodos para adicionar os fatos que cada dimensão participa e a descrição dos valores internos dos fatos ocorridos, métodos dos quais serão implementados na próxima etapa da metodologia. A classe CSCSB criada para generalizar as classes fato foi modelada como classe abstrata, pois mesmo contendo atributos e operações próprios, não será instanciada para armazenar dados. Para implementar o polimorfismo no diagrama, a operação *toString* definida nas classes de fatos foi mapeada também para esta classe CSCSB, pois mesmo que o diagrama de classes seja muito usado no nível conceitual da modelagem de dados, pode-se, conforme necessário, adicionar detalhes de implementação (BEZERRA, 2003).

4.4 Implementação do diagrama UML no BDOO

Nesta seção está descrito o processo de implementação da etapa da metodologia de Borba (2006) que visa codificar o *script* para um determinado BDOO a partir do diagrama de classes obtido anteriormente. Para este trabalho, houve adaptações para a geração do código referente à persistência de objetos que, na verdade, utiliza uma linguagem de programação OO para desenvolver diretamente as classes UML modeladas. Borba & Morales (2006) comentam que o padrão ODMG oferece a linguagem OQL para este tipo de implementação, mas além de não estar difundida no mercado, ela não oferece interoperabilidade e trata-se de um padrão de codificação que será depois implementado, independente de linguagem de programação. Interoperabilidade, segundo Vieira (2001), é a capacidade de uma determinada aplicação em acessar diversos sistemas diferentes. Já neste trabalho a abordagem utilizada delega aos desenvolvedores de linguagens de programação OO, neste caso a linguagem Java, a codificação direta das classes e relacionamentos pelo padrão UML. Java é uma linguagem altamente difundida no desenvolvimento de softwares e oferece portabilidade através da máquina virtual Java (SUN MICROSYSTEMS, 2009).

Conforme Cattell et al (2000), o modelo de objetos propõe pelo padrão ODMG que o estado de um objeto seja representado por seus atributos e relacionamentos. Este tipo de representação é muito importante, pois as consultas em BDOO são feitas de acordo com os estados fornecidos, ou seja, de forma mais específica, o framework db4o utilizado busca os objetos que possuem um determinado estado requerido na consulta.

Na formalização do modelo de dados como também abordado no diagrama de classes UML, foi definido que uma classe também possui operações que devem ser representadas e, posteriormente, implementadas em métodos presentes no escopo desta classe (RUMBAUGH, JACOBSON & BOOCH, 2004). Como citado anteriormente, além das operações definidas explicitamente no diagrama, os métodos implícitos também devem ser codificados, com exceção das operações *sets* que não devem ser implementadas, pois, lembrando novamente, não é usual que algum dado existente nas dimensões e nas classes de fatos seja deletado ou alterado. Somente para ilustrar, as classes na linguagem Java são definidas pela palavra reservada *class*, geralmente acompanhadas pelo termo *public* para que as mesmas sejam acessíveis nas consultas realizadas pelos usuários. Na figura 4.4.1 abaixo a classe dimensional CDAgrupamento está definida em Java, conforme os mesmos atributos definidos no diagrama de classes CSB e as operações implementadas pelos métodos *addFato* e *gets* destes atributos.

```
public class CDAgrupamento {  
  
    private int codAgrupamento;  
    private int codTipoAgrupamento;  
    private String dscAgrupamento;  
    private ArrayList<CSCSB> listaFatos;  
    public int getCodAgrupamento() {  
        return codAgrupamento;  
    }  
    public int getCodTipoAgrupamento() {  
        return codTipoAgrupamento;  
    }  
    public String getDscAgrupamento() {  
        return dscAgrupamento;  
    }  
    public void addFato(CSCSB credito) {  
        this.listaFatos.add(credito);  
    }  
}
```

Figura 4.4.1 – Código parcial da classe de dimensão CDAgrupamento em Java.

Nota-se na figura acima um tipo especial de atributo, modelado como uma lista. Na verdade, este atributo representa o relacionamento de agregação existente entre a classe CDAgrupamento e as classes de fatos do diagrama. Conforme Rumbaugh, Jacobson & Booch (2004), os relacionamentos são representados em atributos ou classes de associação pelo padrão OO. Como a própria multiplicidade indica no diagrama, a agregação foi implementada por atributos do tipo lista de fatos nas classes de dimensão, como no exemplo acima; e por atributos monovalorados nas classes fato, contendo cada um os tipos das dimensões.

Refletindo agora sobre alguns detalhes de implementação, toda classe deve possuir um método construtor para inicializar todos os seus atributos quando a mesma é instanciada, ou seja, quando o objeto correspondente é criado. Este método construtor recebe os valores atômicos referentes a todos os seus atributos para as inicializações, exceto para a lista de fatos que deve ser apenas criada no construtor, pois terá elementos adicionados nela somente quando ocorrer um fato do qual a referida dimensão participe. Mesmo que não seja instanciada, pois não armazenará diretamente os dados, a classe abstrata CSCSB possui um método com estrutura semelhante à de um construtor, mas este serve somente para inicializar os atributos que estão no mais alto nível da generalização e que serão herdados pelas classes de fatos instanciadas.

Como as classes dimensionais possuem o atributo especial para armazenar a lista de fatos, elas carregam também a declaração de uma biblioteca própria da linguagem Java para permitir a implementação deste tipo de lista. Conforme Bezerra (2003), para que o conceito estrutural de objeto seja implementado, os atributos das classes devem possuir visibilidade privada e os métodos, inclusive os utilizados para acessarem esses atributos, devem possuir visibilidade pública, modeladas no Java pelas palavras reservadas *private* e *public*, respectivamente. O código completo referente à implementação de todas as classes do diagrama dimensional CSB definido na etapa anterior encontra-se no APÊNDICE A deste documento.

Observa-se no código Java desenvolvido que os identificadores e descritores não estão assim explícitos, sendo que estes primeiros nem sequer encontram-se implementados. Segundo Cattell et al (2000), os OIDs são transparentes para os usuários e os desenvolvedores, ou seja, o OID deve ser representado nos diagramas, mas não precisa ser codificado explicitamente: o

próprio sistema de gerenciamento de objetos do ambiente OO utilizado para persistir os objetos gera, automaticamente, um OID associado ao objeto no momento em que este é instanciado. Isto se aplica também à restrição de descritores das classes de dimensões, pois mesmo que sejam atributos reais escolhidos pela sua representatividade, não se faz necessário codificar esta restrição. De uma forma geral, as restrições sobre atributos na UML são adornos que facilitam o entendimento do modelo pelos usuários e não refletem, necessariamente, um detalhe real de desenvolvimento (BEZERRA, 2003).

Já nas classes de fatos os atributos que correspondem às métricas foram implementados como variáveis estáticas, usando-se a palavra reservada *static*. Toda vez que um objeto de fato for criado, indicando que um novo tipo de adesão ou cancelamento foi realizado, a métrica é incrementada automaticamente no método construtor, conforme o tipo de serviço ocorrido. Como são atributos estáticos dentro das classes, pelo próprio conceito de variável estática (SUN MICROSYSTEM, 2009), estas métricas são conhecidas por todos os objetos instanciados destas classes, ou seja, os valores pertencem mais à estrutura da classe do que do objeto criado. Dessa forma, os valores deste tipo de atributo são visíveis para todos os objetos que, como já discutido em seções anteriores, respeitam a mesma estrutura da classe da qual foram instanciados. Este tipo de implementação facilita em posteriores consultas OLAP que não sobrecarregariam o banco com cálculos de campos derivados para obter as sumarizações, pois basta consultar o campo pelo método *get* correspondente para obter esse somatório armazenado no BDOO.

4.5 Implementação e Execução de Consultas

Depois da codificação das classes sugere-se realizar a carga do novo banco de dados OO criado ou parte dele com os dados existentes no ambiente operacional. É neste momento que os recursos do BDOO escolhido são verdadeiramente utilizados, pois a definição das classes pode ser feita por qualquer linguagem de programação em ambientes de desenvolvimento diversos, inclusive por *scripts* para modelar as classes, mas é no processo de carga e realização de

consultas que o BDOO tem seu desempenho e funcionalidades avaliadas.

De acordo com Paterson (2006), o db4o trabalha com três formatos de consultas: o QBE, as Native Queries e o SODA. O QBE (Query-By-Example) é o mecanismo mais básico oferecido pelo db4o e fornece à consulta um objeto já criado que possui o mesmo estado do objeto desejado, ou seja, o objeto externo é criado apenas como exemplo para a consulta retornar o objeto armazenado que possui os mesmos valores de alguns ou todos os atributos fornecidos. As Native Queries são muito utilizadas para consultas mais complexas e são escritas com as expressões da linguagem de programação OO utilizada, ultimamente C# ou Java. Neste tipo de formato, informa-se ao banco um trecho de código da linguagem utilizada com toda a lógica requerida pela consulta através de um objeto do tipo *Query*, próprio de uma biblioteca do db4o. Já o formato SODA (Simple Object Data Access) é uma API que oferece diversos tipos de objetos e métodos para realizar as consultas sob uma perspectiva mais OO. Dessa forma, a partir de instancias de objetos próprios suportados pelo db4o, métodos SODA são combinados para realizar consultas que podem retornar uma lista de objetos, o *ObjectSet* (PATERSON, 2006). Sobre esta lista aplicam-se polimorfismo, encapsulamento e diversos mecanismos da OO para processar as informações obtidas. Para realizar as cargas e consultas das classes criadas nesta etapa, foi utilizado o formato SODA para manipular os dados e o código correspondente encontra-se implementado no APÊNDICE B.

O novo BDOO implementado a partir do diagrama CSB é um objeto do tipo *ObjectContainer* representado pela variável *dwds* no código que gera um arquivo de extensão YAP através do método *openFile* do db4o. Para inserir os dados no banco, basta invocar o método *store* deste objeto, passando como parâmetro os objetos a serem armazenados e, depois de utilizado, o método *close* é chamado para encerrar a sessão com o *ObjectContainer*. Em ambientes DW, a carga de dados é conhecida como ETL. De acordo com Kimball & Ross (2002), o processo de ETL pode ser dividido em estágios como a carga e filtragem de dados diretamente da base operacional para a área de *staging*, manipulação destes para melhoria de qualidade e a efetiva gravação no Data Mart correspondente. Estes estágios são comumente conhecidos no mercado por processos ETL1, ETL2 e ETL3.

O código implementado para os testes de carga e consulta possui dois métodos: *processoStagingProd*, que agrupa os dados de *staging* e ETL2 inseridos; e

processoDMVendasAdesao, que representa os dados de ETL3 do modelo dimensional ADESAO_CSB. Como este trabalho baseia-se em um modelo dimensional já existente, ou seja, quando a análise de negócio junto ao cliente já foi feita e o banco operacional não é completamente conhecido, percebe-se no código que os valores para os primeiros estágios de carga das classes dimensionais foram incluídos manualmente nas classes de dimensões. Esta forma de carga é conhecida no mercado pelo termo Carga Fria e foi escolhida no teste de execução pelo fato de não ser objeto deste trabalho detalhar os processos de carga. Conforme o site da Sun (SUN MICROSYSTEMS, 2009), para aplicações reais, estas cargas são facilmente implementadas via JDBC, conjunto de métodos muito utilizados comercialmente oferecidos pela tecnologia Java para acessar bancos de dados transacionais e mapear seus dados em objetos para, a partir disso, gravá-los no *ObjectContainer* do *bd4o* criado.

Já para o método *processoDMVendasAdesao*, além das consultas em SODA via *db4o*, foram utilizados os métodos da biblioteca JDBC com conexão direta ao Oracle Express 10g. Mesmo não conhecendo o ambiente operacional a priori, somente para dar mais consistência à fase de execução, um diagrama ER mais simples foi modelado para simular uma consulta aos dados operacionais, combiná-los aos dados das dimensões mapeadas e, dessa forma, gerar os objetos da classe de fatos *CSAdesaoCSB*. Descrevendo de forma sucinta, pois não é o foco deste trabalho modelar e detalhar os ambientes relacionais dos sistemas de origem, o modelo ER apresenta as tabelas *CLIENTE*, *PLANO*, *TIPO_PLANO*, *SERVICO*, *CLIENTE_PLANO* e *CLIENTE_SERVICO* que representam simplesmente cadastros dos clientes da base de dados da organização, o cadastro dos planos disponibilizados por ela, os tipos de planos, os serviços oferecidos e as tabelas resultantes de relacionamentos n:m entre as entidades *CLIENTE*, *SERVICO* e *PLANO*, pois um cliente pode ter vários serviços e planos. Cada serviço e cada plano podem também estar associados a diversos clientes.

A intenção de esboçar um modelo ER de um suposto ambiente operacional para o modelo dimensional CSB utilizado foi a de promover um teste de execução um pouco mais próximo da realidade a fim de abstrair, logicamente, sobre o uso da metodologia de Borba (2006) em projetos reais de construção de DW e controle de carga. Na figura 4.5.1 a seguir possui um esboço do referido diagrama ER modelado

com alguns dos diversos elementos que a modelagem ER oferece discutidos nas primeiras seções. Maiores detalhes a cerca do desenvolvimento do modelo em diagrama ER, incluindo os atributos das entidades e os *scripts* referentes à inserção de dados nas tabelas, encontram-se implementados em linguagem SQL no APÊNDICE C deste documento.

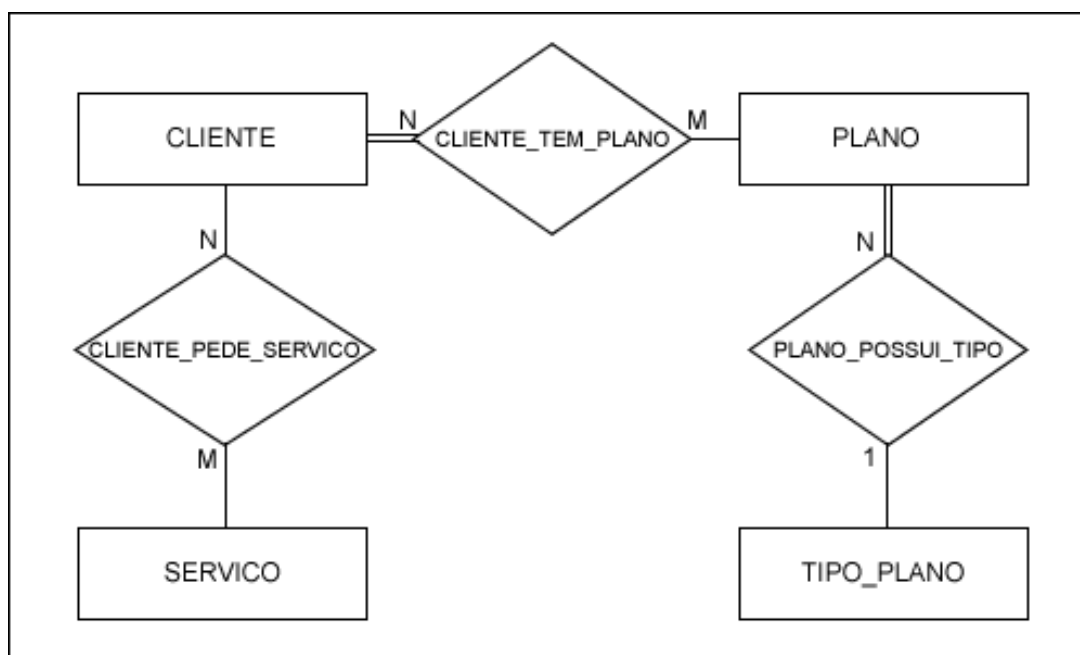


Figura 4.5.1 – Esboço do modelo ER da base operacional de teste.

A primeira consulta observada no método *processoDMVendasAdesao* do APÊNDICE B acessa o banco de dados operacional criado para o teste de execução, enviando via JDBC um trecho SQL que busca determinados valores a serem copiados para o modelo dimensional CSB implementado. A consulta deste trecho SQL, como descrita no código, refere-se aos dados das ativações no serviço Crédito Sem Barreiras (*restricaoServico* = 1) por todos os clientes de planos pós-pagos da Vivo-MG (*restricaoOperadora* = 10) que possuem código DDD igual a 031 (*restricaoDDD* = "031") no mês de dezembro de 2008 (*restricaoTempo* = 2008). Os dados retornados referem-se aos valores de créditos ativados, a forma de pagamento destes créditos, os prefixos de DDD dos beneficiados, os parâmetros de tempo, qual a forma de contato da solicitação, o tipo de plano a ser utilizado para o agrupamento de clientes e o tipo de solicitação utilizado para adesão ao CSB. Estes valores retornados como as restrições aplicadas na consulta ao banco operacional participam das consultas implementadas em SODA que retornam listas *ObjectSet*

dos tipos CDAGrupamento, CDOperadora, CDTipoContato e CDTempoAnoMes. Optou-se por capturar somente o primeiro, e no caso único, objeto destas listas que são combinados para formarem instâncias de objetos de fatos CSAdesaoCSB. Cada instância CSAdesaoCSB é armazenada no banco OO e, logo em seguida, é adicionada às listas de fatos das dimensões envolvidas na consulta, atualizando-as logo em seguida no *ObjectContainer*. Conforme Paterson (2006), no caso em que as listas retornem mais de um objeto, pode-se utilizar os métodos *hasNext* dos *ObjectSets* para verificar a existência de objetos que faltam ser processados.

Pode-se perceber que os objetos referentes a essas dimensões são buscados do banco, alterados e novamente gravados. Como já discutido, é papel do BDOO utilizado associar um novo OID para cada novo objeto armazenado, mas o db4o gerencia esse tipo de situação automaticamente, verificando se o objeto a ser armazenado pelo método *store* é resultante de algum processamento sobre algum objeto consultado do mesmo tipo. Caso seja, ele somente atualiza o objeto consultado, senão um novo objeto é armazenado (PATERSON, 2006).

Por fim, para ilustrar a codificação do teste, realizou-se um busca no banco de dados *dwdsv* de todas as ocorrências de objetos CSAdesaoCSB armazenados, apresentando como resultado de execução os atributos descritores das dimensões participantes e as métricas da própria classe de fatos através do método *toString*, conforme mostrado na figura 4.5.2 abaixo.

```
CodAnoMesReferencia: 200812
DscTipoContato: ATENDIMENTO ELETRONICO
NomOperadora: VIVO
DscAgrupamento: CORPORATIVO
QtdAdesaoAgendamento: 1
QtdAdesaoCliente: 1
VlrCredito: 35.0
IndAtivCartao: CARTAO VISA
CodDDDAcessoPre: 032

CodAnoMesReferencia: 200812
DscTipoContato: LOJA PROPRIA
NomOperadora: VIVO
DscAgrupamento: VAREJO
QtdAdesaoAgendamento: 1
QtdAdesaoCliente: 1
VlrCredito: 22.0
IndAtivCartao: A VISTA
CodDDDAcessoPre: 033
```

Figura 4.5.2 – Conjunto de resposta gerado no teste de execução.

Incluindo a realização desta busca como todo o processamento anterior

discutido, o db4o demonstrou bom desempenho de execução, semelhante aos custos de implementação e testes das aplicações OO pelo fato de basear suas consultas no próprio framework oferecido pela linguagem OO hospedeira. Conforme as informações oficiais do db4o (DB4OBJECTS, 2008), o uso deste banco de objetos permite facilmente o armazenamento de estruturas altamente complexas de dados, atingindo ótimo desempenho de performance em relação aos bancos de dados objeto-relacionais existentes no mercado: o db4o pode ser executado até 44 vezes mais rápido que o MySQL com Hibernate, pois economiza cerca de 90% do custo de desenvolvimento da camada de persistência e agiliza a comercialização em um período de tempo até 10% menor, reduzindo o esforço da equipe de administração de banco de dados.

5 CONCLUSÃO

Neste trabalho, foram descritas algumas etapas da metodologia proposta por Borba em 2006 para aplicação de modelos dimensionais na UML e o desenvolvimento destes em um ambiente orientado a objetos, utilizando a persistência de dados. Focando-se no processo de modelagem de dados, pois as etapas anteriores às aplicadas centralizam-se na análise de requisitos de negócio, um modelo dimensional real foi escolhido para o mapeamento de seus elementos em um diagrama de classes da UML e implementação em uma linguagem de programação OO integrada a um banco de objetos totalmente compatível com a plataforma utilizada. A utilização das técnicas formuladas pela metodologia proporcionou bons resultados que refletiram alto desempenho e baixo custo de desenvolvimento frente às tecnologias utilizadas atualmente para bancos de dados, incluindo maior uso da abstração junto ao cliente e melhoria da semântica de negócio promovida pela união dos ideais da UML e dos conceitos de BI.

Existem hoje no mercado diversas soluções que convergem seus esforços sobre a tecnologia objeto-relacional a qual não suporta integralmente os conceitos da OO. Dependendo do negócio modelado, estes conceitos devem ser aplicados para apresentar aos envolvidos uma estrutura de armazenamento e processamento mais próxima da semântica desejada. Esses conceitos, quando necessários, são modelados de acordo com a abordagem relacional e devem ser controlados no nível de interface com o usuário, exigindo retrabalho no ambiente de produção com alto esforço de mapeamento dos objetos na aplicação para as entidades do modelo relacional. Mesmo que, em um primeiro momento, a tecnologia objeto-relacional seja utilizada para carregar os dados do ambiente operacional, supostamente relacional, como foi o caso do teste de execução implementado neste trabalho, as próximas manipulações dos dados no DW seriam somente sobre o BDOO modelado. Quando utilizado um modelo OO visando a sua implementação em ambientes e bancos de dados, as características e conceitos oferecidos pelo paradigma OO são preservados e aplicados de forma direta entre o banco e as aplicações, incluindo as cargas de dados ocorridas dentro do DW e as consultas OLAP realizadas pelos usuários nos sistemas gerenciais.

Sob uma visão analítica, é perceptível que a metodologia discutida abrange todas as etapas usuais para um processo de desenvolvimento de DW descritas por Kimball muito utilizadas atualmente, mas seu principal destaque é a fundamentação no paradigma da OO. Como a programação OO está cada vez mais presente no desenvolvimento de softwares, recentemente alcançando até as linguagens do ambiente *web*, e nenhuma organização sobrevive sem a implantação e gerência de um banco de dados, é notável a compatibilidade da abordagem OO para a modelagem de bancos de objetos.

Ainda sobre esta discussão, mas agora com uma visão mais prática, estas mesmas organizações, com o rápido crescimento do seu volume de dados, vêm implantando soluções de BI buscando por diferencial competitivo. Dessa forma, pela dinâmica do mercado, é interessante que as empresas possuam uma metodologia bem definida e ágil para a implantação de soluções inteligentes no gerenciamento de dados do ambiente de DW modelado. A metodologia discutida neste trabalho, como as adaptações realizadas, projeta um único processo de conceituação, definição e implementação, utilizando uma única abordagem e promovendo baixos custos de desenvolvimento e administração com ferramentas muito difundidas no mercado. A partir dos conceitos abordados e a estrutura gerada para a realização do teste de desenvolvimento (APENDICE B) realizado com o modelo dimensional CSB mantido pela Core Synesis, a modelagem para BDOO apresenta-se como uma opção viável para a modelagem dimensional, permitindo o tratamento de objetos complexos, aumento da reusabilidade de código e desempenho compatível com os modelos relacionais.

Conforme as análises acima, este trabalho possui como contribuições a descrição do processo de implementação de modelos dimensionais relacionando os conceitos do paradigma OO sobre seus elementos; a melhoria da representatividade da semântica de negócio conforme a visão do cliente; a aplicação do processo de codificação do ambiente com a linguagem de programação OO Java; e detalha o funcionamento dos principais métodos e mecanismos do framework de persistência de objetos db4o, mostrando-se um banco muito promissor como relatado na revisão de literatura sobre as experiências reais de seu uso em grandes empresas.

Para trabalhos futuros, seriam interessantes pesquisas para o aperfeiçoamento das interfaces de gerenciamento de dados em BDOO, a

implementação de sistemas para gerar automaticamente o diagrama de classes e o código da linguagem OO escolhida a partir de um modelo dimensional dado e, principalmente, o desenvolvimento de aplicações OLAP e ETL segundo o paradigma OO com total suporte para o db4o. Dessa forma, como os mercados de e-commerce e de BI encontram-se em expansão, sugere-se também a adaptação do framework db4o para as linguagens de programação OO da web e a aplicação das etapas da metodologia descritas neste trabalho para a implementação de ambientes DW ou Data Marts sobre os assuntos de negócio modelados para o mundo virtual.

Em suma, este trabalho apresentou etapas de uma metodologia para a implantação de ambientes de DW quando os modelos dimensionais, uma vez modelados, são implementados seguindo definições do paradigma OO. Esta fase é realizada em sistemas que melhoram a representação da semântica do negócio pela UML e permitem seu desenvolvimento de forma ágil com a utilização de soluções de programação comercialmente difundidas. O db4o surge como uma alternativa de integração a estes sistemas para a implementação e gerenciamento de dados em objetos persistentes. A representação do modelo dimensional através do diagrama de classes fornece um ambiente natural de modelagem OO, para posterior implementação conforme os padrões de desenvolvimento vigentes.

REFERÊNCIAS

BARBIERI, Carlos. **“BI - Business Intelligence – Modelagem & Tecnologia”**. Rio de Janeiro (RJ): Axcel Books, 2006. 424 p. ISBN 8573231483.

BEZERRA, Eduardo. **“Princípios de Análise e Projeto de Sistemas com UML”**. Rio de Janeiro (RJ): Elsevier: Campus, 2007. 2ª ed. 369 p. ISBN 8535216960.

BORBA, Sueli de Fátima Poppi. **“Metodologia para implantação de modelos dimensionais em banco de dados orientado a objetos”**. Florianópolis (SC), 2006. 228p. Tese (Doutorado em Engenharia de Produção) - Universidade Federal de Santa Catarina - UFSC. Disponível em: <http://www.tede.ufsc.br/teses/PEPS5029.pdf> - Acesso em janeiro de 2009.

BORBA, Sueli de Fátima Poppi; MORALES, Aran Bey Tcholakian. **“Aplicação de Banco de Dados Orientado a Objetos na Modelagem Multidimensional”**. Florianópolis (SC), 2006. XXI Simpósio Brasileiro de Banco de Dados - SBBD. Disponível em: <http://www.lbd.dcc.ufmg.br:8080/colecoes/sbbd/2006/010.pdf> - Acesso em janeiro de 2009.

BOSCARIOLI, Clodis; BEZERRA, Anderson; BENEDICTO, Marcos de; DELMIRO, Gilliard. **“Uma reflexão sobre Banco de Dados Orientados a Objetos”**. Ponta Grossa (PR), 2006. IV Congresso de Tecnologias para Gestão de Dados e Metadados do Cone Sul - CONGED. Universidade Estadual de Ponta Grossa. Disponível em: <http://conged.deinfo.uepg.br/artigo4.pdf> - Acesso em janeiro de 2009.

CATTELL, Rick. G. G.; BARRY, Douglas K.; BERLER, Mark; EASTMAN, Jeff; JORDAN, David; RUSSELL, Craig; SCHADOW, Olaf; STANIENDA, Torsten; VELEZ, Fernando. **“The Object Data Standard: ODMG 3.0”**. San Francisco (USA), 2000. Grupo de Gestão de Objetos - Objects Management Group - OMG. Disponível em: <http://www.omg.org/docs/omg/04-07-02.pdf> - Acesso em janeiro de 2009.

CHEN, Peter Pin-Shan. **“The Entity-Relationship Model - Toward a Unified View of Data”**. ACM Transactions on Database Systems, páginas 9 a 36. Disponível em: <http://bit.csc.lsu.edu/~chen/chen.html> - Acesso em janeiro de 2009.

Core Synesis. Disponível em: <http://www.coresynesis.com.br> - Acesso em dezembro de 2008.

COUGO, Paulo. **“Modelagem conceitual e projeto de banco de dados”**. Rio de Janeiro (RJ): Campus, 1997. 1ª ed. 296 p. ISBN 8535201580.

DB4OBJECTS. **“db4o - Banco de objetos de código aberto”**. Disponível em: [https://www.db4o.com/portugues/db4o%20Product%20Information%20V5.0\(Portuguese\).pdf](https://www.db4o.com/portugues/db4o%20Product%20Information%20V5.0(Portuguese).pdf) - Acesso em dezembro de 2008.

ELMASRI, Ramez; NAVATHE, Sham. **“Sistemas de banco de dados: fundamentos e aplicações”**. Rio de Janeiro (RJ): LTC, 2006. 3ª ed. 837 p. ISBN 852161313X.

FREITAS JÚNIOR, Olival de Gusmão; PACHECO, Roberto C. S.; BARBOSA, Daniel M.; TODESCO, José L. **“Abordando o Uso da Orientação a Objetos em um Sistema de Data Warehouse”**. Itajaí (SC), 2002. II Congresso Brasileiro de Computação - CBCOMP. Disponível em: <http://200.169.53.89/download/CD%20congressos/2002/2%20CBComp/html/artigos/banco%20de%20dados/bcd008.pdf> - Acesso em janeiro de 2009.

HEUSER, Carlos Alberto. **“Projeto de Banco de Dados”**. Porto Alegre (RS): Sagra Luzzatto, 2004. Número 4. 4ª ed. 232 p. ISBN 8524105909.

INMON, William H. **“Building the Data Warehouse”**. Indianapolis (USA): Wiley Computer Publishing, 2005. 4ª ed. 543 p. ISBN 0764599445.

KIMBALL, Ralph. **“Data Warehouse Toolkit”**. Tradução de Mônica Rosemberg. São Paulo (SP): Makron Books, 1998. 388 p. ISBN 8534608172.

KIMBALL, Ralph; ROSS, Margy. **“The Data Warehouse Toolkit”**. Indianapolis (USA): Wiley Computer Publishing, 2002. 2ª ed. 421 p. ISBN 0471200247.

KIMBALL, Ralph; ROSS, Margy; REEVES, Laura; THORNTHWAITE, Warren. **“Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses”**. New York (USA): John Wiley & Sons, 1998. 2ª ed. 405 p. ISBN 0470149775.

KLEIN, Lawrence Zordam. **“A Tecnologia Relacional-Objeto em Ambientes de Data Warehouse”**. Rio de Janeiro (RJ), 1999. 191p. Teste (Mestrado em Ciências em Sistemas e Computação) – Instituto Militar de Engenharia. Disponível em: <http://dataware.nce.ufrj.br:8080/dataware/publicacoes/dataware/fisico/teses/datawarehousing/KLEIN-1999.pdf> - Acesso em janeiro de 2009.

KLEIN, Lawrence Zordam; CAMPOS, Maria Luiza Machado; TANAKA, Astério Kiyoshi. **“A Tecnologia Objeto-Relacional em Ambientes de Data Warehouse: Uso de Séries Temporais como Tipo de Dado Não Convencional”**. Florianópolis (SC), 1999. XIV Simpósio Brasileiro de Banco de Dados - SBBD. Disponível em: <http://www.inf.ufsc.br/sbbd99/anais/SBBD-Completo/30.PDF> - Acesso em janeiro de 2009.

LOPES, Maurício Capobianco; OLIVEIRA, Percio Alexandre de. **“Ferramenta de Construção de Data Warehouse”**. Blumenau (SC), 2007. XVI Seminário de Computação. Disponível em: www.inf.furb.br/seminco/2007/artigos/12_35427.pdf - Acesso em janeiro de 2009

MACHADO, Felipe Nery Rodrigues; ABREU, Maurício Pereira de. **“Projeto de Banco de Dados - Uma visão prática”**. São Paulo (SP): Érica, 2004. 11ª ed. 299 p. ISBN 8571943125.

PATERSON, Jim. EDILCH, Stefan. HÖRNING, Henrik. HÖRNING, Reidar - **“The Definitive Guide to bd4o”** - Nova Iorque (EUA): Apress, 2006. 484 p. ISBN 1590596560.

PINHEIRO, Hedenir Monteiro. **“Tópicos de Pesquisa em Data Warehouse”**. Goiânia (GO), 2003. 57 p. Monografia (Especialização em Banco de Dados). Instituto de Informática - Universidade Federal de Goiás - UFG. Disponível em: <http://www.inf.ufg.br/~juliano/ensino/especializacao/cursobd/2003/bdnaconvencional/Monografia.pdf> - Acesso em janeiro de 2009.

ROSEMBERG, Dave. **“INDRA: Sistema de Missão Crítica para controle de trens de alta velocidade”**. Tradução de Cássio R. Eskelsen. Disponível em: [https://www.db4o.com/portugues/db4o%20Success%20Story%20-%20INDRA%20Sistemas\(Portuguese\).pdf](https://www.db4o.com/portugues/db4o%20Success%20Story%20-%20INDRA%20Sistemas(Portuguese).pdf) - Acesso em dezembro de 2008.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. **“The Unified Modeling Language Reference Manual”**. Boston (EUA): Addison-Wesley, 2004. 2ª ed. 721 p. ISBN 0321245628.

SILVA, Alberto Manuel Rodrigues da; VIDEIRA, Carlos Alberto Escaleira. **“UML, Metodologias e Ferramentas CASE”**. Porto (Portugal): Centro Atlântico, 2001. 552 p. ISBN 9728426364.

SUN MICROSYSTEMS. Disponível em: <http://www.sun.com/> - Acesso em janeiro de 2009

VIDOTTI, Julio Cesar. **“Projeto de um Data Warehouse: Análise de Custo/Benefício”**. Cuiabá (MT), 2001. 36 p. Monografia (Bacharelado em Ciência da Computação). Universidade Federal do Mato Grosso - UFMT. Disponível em: <http://www.ufmt.br/cacomp/Downloads/monografias/ProjetoDataWareHouse.pdf> - Acesso em janeiro de 2009.

VIEIRA, Marina Teresa Pires. **“Banco de Dados Orientado a Objetos”**. São Carlos (SP), 2001. 60 p. Universidade Federal de São Carlos - UFSCAR. Disponível em: <http://www.recope.dc.ufscar.br/recope/download/bd/apostilabdoo.pdf> - Acesso em janeiro de 2009.

APÊNDICES

APÊNDICE A – Implementação das classes do diagrama dimensional CSB

Classe de Dimensão CDTempoAnoMes:

```
import java.util.ArrayList;

public class CDTempoAnoMes {

    private int codAnoMesReferencia;
    private int codMesReferencia;
    private String dscAnoMesReferencia;
    private int codAnoQuadrimestreRef;
    private int codAnoTrimestreReferencia;
    private int codAnoReferencia;
    private int qtdDiasUteisMes;
    private String dscMesReferencia;
    private int qtdDiasUteisRestanteAno;
    private String dscAnoMesAbreviado;
    private int qtdDiasTotaisMes;
    private ArrayList<CSCSB> listaFatos;

    public CDTempoAnoMes(int arg1, int arg2, String arg3, int arg4, int arg5,
        int arg6, int arg7, String arg8, int arg9, String arg10, int arg11) {
        this.codAnoMesReferencia = arg1;
        this.codMesReferencia = arg2;
        this.dscAnoMesReferencia = arg3;
        this.codAnoQuadrimestreRef = arg4;
        this.codAnoTrimestreReferencia = arg5;
        this.codAnoReferencia = arg6;
        this.qtdDiasUteisMes = arg7;
        this.dscMesReferencia = arg8;
        this.qtdDiasUteisRestanteAno = arg9;
        this.dscAnoMesAbreviado = arg10;
        this.qtdDiasTotaisMes = arg11;
        this.listaFatos = new ArrayList<CSCSB>();
    }

    public int getCodAnoMesReferencia() {
        return codAnoMesReferencia;
    }

    public int getCodAnoQuadrimestreRef() {
        return codAnoQuadrimestreRef;
    }

    public int getCodAnoReferencia() {
        return codAnoReferencia;
    }

    public int getCodAnoTrimestreReferencia() {
        return codAnoTrimestreReferencia;
    }

    public int getCodMesReferencia() {
        return codMesReferencia;
    }

    public String getDscAnoMesAbreviado() {
        return dscAnoMesAbreviado;
    }

    public String getDscAnoMesReferencia() {
        return dscAnoMesReferencia;
    }

    public String getDscMesReferencia() {
        return dscMesReferencia;
    }

    public int getQtdDiasTotaisMes() {
        return qtdDiasTotaisMes;
    }

    public int getQtdDiasUteisMes() {
        return qtdDiasUteisMes;
    }

    public int getQtdDiasUteisRestanteAno() {
```

```

        return qtdDiasUteisRestanteAno;
    }
    public void add(CSCSB credito) {
        this.listaFatos.add(credito);
    }
}

```

Classe de Dimensão CDTipoContato:

```

import java.util.ArrayList;

public class CDTipoContato {

    private int codTipoContato;
    private String indAtivCartao;
    private String dscTipoContato;
    private String sglTipoContato;
    private String codCanalContato;
    private ArrayList<CSCSB> listaFatos;

    public CDTipoContato(int arg1,String arg2,String arg3,String arg4,
String arg5){
        this.codTipoContato = arg1;
        this.indAtivCartao = arg2;
        this.dscTipoContato = arg3;
        this.sglTipoContato = arg4;
        this.codCanalContato = arg5;
        this.listaFatos = new ArrayList<CSCSB>();
    }
    public int getCodTipoContato() {
        return codTipoContato;
    }
    public String getIndAtivCartao() {
        return indAtivCartao;
    }
    public String getDscTipoContato() {
        return dscTipoContato;
    }
    public String getSglTipoContato() {
        return sglTipoContato;
    }
    public String getCodCanalContato() {
        return codCanalContato;
    }
    public void add(CSCSB credito) {
        this.listaFatos.add(credito);
    }
}

```

Classe de Dimensão CDOperadora:

```

import java.util.ArrayList;

public class CDOperadora {

    private int codOperadora;
    private int codTipoAssinatura;
    private int codCompanhia;
    private String nomLocalidade;
    private int numSeqPlano;
    private String nomTipoSegmento;
    private String nomOperadora;
    private String sglOperadora;
    private String sqlUFOperadora;
    private int codTecnologia;
    private String codDDDAcesso;
    private ArrayList<CSCSB> listaFatos;

    public CDOperadora(int arg1, int arg2, int arg3, String arg4, int arg5,
String arg6, String arg7, String arg8, String arg9, int arg10, String arg11){
        this.codOperadora = arg1;
    }
}

```

```

        this.codTipoAssinatura = arg2;
        this.codCompanhia = arg3;
        this.nomLocalidade = arg4;
        this.numSeqPlano = arg5;
        this.nomTipoSegmento = arg6;
        this.nomOperadora = arg7;
        this.sglOperadora = arg8;
        this.sqlUFOperadora = arg9;
        this.codTecnologia = arg10;
        this.codDDDAcesso = arg11;
        this.listaFatos = new ArrayList<CSCSB>();
    }
    public int getCodOperadora() {
        return codOperadora;
    }
    public int getCodTipoAssinatura() {
        return codTipoAssinatura;
    }
    public int getCodCompanhia() {
        return codCompanhia;
    }
    public String getNomLocalidade() {
        return nomLocalidade;
    }
    public int getNumSeqPlano() {
        return numSeqPlano;
    }
    public String getNomTipoSegmento() {
        return nomTipoSegmento;
    }
    public String getNomOperadora() {
        return nomOperadora;
    }
    public String getSglOperadora() {
        return sglOperadora;
    }
    public String getSqlUFOperadora() {
        return sqlUFOperadora;
    }
    public int getCodTecnologia() {
        return codTecnologia;
    }
    public String getCodDDDAcesso() {
        return codDDDAcesso;
    }
    public void add(CSCSB credito) {
        this.listaFatos.add(credito);
    }
}

```

Classe de Dimensão CDAgrupamento:

```

import java.util.ArrayList;

public class CDAgrupamento {

    private int codAgrupamento;
    private int codTipoAgrupamento;
    private String dscAgrupamento;
    private ArrayList<CSCSB> listaFatos;

    public CDAgrupamento(int arg1, int arg2, String arg3) {
        this.codAgrupamento = arg1;
        this.codTipoAgrupamento = arg2;
        this.dscAgrupamento = arg3;
        this.listaFatos = new ArrayList<CSCSB>();
    }

    public int getCodAgrupamento() {
        return codAgrupamento;
    }

    public int getCodTipoAgrupamento() {
        return codTipoAgrupamento;
    }
}

```

```

    }

    public String getDscAgrupamento() {
        return dscAgrupamento;
    }

    public void add(CSCSB credito) {
        this.listaFatos.add(credito);
    }
}

```

Super-Classe de Fatos CSCSB:

```

public abstract class CSCSB {

    private double vlrCredito;
    private String indFormaPagto;
    private String codDDDAcessoPre;

    public CSCSB(double arg1, String arg2, String arg3) {
        this.vlrCredito = arg1;
        this.indFormaPagto = arg2;
        this.codDDDAcessoPre = arg3;
    }

    public double getVlrCredito() {
        return vlrCredito;
    }

    public String getIndFormaPagto() {
        return indFormaPagto;
    }

    public String getCodDDDAcessoPre() {
        return codDDDAcessoPre;
    }

    public abstract String toString();
}

```

Classe de Fatos CSAdesaoCSB:

```

public class CSAdesaoCSB extends CSCSB {

    private CDTempoAnoMes tempoAnoMes;
    private CDTipoContato tipoContato;
    private CDOperadora operadora;
    private CDAgrupamento agrupamento;
    private static int qtdAdesaoAgendamento = 0;
    private static int qtdAdesaoCliente = 0;

    public CSAdesaoCSB(int arg1, String arg2, String arg3, CDTempoAnoMes arg4,
        CDTipoContato arg5, CDOperadora arg6, CDAgrupamento arg7, int tipo) {
        super(arg1, arg2, arg3);
        tempoAnoMes = arg4;
        tipoContato = arg5;
        operadora = arg6;
        agrupamento = arg7;
        if (tipo == 1) {
            qtdAdesaoAgendamento++;
        } else {
            qtdAdesaoCliente++;
        }
    }

    public CDTempoAnoMes getTempoAnoMes() {
        return tempoAnoMes;
    }

    public CDTipoContato getTipoContato() {
        return tipoContato;
    }

    public CDOperadora getOperadora() {
        return operadora;
    }

    public CDAgrupamento getAgrupamento() {
        return agrupamento;
    }
}

```



```

public int getQtdAdesaoAgendamento() {
    return qtdAdesaoAgendamento;
}
public int getQtdAdesaoCliente() {
    return qtdAdesaoCliente;
}
public String toString() {
    return "\nCódAnoMesReferencia: " + tempoAnoMes.getCodAnoMesReferencia() +
        "\nDscTipoContato: " + tipoContato.getDscTipoContato() +
        "\nNomOperadora: " + operadora.getNomOperadora() +
        "\nDscAgrupamento: " + agrupamento.getDscAgrupamento() +
        "\nQtdAdesaoAgendamento: " + qtdAdesaoAgendamento +
        "\nQtdAdesaoCliente: " + qtdAdesaoCliente +
        "\nVlrCredito: " + super.getVlrCredito() +
        "\nIndAtivCartao: " + super.getIndFormaPagto() +
        "\nCódDDDAcessoPre: " + super.getCódDDDAcessoPre();
}
}

```

Classe de Fatos CSCancelamentoCSB:

```

public class CSCancelamentoCSB extends CSCSB {

    private CDTempoAnoMes tempoAnoMes;
    private CDTipoContato tipoContato;
    private CDOperadora operadora;
    private CDAgrupamento agrupamento;
    private static int qtdCancelamentoAgend = 0;
    private static int qtdCancelamentoCliente = 0;

    public CSCancelamentoCSB(int arg1, String arg2, String arg3, CDTempoAnoMes arg4,
        CDTipoContato arg5, CDOperadora arg6, CDAgrupamento arg7, int tipo) {
        super(arg1, arg2, arg3);
        this.tempoAnoMes = arg4;
        this.tipoContato = arg5;
        this.operadora = arg6;
        this.agrupamento = arg7;
        if (tipo == 3) {
            qtdCancelamentoAgend++;
        } else {
            qtdCancelamentoCliente++;
        }
    }

    public CDTempoAnoMes getTempoAnoMes() {
        return tempoAnoMes;
    }

    public CDTipoContato getTipoContato() {
        return tipoContato;
    }

    public CDOperadora getOperadora() {
        return operadora;
    }

    public CDAgrupamento getAgrupamento() {
        return agrupamento;
    }

    public int getQtdCancelamentoAgend() {
        return qtdCancelamentoAgend;
    }

    public int getQtdCancelamentoCliente() {
        return qtdCancelamentoCliente;
    }

    public String toString() {
        return "\nCódAnoMesReferencia: " + tempoAnoMes.getCodAnoMesReferencia() +
            "\nDscTipoContato: " + tipoContato.getDscTipoContato() +
            "\nNomOperadora: " + operadora.getNomOperadora() +
            "\nDscAgrupamento: " + agrupamento.getDscAgrupamento() +
            "\nQtdCancelamentoAgend: " + qtdCancelamentoAgend +
            "\nQtdCancelamentoCliente: " + qtdCancelamentoCliente +
            "\nVlrCredito: " + super.getVlrCredito() +
            "\nIndAtivCartao: " + super.getIndFormaPagto() +
            "\nCódDDDAcessoPre: " + super.getCódDDDAcessoPre();
    }
}

```

APÊNDICE B – Implementação dos testes de carga e consulta do BDOO

Classe Principal para testes de execução:

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import oracle.jdbc.pool.OracleDataSource;
import com.db4o.Db4o;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.query.Constraint;
import com.db4o.query.Query;

public class Principal {

    public static void processoStagingProd() {

        ObjectContainer dwdsv = Db4o.openFile("C:/DWDSV/DWDSV.yap");
        try {
            dwdsv.store(new CDTempoAnoMes(200812,12,"DEZEMBRO DE 2008",
                200803,200804,2008,22,"DEZEMBRO",0,"2008DEZ",31));
            dwdsv.store(new CDTempoAnoMes(200901,01,"JANEIRO DE 2009",
                200901,200901,2009,21,"JANEIRO",233,"2009JAN",31));
            dwdsv.store(new CDTipoContato(1,"SIM","ATENDIMENTO ELETRONICO",
                "URA","1054"));
            dwdsv.store(new CDTipoContato(2,"SIM","LOJA PROPRIA","LJP","LJPAP"));
            dwdsv.store(new CDOperadora(10,2,1,"BELO HORIZONTE",15,"ULTRA",
                "Vivo","Vivo","MG",3,"031"));
            dwdsv.store(new CDOperadora(10,2,2,"JUIZ DE FORMA",12,"ESPECIAL",
                "Vivo","Vivo","MG",2,"032"));
            dwdsv.store(new CDAgrupamento(1,2,"CORPORATIVO"));
            dwdsv.store(new CDAgrupamento(2,2,"SERVIÇOS I"));
        } catch (Exception e) {
            System.out.println("Erro na manipulacao do DW em Staging.");
        } finally {
            dwdsv.close();
        }
    }

    public static void processoDMVendasAdesao() {

        ObjectContainer dwdsv = Db4o.openFile("C:/DWDSV/DWDSV.yap");
        String url = "jdbc:oracle:thin:@localhost:1521/XE";
        Constraint restrição;
        /* Parâmetros incluídos pelo desenvolvedor conforme a semântica da demanda ETL */
        int restricaoServico = 1;
        int restricaoTempo = 200812;
        int restricaoOperadora = 10;
        String restricaoDDD = "031";

        String select = "SELECT CS.VLR_CREDITO VLR_CREDITO, " +
            " CS.DSC_FORMA_PAGTO DSC_FORMA_PAGTO, " +
            " SUBSTR (CS.NUM_BENEFICIADO,1,3) NUM_PRE_DDD, " +
            " CS.COD_CONTATO COD_CONTATO, " +
            " PL.COD_TIPO COD_AGRUPAMENTO, " +
            " CS.COD_TIPO_SOL COD_SOLICITACAO " +
            "FROM CLIENTE CL, " +
            " CLIENTE_SERVICO CS, " +
            " CLIENTE_PLANO CP, " +
            " PLANO PL " +
            "WHERE CL.COD_CLIENTE = CS.COD_CLIENTE " +
            " AND CL.COD_CLIENTE = CP.COD_CLIENTE " +
            " AND PL.COD_PLANO = CP.COD_PLANO " +
            " AND CS.NUM_BENEFICIADO IS NOT NULL " +
            " AND CS.COD_SERVICO = " + restricaoServico +
            " AND SUBSTR (CL.NUM_CELULAR,1,3) = '" + restricaoDDD + "'" +
            " AND TO_NUMBER (" +
            " TO_CHAR(CS.DATA_INICIO,'YYYYMM') " +
            " ) = " + restricaoTempo;

        try {
            OracleDataSource dbREL = new OracleDataSource();
            dbREL.setURL(url);
            Connection conexao = dbREL.getConnection("SYSTEM", "DBDSV");
```

```

Statement transacao = conexao.createStatement();
ResultSet queryREL = transacao.executeQuery(select);

while (queryREL.next()) {
    double valorCredito = queryREL.getDouble("VLR_CREDITO");
    String formaPagamento = queryREL.getString("DSC_FORMA_PAGTO");
    String numeroPreDDD = queryREL.getString("NUM_PRE_DDD");
    int restricaoContato = queryREL.getInt("COD_CONTATO");
    int restricaoAgrupamento = queryREL.getInt("COD_AGRUPAMENTO");
    int restricaoSolicitacao = queryREL.getInt("COD_SOLICITACAO");

    Query query = dwdsv.query();
    query.constrain(CDTempoAnoMes.class);
    query.descend("codAnoMesReferencia").constrain(restricaoTempo);
    CDTempoAnoMes tempo = (CDTempoAnoMes) query.execute().next();

    query = dwdsv.query();
    query.constrain(CDTipoContato.class);
    query.descend("codTipoContato").constrain(restricaoContato);
    CDTipoContato contato = (CDTipoContato) query.execute().next();

    query = dwdsv.query();
    query.constrain(CDOperadora.class);
    restricao = query.descend("codDDDAcesso").constrain(restricaoDDD);
    query.descend("codOperadora").constrain(restricaoOperadora)
        .and(restricao);
    CDOperadora operadora = (CDOperadora) query.execute().next();

    query = dwdsv.query();
    query.constrain(CDAgrupamento.class);
    query.descend("codAgrupamento").constrain(restricaoAgrupamento);
    CDAgrupamento agrupamento = (CDAgrupamento) query.execute().next();

    CSAdesaoCSB adesao = new CSAdesaoCSB(valorCredito, formaPagamento,
        numeroPreDDD, tempo, contato, operadora, agrupamento,
        restricaoSolicitacao);

    dwdsv.store(adesao);
    tempo.add(adesao);
    dwdsv.store(tempo);
    contato.add(adesao);
    dwdsv.store(contato);
    operadora.add(adesao);
    dwdsv.store(operadora);
    agrupamento.add(adesao);
    dwdsv.store(agrupamento);
}
conexao.close();
queryREL.close();
transacao.close();
} catch (Exception e) {
    System.out.println("Erro na manipulacao dos bancos em DMVendasAdesao.");
} finally {
    dwdsv.close();
}
}

public static void main(String[] args) {

    processoStagingProd();
    processoDMVendasAdesao();
    ObjectContainer dwdsv = Db4o.openFile("C:/DWDSV/DWDSV.yap");
    try {
        // Obtendo todos os fatos de CSB referentes às adesões
        ObjectSet<CSAdesaoCSB> adesao = dwdsv.queryByExample(CSAdesaoCSB.class);
        while (adesao.hasNext()) {
            System.out.println(adesao.next().toString());
        }
    } catch (Exception e) {
        System.out.println("Erro na manipulacao do DW em Producao.");
    } finally {
        dwdsv.close();
    }
}
}

```

APÊNDICE C – Modelagem ER de Base Operacional de Teste

Script SQL de criação e inserção de dados do banco operacional:

```
CREATE TABLE CLIENTE (
    COD_CLIENTE    NUMBER(5)      NOT NULL,
    NOM_CLIENTE    VARCHAR2(20)   NOT NULL,
    NUM_CELULAR    VARCHAR2(11)   NOT NULL,

    CONSTRAINT PK_CLIENTE PRIMARY KEY (COD_CLIENTE)
);

INSERT INTO CLIENTE VALUES (1000,'JOÃO SILVA','03199998877');
INSERT INTO CLIENTE VALUES (2000,'JOSÉ CARLOS','03197661100');
INSERT INTO CLIENTE VALUES (3000,'RUAN JULIO','03298991819');
INSERT INTO CLIENTE VALUES (4000,'FABIO SOUZA','03396559655');

CREATE TABLE TIPO_PLANO (
    COD_TIPO        NUMBER(2)      NOT NULL,
    NOM_TIPO        VARCHAR2(20)   NOT NULL,

    CONSTRAINT PK_TIPO PRIMARY KEY (COD_TIPO)
);

INSERT INTO TIPO_PLANO VALUES (1,'CORPORATIVO');
INSERT INTO TIPO_PLANO VALUES (2,'VAREJO');

CREATE TABLE PLANO (
    COD_PLANO       NUMBER(2)      NOT NULL,
    NOM_PLANO       VARCHAR2(20)   NOT NULL,
    COD_TIPO        NUMBER(2)      NOT NULL,

    CONSTRAINT PK_PLANO PRIMARY KEY (COD_PLANO),
    CONSTRAINT FK_TIPO_PLANO FOREIGN KEY (COD_TIPO)
        REFERENCES TIPO_PLANO (COD_TIPO)
);

INSERT INTO PLANO VALUES (1,'VIVO CONTA',1);
INSERT INTO PLANO VALUES (2,'VIVO CONTROLE',2);
INSERT INTO PLANO VALUES (3,'VIVO PRE',2);

CREATE TABLE CLIENTE_TEM_PLANO (
    COD_CLIENTE     NUMBER(5)      NOT NULL,
    COD_PLANO       NUMBER(5)      NOT NULL,

    CONSTRAINT PK_CLIENTE_PLANO PRIMARY KEY (COD_CLIENTE,COD_PLANO),
    CONSTRAINT FK_CLIENTE_PLANO FOREIGN KEY (COD_CLIENTE)
        REFERENCES CLIENTE (COD_CLIENTE),
    CONSTRAINT FK_PLANO_CLIENTE FOREIGN KEY (COD_PLANO)
        REFERENCES PLANO (COD_PLANO)
);

INSERT INTO CLIENTE_TEM_PLANO VALUES (1000,1);
INSERT INTO CLIENTE_TEM_PLANO VALUES (2000,2);
INSERT INTO CLIENTE_TEM_PLANO VALUES (3000,3);
INSERT INTO CLIENTE_TEM_PLANO VALUES (4000,3);

CREATE TABLE SERVICO (
    COD_SERVICO     NUMBER(5)      NOT NULL,
    NOM_SERVICO     VARCHAR2(30)   NOT NULL,

    CONSTRAINT PK_SERVICO PRIMARY KEY (COD_SERVICO)
);

INSERT INTO SERVICO VALUES (1,'CREDITO SEM BARREIRAS');
INSERT INTO SERVICO VALUES (2,'FALE FACIL');

CREATE TABLE CLIENTE_PEDA_SERVICO (
    COD_CLIENTE     NUMBER(5)      NOT NULL,
    COD_SERVICO     NUMBER(5)      NOT NULL,
    DATA_INICIO    DATE           NOT NULL,
    DATA_FINAL     DATE           NULL,
    VLR_CREDITO     NUMBER(5,2)    NOT NULL,
    DSC_FORMA_PAGTO VARCHAR2(20)   NOT NULL,
```

```

        COD_CONTATO      NUMBER(2)      NOT NULL,
        NUM_BENEFICIADO  VARCHAR2(11)   NULL,
        COD_TIPO_SOL     NUMBER(2)      NOT NULL,

        CONSTRAINT PK_CLIENTE_SERVICO PRIMARY KEY (COD_CLIENTE,COD_SERVICO),
        CONSTRAINT FK_CLIENTE_SERVICO FOREIGN KEY (COD_CLIENTE)
            REFERENCES CLIENTE (COD_CLIENTE),
        CONSTRAINT FK_SERVICO_CLIENTE FOREIGN KEY (COD_SERVICO)
            REFERENCES SERVICO (COD_SERVICO)
    );

INSERT INTO CLIENTE_PEDA_SERVICO VALUES (
    1000,1,'24-12-2008',NULL,35,'CARTAO VISA',1,'03298991819',1
);
INSERT INTO CLIENTE_PEDA_SERVICO VALUES (
    1000,2,'30-12-2008',NULL,50,'A VISTA',1,'03298991819',2
);
INSERT INTO CLIENTE_PEDA_SERVICO VALUES (
    2000,1,'10-12-2008',NULL,22,'A VISTA',2,'03396559655',2
);
INSERT INTO CLIENTE_PEDA_SERVICO VALUES (
    2000,2,'05-01-2009',NULL,15,'MASTERCARD',1,NULL,1
);


```

Autorização

1. Eu, Moisés Henrique Ramos Pereira, **portador da carteira de identidade** MG-10.426.602 **CPF** 013.976.896-33, autorizo a publicação em formato digital, sem ônus, da obra **Aplicação de Metodologia para Implementação de Modelo Dimensionais em Banco de Dados Orientado a Objetos** de minha autoria, pelo **Portal Domínio Público**, biblioteca digital do Ministério da Educação, no endereço de internet www.dominiopublico.gov.br. É de meu conhecimento que a publicação das obras na internet terá fins estritamente não-comerciais, permitindo a reprodução e a impressão gratuitas pelos usuários da biblioteca.

Belo Horizonte, 08 de maio de 2010

Ass.:


Nome: Moisés Henrique Ramos Pereira
CPF: 013.976.896-33
RG: MG-10.426.602