UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC

CURSO DE CIENCIA DA COMPUTAÇÃO

DANIEL GUESSI PLÁCIDO

IMPEDÂNCIA DE DADOS EM BANCO DE DADOS

CRICIÚMA, JULHO DE 2008

DANIEL GUESSI PLÁCIDO

IMPEDÂNCIA DE DADOS EM BANCO DE DADOS

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientador: Prof. MSc. Daniel Pezzi da Cunha

CRICIÚMA, JULHO DE 2008

FOLHA DE APROVAÇÃO

Aos meus pais, Tomaz de Aquino Plácido e Rosânia Guessi Plácido, meus irmãos, e a minha namorada Deisi Martinello Gomes.

AGRADECIMENTOS

A Deus, que iluminou meu caminho durante esta caminhada.

Aos meus familiares, especialmente meus pais e meus irmãos, pelo apoio, pelo amor, pela dedicação durante toda a minha vida.

A minha namorada, pelo carinho, amor, compreensão e por dicas importantes para desenvolvimento deste trabalho.

Ao meu orientador, Daniel Pezzi da Cunha, pela dedicação e empenho fornecido durante a realização deste trabalho.

Aos amigos e companheiros de turma, que compartilharam das mesmas preocupações, dúvidas e descobertas.

Enfim, a todos que me ajudaram em mais esta etapa.

“Só porque é possível empurrar pequenos galhos pelo chão com o nariz de alguém não quer necessariamente dizer que é a melhor maneira de coletar lenha”

(Anthony Berglas)

RESUMO

Nos dias de hoje os bancos de dados relacionais e pós-relacionais estão incontestavelmente no núcleo da corporação moderna. Técnicas, métodos, ferramentas e aplicações de desenvolvimento de sistemas usados nas empresas mudaram e evoluíram de forma incrível. Entretanto, com o uso de ferramentas nas organizações, alguns problemas foram surgindo. Dentre os quais, a incompatibilidade entre dados, intitulada Impedância de Dados. Esse problema é um desafio aos projetistas de sistemas e desenvolvedores em geral. Cada banco de dados possui suas próprias características, tornando a incompatibilidade entre sistema e banco de dados um dos um dos grandes problemas no desenvolvimento e na continuidade de um software. Tendo em vista estes pontos, neste trabalho objetivou-se principalmente estudar os problemas de mapeamento de dados entre bancos de dados orientados a objetos e relacionais, por meio de uma linguagem de programação orientada a objetos. Para auxílio da pesquisa foi utilizado o *framework* de mapeamento objeto relacional Hibernate. Foram construídos dois modelos de testes, um com uma aplicação orientada a objeto com bancos de dados relacionais (MySQL e PostGreSQL), utilizando-se do *framework* Hibernate afim de se realizar o mapeamento objeto relacional e outro com sistema orientado a objetos com bancos de dados orientado a objetos (Caché e DB4O) verificando o comportamento de padrões de objetos. Por fim este trabalho apresenta os resultados obtidos, diferenciando os modelos de bancos de dados e quais as melhores opções para se evitar o problema de impedância de dados.

Palavras chave: Impedância de Dados, Bancos de Dados Relacionais, Bancos de Dados Objetos.

ABSTRACT

Nowadays, the relational databases and post-relational are unquestionably in the core of the modern corporation. Techniques, methods, tools and applications for development of systems used in companies have changed and evolved so incredible. However, with the use of tools in organizations, some problems were emerging. Among them the incompatibility of data, entitled Impedance Data. This problem is a challenge to designers of systems and developers in general. Each database has its own characteristics, making the incompatibility between system and a database of one of the major problems in the development and continuity of software. In view of these points, this work is primarily aimed to study the problems of mapping of data between the databases and object-oriented relational, through a programming language geared to objects. To aid the research was used object relational mapping framework Hibernate. They were built two types of tests, one with a targeted application with the object relational databases (MySQL and PostGreSQL), using the Hibernate framework in order to achieve the object relational mapping and other objects with the target system with data banks oriented to objects (Caché and DB4O) checking the behavior patterns of objects. Finally this work presents the results, differentiating the models of databases and what the best options to avoid the problem of impedance data.

Key words: Impedance Data, Relational Database, Database Objects.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ANSI	American National Standards Institute
API	Application Programming Interface
BD	Banco de Dados
BDOO	Banco de Dados Orientado a Objetos
COBOL	Common Business Oriented Language
CODASYL	Committee for Data Systems Languages
CRUD	Creat, Retrieve, Update, Delete
DBTG	Data-Base Task Group
DEC	Digital Equipament Corporation
GE	General Eletric
HQL	Hibernate Query Language
IBM	International Business Machines
IDS	Information Data System
IMS	Information Management System
JDO	Java Data Objects
LGPL	Lesser General Public License
OBJ	Object Relational Bridge
ODL	Object Definition Language
OID	Object Identifier
OO	Orientação a Objetos
OQL	Object Query Language
SBD	Sistema de Banco de Dados
SBDOO	Sistema Banco de Dados Orientado a Objetos

SGBD	Sistema Gerenciador de Banco de Dados
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SPARC	Systems Planning and Requirements Committee
SQL	Structured Query Language
TMDM	Transactional Multidimensional Database Management
WWW	World Wide Web
XML	Extended Markup Language

SUMARIO

1 INTRODUÇÃO

Os bancos de dados atuais deixaram de ter características concentradas para se adaptarem a ambientes distribuídos, de forma a atender o maior número de usuários possíveis.

Segundo Silberschatz e Korth (1995) o controle de banco de dados evoluiu de uma aplicação específica de computador para um componente central e moderno da computação. Os sistemas de banco de dados têm se tornado parte fundamental de pesquisa da ciência computacional.

Estas mudanças se tornaram visíveis na última década e em uma velocidade nunca antes experimentada. Segundo Machado e Abreu (1996) a era da informação não só surgiu como já faz parte da vida diária dos usuários sendo que muitas vezes esta evolução não é percebida, assim como as mudanças no perfil das pessoas e das organizações.

Nos dias de hoje os bancos de dados relacionais e pós-relacionais estão incontestavelmente no núcleo da corporação moderna. Técnicas, métodos, ferramentas e aplicações de desenvolvimento de sistemas usados nas empresas, mudaram e evoluíram de forma incrível. Com o reconhecimento de que informações são recursos importantes na vida de uma organização, as aplicações de banco de dados ganharam técnicas de acordo com o tipo de dado usado pela organização, atendendo necessidades específicas. Com o tempo e o uso destas ferramentas nas organizações, alguns problemas foram surgindo. Dentre os quais, a incompatibilidade entre dados é um dos principais problemas encontrados desde o inicio do surgimento dos bancos de dados. Muitas das vezes as linguagens de programação usam características incompatíveis com o banco de

dados adotado pela equipe de desenvolvimento. Este problema é chamado de impedância de dados em banco de dados.

A impedância refere-se à diferença existente entre os bancos de dados orientados a objetos e os modelos relacionais. O modelo relacional organiza os dados em tuplas e atributos, sendo que cada tupla representa um cadastro na relação. Já o modelo orientado a objetos é composto por classes e objetos. Ao usar um banco de dados relacional, em uma linguagem orientada a objetos como o Java ou o C++, a classe de objetos da linguagem deve ser traduzida para as tabelas dispares de um banco de dados relacional. Enquanto linguagens de programação a objetos como Java fornece uma visão de orientação a objetos e de entidades, onde as regras de negócio são especificadas no nível de aplicação, a inserção em banco de dados respectivos são relacionais por natureza (FINN, 2002).

Desta forma se faz necessário verificar quais os modelos mais eficientes no tratamento da impedância de banco de dados com análise comparativa entre os modelos relacional e orientado a objetos, utilizando bancos de dados gratuitos ou de licença gratuita para estudo e pesquisa.

1.1 OBJETIVO GERAL

Estudar os problemas de mapeamento de dados entre sistemas orientados a objetos e de modelos de bancos de dados mais utilizados na atualidade.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho consistem em:

a) compreender o funcionamento dos modelos de bancos de dados relacional e orientado a objetos;
b) analisar o comportamento dos bancos de dados ao encontrar um problema de impedância com um sistema orientado a objetos;
c) disponibilizar uma análise comparativa de comportamento entre os bancos de dados testados;
d) apresentar técnicas para redução de impedância de dados.

1.3 JUSTIFICATIVA

Nos últimos anos, as linguagens Java, C++ e COM, tornaram-se as tecnologias dominantes para desenvolvimento de sistemas. Assim os desenvolvedores de aplicações precisam ser capazes de representarem dados como objetos, independente da tecnologia de banco de dados adotada. A diferença de impedância não pode ser evitada, mas pode ser reduzida conforme a linguagem e a tecnologia de banco de dados adotada (FINN, 2002).

Tentativas de se criar uma forma de comunicação de padrões entre a tecnologia de objetos e o modelo relacional sempre foi um desafio da computação corporativa hoje em dia. Um dos projetos de pesquisas mais ambiciosos de se mapear objetos em atributos resultou no *framework* Hibernate que fornece uma ligação entre essas tecnologias, de forma a atender todos os tipos de dados característicos dos banco de dados relacionais. O uso de banco de dados orientado a objetos com sistemas orientado a objetos também pode ser uma das soluções para resolver o problema de impedância de dados em bancos de dados ao se utilizar uma linguagem orientada a objetos.

Durante toda a pesquisa serão documentadas informações referente aos tipos de banco de dados pesquisados, abordando características sobre a diferença de impedância de cada um e informações comparativas sobre o comportamento inerente do problema de impedância. Serão utilizados banco de dados para testes, verificando o comportamento de cada tipo com a aplicação orientada a objetos escolhida.

Com o projeto de pesquisa, pretende-se também abordar formas de amenizar as diferenças de impedância nos bancos de dados, afim de evitar problemas na manutenção de aplicações.

1.4 TRABALHOS CORRELATOS

O tema "impedância de dados" é um tema novo e muito divulgado atualmente. Mesmo assim existe pouca documentação sobre o assunto.

Apenas um trabalho foi encontrado sobre com o foco de padronização de dados. Este trabalho foi apresentado pelo acadêmico Felipe Cesar Stanzine Fonseca em defesa do graú de bacharel em Ciência da Computação pela Faculdades Associadas Espírito-Santenses, sobe o título de *Uma Solução de Projeto de Software para Comparar a Persistência dos Dados em Banco de Dados Relacional e OO* no ano de 2007. Neste projeto, Fonseca faz uma análise comparativa entre a persistência dos dados e suas padronizações por meio de uma *software* desenvolvido. Fonseca conclui que à medida que o nível de complexibilidade orientada a objeto aumenta em um sistema de objetos, persistir dados em Bancos de Dados Orientado a Objetos se torna mais fácil que em Bancos de dados Relacionais. Porém, é necessário ter profundo conhecimento de paradigmas orientados a objetos para tirar o máximo de proveito deste sistema.

1.5 ESTRUTURA DO TRABALHO

O trabalho foi organizado em cinco etapas.

Na primeira etapa há a introdução do trabalho, abordando os objetivos do sistema, especificando os possíveis problemas de impedância de dados nos banco de dados e justificando a criação do mesmo, relatado no Capítulo 1.

Na segunda etapa foi feito um estudo teórico sobre os modelos de bancos de dados, caracterizando cada estrutura e seu funcionamento, assim como o levantamento do problema de impedância de dados nos bancos de dados, envolvendo os bancos de dados escolhidos, distribuídos nos capítulos 2 e 3.

Na terceira etapa foi abordado os modelos banco de dados orientado a objetos e os tipos de objetos existentes, relatado no Capítulo 4.

Na quarta etapa foi feito testes com a linguagem de programação orientada a objetos Java, juntamente com o *framework* Hibernate e os bancos de dados relacionais MySQL e PostGreSQL.Logo em seguida foram feitos testes com o Java e os bancos de dados orientados a objetos DB4Objects e Intersystems Caché, conforme Capítulo 5.

Por fim, na quinta etapa foi relatada a conclusão do trabalho informando os resultados obtidos e os possíveis trabalhos futuros a serem realizados a partir desta pesquisa, relatado no Capítulo 6.

2 BANCO DE DADOS

Muitas empresas confiam em seus Sistemas de Banco de Dados (SBD) para manter informações atualizadas e consistentes. Sem eles, a maioria das empresas são incapazes de realizar suas transações diárias, muito menos de ajudar um gerente a tomar decisões estratégicas (HARRINGTON, 2002).

2.1 INTRODUÇÃO A SISTEMAS DE BANCOS DE DADOS

Segundo Date (1990) um banco de dados é um sistema de manutenção de informações, cujo objetivo principal é mantê-los e torná-los disponíveis quando solicitados. O grande objetivo de um banco de dados é oferecer uma visão “abstrata” dos dados ao usuário. A forma como essas informações são armazenadas não interessa ao usuário, mas sim como são disponibilizadas ao mesmo. Trata-se a informação considerada como dados significativos ao indivíduo ou à organização que gerou essas informações.

A informação acrescenta algo ao conhecimento, já o dado é uma representação, um registro de uma informação. O tratamento de informações dá origem a vários tipos de dados, porém o dado deve registrar apenas aspectos relevantes da informação. Desta forma, um banco de dados deve armazenar apenas informações importantes aos objetivos do sistema.

O banco de dados refere-se apenas aos dados, o sistema controlador de dados no banco de dados é denominado Sistema Gerenciador de Banco de Dados (SGBD). O principal objetivo de um SGBD é proporcionar um ambiente tanto conveniente quando eficiente para o armazenamento e disponibilização de informações do banco de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Os SGBD utilizam diferentes formas de representação ou modelagem de dados para descrever a estrutura das informações contidas em seus bancos de dados. Atualmente, os modelos utilizados são: modelo hierárquico, modelo em redes, modelo relacional (amplamente o mais utilizado) e o modelo orientado à objetos.

2.2 HISTÓRICO

Um dos primeiros sistemas de dados e antecessor do banco de dados e do modelo relacional foram os arquivos de dados. Segundo Harrington (2002), no inicio surgiram os arquivos de dados, mas com a necessidade de se gerenciar o armazenamento de informações nestes arquivos, surgiram diversas formas de gerenciamento de dados, sendo que a maioria dos quais antecederam o modelo relacional, no qual devido aos seus problemas, preparou o caminho para a aceitação dos bancos de dados relacionais.

Harrington (2002) afirma que os arquivos são armazenados e organizados em sistemas fixos e consistentes. Cada parte individual das informações (um nome, um sobrenome, um endereço e assim por diante) é conhecido como um campo.

Estes sistemas eram usados sob o conceito de sistema por sistema, ou seja, cada sistema teria uma aplicação separada de todos os outros setores. Segundo Caruso e Moraes (1999) devido às limitações das tecnologias de *software* e *hardware*, os sistemas antigos eram usados para processar poucas informações, organizados em ordem seqüencial. Os usuários tinham que controlar seus próprios registros e geralmente, a saída de um registro era a entrada de outro.

Silberschatz, Korth e Sudarshan (2006) explicam que entre os anos de 1950 até o inicio de 1960, as fitas magnéticas foram desenvolvidas para armazenamento seqüencial de registros, sendo que os dados eram muitas vezes maiores que a memória

principal do computador, obrigando os sistemas a processar lentamente os dados seqüencialmente. Entre os anos de 1960 à 1970, o uso dos discos rígidos para armazenamento de dados mudou drasticamente o processamento de informações. Com os registros livres da seqüencialidade, iniciou-se o surgimento dos primeiros sistemas de banco de dados, os hierárquicos e os de rede. Em 1970, E.F.Codd definiu o modelo de dados relacional e seus principais métodos de consulta responsáveis por obter as informações do banco de dados.

Já na década de 80, os bancos de dados relacionais tornaram-se tão competitivos quantos os sistemas hierárquicos e os de rede. Os primeiros sistemas de banco de dados comerciais foram: *International Business Machines* (IBM) DB2, Oracle, Ingres e *Digital Equipament Corporation* (DEC) Rdb. Todos foram fundamentais para o desenvolvimento de técnicas eficientes em consultas procedurais. Também foi nesta década que se realizaram muitas pesquisas sobre bancos de dados paralelos e distribuídos e o primeiro trabalho sobre banco de dados orientado a objetos (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

A linguagem *Structured Query Language* (SQL) surgiu no inicio da década de 90, projetada para aplicações comerciais. Os bancos de dados começaram a acrescentar suporte relacional de objeto em suas aplicações (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Com o surgimento da *World Wide Web* (WWW) no final da década de 90, os bancos de dados tiveram que se adaptar as interfaces Web e gerenciar gigantescas movimentações de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

O *Extended Markup Language* (XML) e a linguagem de consulta associada, XQuery surgiram no inicio do milênio, notando-se o aperfeiçoamento de técnicas de "computação autonômica/auto-administração" para reduzir esforços de administração

de sistema por meio de aplicações que gerenciam a si próprios, sem a intervenção humana (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

2.3 CONCEITOS SOBRE BANCO DE DADOS

A seguir encontram-se os principais conceitos utilizados em banco de dados.

2.3.1 Dados

O dado é parte fundamental na formação de informações. A palavra *dado* vem do latim *"data"*, plural de *"datum"*, que significa a "dar". Dados então, são "fatos dados", no quais podemos deduzir fatos adicionais. Os "fatos dados" correspondem ao que os matemáticos chamam proposição verdadeira, por exemplo, a afirmação "O fornecedor F1 esta localizado em Londres" pode ser uma proposição verdadeira (DATE, 2003).

Setzer (1999) define *dado* como uma seqüência de símbolos quantificados ou quantificáveis, podendo texto, imagem e som ser um dado já que as letras são símbolos quantificados, assim como imagens e sons.

Já a informação, segundo Date (2003) pode ser qualquer coisa com algum significado ao indivíduo, ou comumente ser tratada como sinônimo de dado e/ou também para se referir ao significado desse dado ao usuário.

O Tabela 1 apresenta um exemplo de uma coleção de dados encontrada em uma tabela de nome VINHO.

COD#	NOME	PRODUTOR
1	Cab. Sauvignon	Windsor
2	Pinot Noir	Fetzer
3	Merlot	Clos du Bois

Tabela 1. Exemplo de dados sobre vinhos
Fonte: Date, C. (2003)

2.3.2 Banco de dados

O termo banco de dados foi originado inicialmente pela comunidade cientifica da computação para especificar informações armazenadas em meio digital, porem atualmente, este termo pode ser usado tanto para indicar banco de dados digitais como banco de dados armazenados de outra forma.

Ao referir-se a banco de dados, Silberschatz, Korth e Sudarshan (1999) dizem que os primeiros bancos de dados desenvolveram-se a partir de sistemas de gerenciamento de arquivos. Que evoluíram primeiro para bancos de dados hierárquicos, depois para banco de dados de rede e mais tarde para banco de dados relacionais.

Segundo Date (2003) um banco de dados é um conjunto de proposições verdadeiras, ou seja, ele é um repositório para uma coleção de arquivos de dados.

A Figura 1 demonstra a estrutura de um banco de dados armazenado.

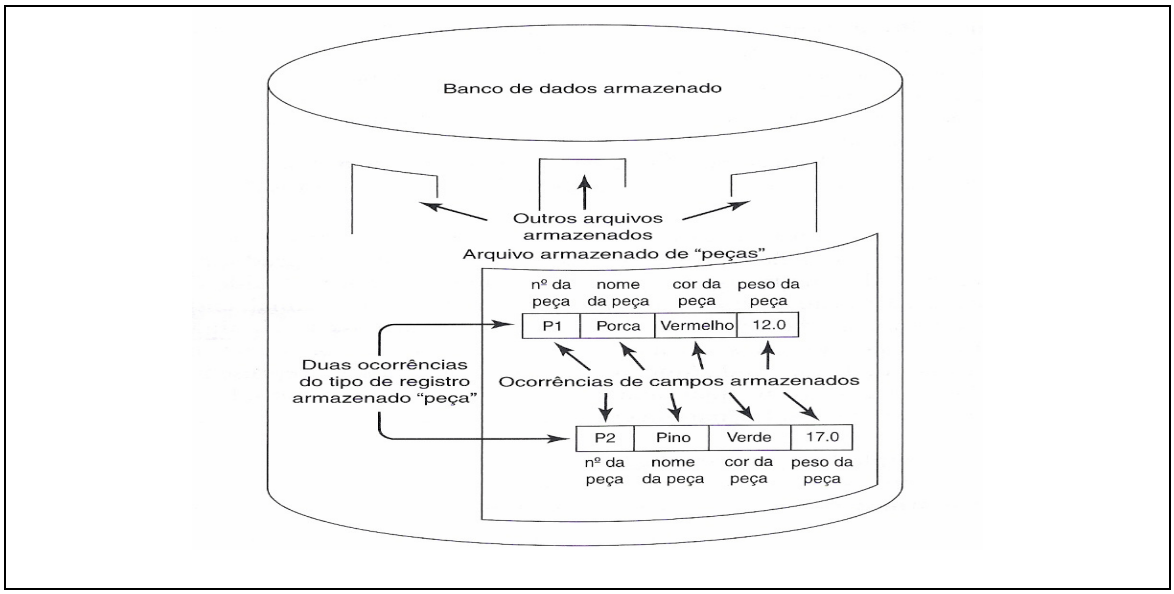


Figura 1. Estrutura de um banco de dados armazenado
Fonte: Date, C. (2003)

Na Figura 1 pode-se observar cada tipo de dado dos atributos armazenados em suas devidas tuplas. A chave primária é o "nrp da peça" armazenado na base de "peça". Essa mesma estrutura pode-se ligar a outras bases de dados por meio de chaves estrangeiras.

2.3.3 Sistemas de banco de dados

Um SBD é uma coleção de registros relacionados e um conjunto de programas que permitem as principais operações sobre esses dados. O sistema de banco de dados se preocupa em fornecer ao usuário uma visão abstrata da informação e ocultar a forma de armazenamento de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

A independência de dados é fundamental para os processos envolventes em um sistema de banco de dados. Ela especifica como a imunidade das aplicações a alterações na representação física e na técnica de acesso. As aplicações não devem depender de qualquer representação física ou técnica de acesso específico aos dados

(DATE, 2003).

Um SBD diferente de um Banco de Dados (BD) é algo mais complexo e envolve quatro componentes principais, conforme Date (2003):

a) dados;

b) hardware;

c) software;

d) usuários.

Os usuários de um sistema de banco de dados podem solicitar que ele realize diversas operações sobre os dados tais como:

a) inserir novos registros;

b) inserir registros em registros já existentes;

c) buscar registros em registros já existentes;

d) alterar registros em registros já existentes;

e) excluir registros.

Segundo Silberschatz, Korth e Sudarshan (2006) o banco de dados tem como objetivo proporcionar ao usuário uma visão transformada dos dados.

A arquitetura da *American National Standards Institute/Systems Planning and Requirements Committee* (ANSI/SPARC) de um SBD, conforme Date (2003) é dividida em:

a) nível externo: nível conceitual do banco de dados, podendo ser vista por um grupo de usuários;

b) nível conceitual: responsável pela persistência das informações dos banco de dados;

c) nível interno ou físico: responsável pela estrutura de armazenamento do banco de dados, caracterizando como ele esta armazenado.

A Figura 2 apresenta os três níveis da arquitetura de um SBD.

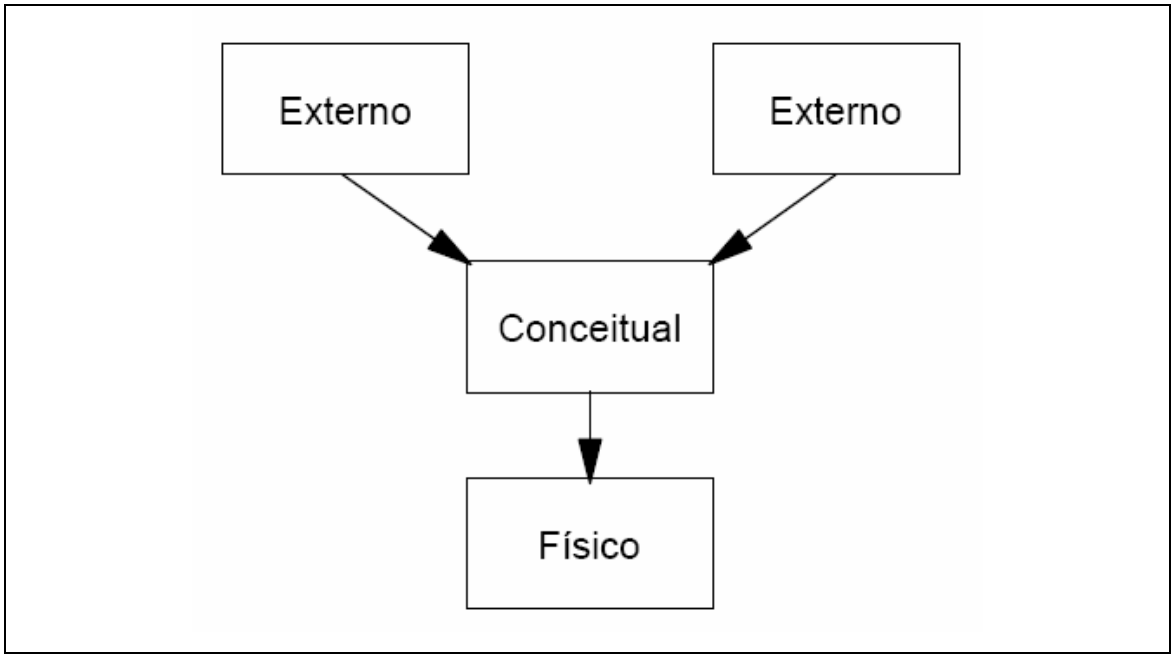


Figura 2. Níveis da arquitetura de um SBD
Fonte: Date, C. (2003)

2.3.4 Sistemas de Gerenciamento de Banco de Dados - SGBD

Segundo Date (2003) o SGBD é o software responsável por todo o acesso ao banco de dados. Todo o processo do banco de dados ocorre da seguinte forma:

a) um usuário executa um pedido de acesso usando uma sublinguagem, que geralmente é a SQL;
b) o SGBD interpreta o pedido;
c) o SGBD verifica por sua vez o esquema externo, ou as versões objeto deste esquema. Para esse usuário o mapeamento externo corresponde ao esquema conceitual, o mapeamento interno e a definição da estrutura de armazenamento;
d) por sua vez, o SGBD executa as operações necessárias sobre os banco de dados armazenados.

Conceitualmente, o SGBD deve buscas todas às tuplas de registros armazenados, depois, construir as ocorrências de registros conceituais exigidos e construir a ocorrência do registro externo. Em cada passo, pode ser necessário a conversão dos tipos de dados, ou qualquer outro tipo de conversão.

2.3.5 Diferença de Impedância em Banco de Dados

Segundo FINN (2002) o termo “diferença de impedância” é uma expressão utilizada em engenharia elétrica, mas na área computacional refere-se à diferença entre a comunicação dos modelos de banco de dados relacional e orientado a objetos.

No modelo relacional, se os dados forem complexos demais para serem representados em uma grade multidimensional, tabelas auxiliares serão criadas para conter as informações relacionadas. Sendo assim, cada tabela conterá alguns, mais não todos os dados para um grande volume de registros (FINN, 2002).

Já o modelo orientado a objetos, não esta limitado a manter as informações em tuplas e tributos. O desenvolvedor cria uma definição sobre a classe da informação. Este modelo organiza seus dados em objetos, sendo que cada objeto é uma instância daquela classe. Portanto, cada objeto contém todas as informações referentes aos itens da classe instanciada, sendo que as definições desta classe podem conter funções de programação, denominados métodos, que trabalham sobre os dados desta classe (FINN, 2002).

O problema de impedância ocorre quando há um grande esforço para se mapear dados relacionais em objetos e objetos em dados relacionais, para logo depois armazená-los, pois prejudicam tanto a produtividade do programador quanto o desempenho e agilidade da aplicação (FINN, 2002).

Existem técnicas e ferramentas de mapeamento objeto relacional como forma de se amenizar este tipo de problema. A ferramenta Hibernate da *Red Hat Inc* é uma delas.

2.4 MODELOS DE DADOS

Alem de seus componentes, um sistema de banco de dados é definido pelo seu modelo de dados, que é um conjunto de ferramentas conceituais, que descreve os dados, as relações que ele mantém, juntamente com sua semântica e restrições de consistência que fazem parte dos sistemas de banco de dados. O modelo de dados fica oculto em níveis de abstração para simplificar as interações do usuário com o sistema (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Os dois modelos de dados mais usados para armazenar e manipular informações, antes de existir o modelo relacional, eram o modelo hierárquico e o modelo de rede (HERNANDEZ, 2000).

> "Um **modelo de dados** é uma definição abstrata, autônoma e lógica dos objetos, operadores e outros elementos que, juntos, constituem a *máquina abstrata* com a qual os usuários interagem. Os objetos nos permitem modelar a *estrutura* dos dados. Os operadores nos permitem modelar a estrutura e seu *comportamento* " (DATE, 2003, p. 14).

2.4.1 Modelo Hierárquico

O modelo hierárquico foi o primeiro a ser reconhecido como modelo de dados no ano de 1966, como base de um produto comercial, o *Information Management System* (IMS) da IBM. Os utilizadores deste modelo de dados estão entre os primeiros a tratarem características como concorrência, recuperação, integridade e processamento eficiente na execução de consultas.

2.4.1.1 Características

Uma base de dados hierárquica consiste em uma coleção de registros, contados entre si através de ligações. Cada registro é uma coleção de campos (atributos), cada uma dos quais contendo somente uma informação. O modelo hierárquico limita seus relacionamentos a uma estrutura de árvore, que por meio de ponteiros fica "caminhando" para cima e para baixo (DATE, 2003).

A Figura 3 mostra uma coleção de registros ligados entre si numa estrutura hierárquica.

Os registros são organizados como árvores com raiz:

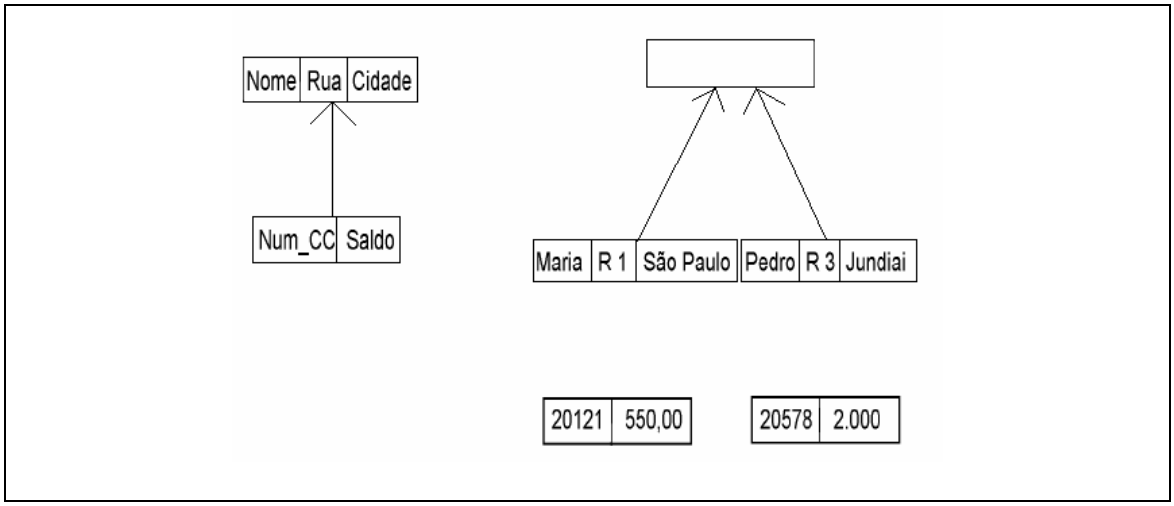


Figura 3. Registros conectados por meio de ponteiros
Fonte: SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, F., 2006

Cada árvore tem uma raiz, que é um pseudonó (cada nó é um registro, mas a raiz tem apenas a função de ser uma origem comum). Cada árvore com raiz é referida como uma árvore de base de dados; a base de dados hierárquica é uma coleção de árvores da base de dados (que formam uma floresta).

Em geral, as árvores se resumem a dois pontos:

a) não podem existir ciclos entre os nós (registros);

b) ligações formadas na árvore devem ser tais que somente retratem relações um-para-um ou um-para-muitos entre um pai e um filho.

Usa-se um diagrama de estrutura de árvore para apresentar o esquema para uma base de dados hierárquica. Consiste de dois componentes básicos: caixas (que correspondem ao tipo registro) e tuplas (que correspondem às ligações). Seu propósito é especificar a estrutura lógica geral da base de dados.

O esquema da base de dados é representado como uma coleção de diagramas de estrutura de árvore, para cada um destes diagramas, existe um único exemplo de uma árvore da base de dados. A raiz desta árvore é um pseudonó, e seus filhos deste são exemplos do tipo registro apropriado, que pode assim, por sua vez, ter vários exemplos de vários tipos registro.

Segundo Harrington (2002) se comparado com o sistema de armazenamento de arquivos, este modelo permite ao usuário armazenar e recuperar dados com base em relacionamento lógico de dados, portando fornece alguma independência de dados.

Este modelo ainda permite vantagens devido ao fato de que o dado pode ser acessado rapidamente, pois as estrutura das tabelas estão conectadas, sendo que a integridade referencial é construída e automaticamente executada no momento em que um novo nó é inserido na árvore (HERNANDEZ, 2000).

2.4.2 Modelo de Redes

O primeiro modelo de redes em 1967 como base no produto comercial da *Information Data System* (IDS) da empresa *General Eletric* (GE). Sua estrutura foi bem

aceita porque resolvia algumas limitações encontradas no modelo hierárquico. (HARRINGTON, 2002).

Em meados da década de 60 governo e profissionais da indústria dos Estados Unidos da América (EUA) organizaram-se no *Committee for Data Systems Languages* (CODASYL), cujo objetivo era desenvolver uma linguagem de programação corporativa, cujo resultado foi a linguagem *Common Business Oriented Language* (COBOL). À medida que trabalhava, o comitê percebeu que tinha outra saída além da linguagem de programação. O CODASYL gerou o *Data-base Task Group* (DBTG), que no ano de 1969 lançou seu conjunto de especificações, em que o ANSI fez algumas modificações no padrão para separar ainda mais o projeto lógico do banco de dados do layout do seu armazenamento físico (HARRINGTON, 2002).

O SGBD CODASYL mais bem sucedido foi o IDMS, originalmente desenvolvido pela empresa Cullinet. O IDMS era um produto de mainframe muito popular nos anos 1980, quando os SGBD relacionais começaram a dominar o mercado. O IDMS recebeu uma linguagem de consulta do tipo relacional e comercializou como IDMS/R, em seguida a Cullinet foi vendida para a empresa Computer Associates que o vende e suporta como CA-IDMS (HARRINGTON, 2002).

2.4.2.1 Características

No modelo de redes é semelhante ao modelo hierárquico, com a diferença de que cada filho pode ser ligado a mais de um pai, apresentando características mais complexas e muito utilizado em computadores de grande porte, portando este modelo acesso a qualquer nó da rede sem passar pela raiz. Este modelo também gerenciava problemas de concorrência e segurança. O mecanismo de segurança fornecia uma

facilidade na qual era possível bloquear parte do banco de dados para prevenir acessos simultâneos quanto necessários. Uma senha poderia ser adicionada a cada registro inserido no esquema. Já na recuperação de informações, era necessário que os dados a serem recuperados tivessem a mesma estrutura no esquema da base de dados (TAKAI; ITALIANO; FERREIRA, 2005).

A Figura 4 ilustra um exemplo de diagrama do modelo em rede.

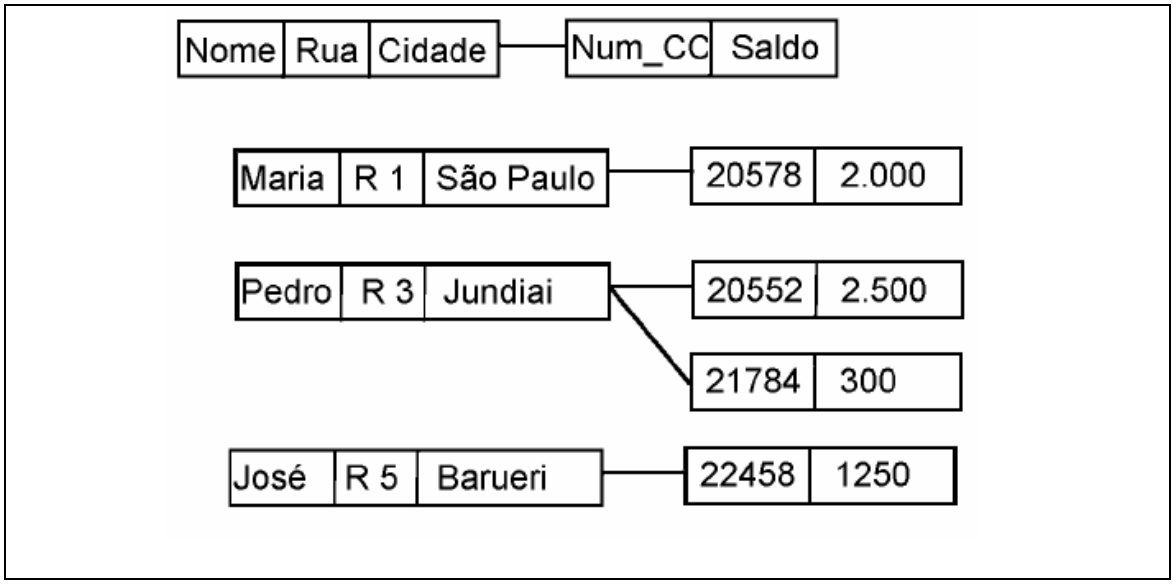


Figura 4. Exemplo de diagrama do modelo em rede
Fonte: TAKAI, K.; ITALIANO, I.; FERREIRA, J., 2005.

Tanto este modelo, quanto o modelo hierárquico são orientados a registro, isto é, cada operação na base de dados é feita de um em um registro por vez (TAKAI; ITALIANO; FERREIRA; 2005).

2.4.3 Modelo Relacional

Os primeiros sistemas relacionais tiveram seu inicio no final da década de 70 e no início da década de 80 (DATE, 2000).

> Um documento revolucionário de Codd [1970] definiu o modelo relacional e os métodos procedurais de consultar dados no modelo relacional, dando origem aos banco de dados relacionais. A simplicidade do modelo relacional e a possibilidade de ocultar completamente os detalhes de implementação do

> programador eram propostas realmente tentadoras. Mais tarde, Codd ganhou o prestigiado prêmio da Association of Computing Machinery Turing pelo seu trabalho (SILBERCHATZ; KORTH; SUDARSHAN, 2006, p. 19).

Desde o projeto de Codd, o modelo relacional não era muito utilizado, devido as desvantagens relacionadas ao desempenho dos modelos de redes e hierárquicos existentes. Isso mudou com o surgimento do *System R*, um projeto inovador da IBM Research que projetou técnicas para o desenvolvimento de banco de dados relacional eficiente (SILBERCHATZ; KORTH; SUDARSHAN, 2006).

Ainda segundo Silberchatz, Korth e Sudarshan (2006) no inicio da década de 80 os sistema de bancos de dados relacionais já haviam se tornado eficientes e competitivos com os bancos de dados hierárquicos e de redes.

Com o surgimento da linguagem SQL, na década de 90, vários fornecedores de banco de dados relacionais adaptaram suas ferramentas a esta linguagem de suporte a decisão (SILBERCHATZ; KORTH; SUDARSHAN, 2006).

Sendo assim, este modelo de dados se tornou um dos principais modelos para aplicações comerciais de processamento de dados. Seu principal destaque é devido a sua simplicidade, que facilita muito a vida do programador quando comparado com os modelos anteriores, como o de redes e o modelo hierárquico (SILBERCHATZ; KORTH; SUDARSHAN, 2006).

Já segundo Date (2000) o modelo relacional surgiu para dominar o mercado de banco de dados. O surgimento do modelo relacional em 1969-1970 foi sem duvida o evento mais importante em toda a história de banco de dados.

2.4.3.1 Características

Segundo Silberchatz, Korth e Sudarshan (2006) um sistema de banco de dados relacional consistem em uma coleção de tabelas para representar os dados e as

relações entre eles. Cada tabela contém vários atributos, sendo que cada atributo possui um nome único.

A Figura 5 ilustra um exemplo de banco de dados relacional.

Cod_Cliente	Nome	Rua	Cidade
1	Pedro	A	São Paulo
2	Maria	B	Jundiai

Num_CC	Saldo
20121	1200
21582	1320
21352	652

Cod_Cliente	Num_CC
1	20121
2	21582
3	21352

Figura 5. Exemplo de banco de dados relacional.
Fonte: SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, F., 2006

Este modelo de banco de dados usa uma estrutura baseada em registros. Ela recebe este nome porque o banco de dados é estruturado em formato fixo de vários tipos. Cada tipo de registro define o número fixo de campos. Os atributos por sua vez correspondem as características dos tipos de registros (SILBERCHATZ; KORTH; SUDARSHAN, 2006).

Segundo Date (2003) o modelo relacional usa três aspectos principais dos dados: a estrutura de dados, a manipulação de dados e a integridade de dados, tendo como base uma formação formal, ou teoria chamada modelo relacional de dados, tratando apenas das questões lógicas.

Nesta estrutura são observadas as seguintes características:

a) em um sistema deste tipo, as informações no banco de dados são vistas como tabelas (aspecto estrutural);

b) essas tabelas seguem regras de integridade, como chaves primárias e chaves estrangeiras (aspecto de integridade);

c) os manipuladores são operadores que derivam tabelas de outras tabelas, como junção, projeção, restrição etc. (aspecto manipulativo).

TAB: DEPTO

DEPTO	NOMEDEPTO	ORÇAMENTO
D1	Marketing	10M
D2	Desenvolvimento	12M
D3	Pesquisa	5M

DEPTO

DEPTO	NOMEDEPTO	ORÇAMENTO
D1	Marketing	10M
D2	Desenvolvimento	12M
D3	Pesquisa	5M

Tabela 2. Banco de dados em tabelas de departamentos e empregados
Fonte: Adaptado de Date (2000).

O exemplo de banco de dados na Tabela 2 apresenta tabelas relacionadas por meio de um campo chave em comum entre as tabelas DEPTO e EMP. O campo DEPTO# é utilizado de modo que os registros possam ser unidos com base no valor desta coluna.

Basicamente, sobre as tabelas ilustradas na Tabela 2 podem ser aplicadas regras estruturais, de integridade e manipulação.

Segundo Date (2000) toda estrutura de relação adota um cabeçalho e um corpo. O cabeçalho de uma dada relação pode ser considerado um predicado.

Geralmente, este tipo de banco de dados é usado por sistemas relacionais devido ao fato de que os sistemas relacionais têm maior controle sobre banco de dados relacionais, adotando características compatíveis com o banco de dados. Isso não significa que um outro tipo de sistema não possa usar um banco de dados relacional (Date, 2000).

Esta estrutura usa dados organizados em relações, controlado por restrições para evitar aspectos indesejáveis como repetição de dados, perda de dados ou a própria inconsistência de dados. Estas restrições são:

a) integridade referencial;

b) chaves primárias e chaves estrangeiras;

c) integridade de junções de relações.

De acordo com a arquitetura ANSI / SPARC em três níveis, os bancos de dados relacionais consistem de três componentes:

a) uma coleção de estruturas de dados, formalmente chamadas de relações, ou informalmente tabelas, compondo o nível conceitual;

b) uma coleção dos operadores, a álgebra e o cálculo relacionais, que constituem a base da linguagem SQL;

c) uma coleção de restrições da integridade, definindo o conjunto consistente de estados de base de dados e de alterações de estados. As restrições de integridade podem ser de quatro tipos: domínio (também conhecidas como type), atributo, relvar e restrições de base de dados.

Um dos pontos fortes do modelo relacional de banco de dados é a possibilidade de definição de um conjunto de restrições de integridade. Estas definem os conjuntos de estados e mudanças de estados consistentes do banco de dados, determinando os valores que podem e os que não podem ser armazenados.

2.4.4 Modelo Orientado a Objetos

As técnicas de Orientação a Objetos (OO) simulam realidades do mundo real. Os objetos podem ser criados a partir de outro objeto, da mesma forma que um

maquinário complexo, que possuem conjuntos de vários componentes (MARTIN, 1994).

Um Banco de Dados Orientado a Objetos (BDOO) é um SBD que armazena e recupera objetos. É possível armazenar o objeto na memória do computador e salvá-lo sem modificar sua estrutura. Isto facilita muito quando é utilizada uma linguagem Programação Orientada a Objetos (POO) com um BDOO, pois não é necessário mapear os objetos (RAO, 1995).

Esta tecnologia surgiu na década de 60. Originou-se da necessidade de se simular fenômenos como redes nervosas, fluxo de veículos, sistemas de redes, sistemas administrativos e até sistemas sociais. A primeira linguagem foi a SIMULA, criada por Kristem Nygaard (MARTIN, 1994).

Na década de 70, as técnicas de programação orientada a objetos foram melhor exploradas. Surgiram várias linguagens de programação, como a Java (PENDER, 2004).

Os primeiros BDOO surgiram na década de 80.

2.4.4.1 Características

O modelo orientado à objetos armazena as informações utilizando os mesmos conceitos da linguagem orientada à objetos. A única diferença é a persistência dos objetos, que garante a existência dos dados após o encerramento de um aplicativo. Para um objeto ser persistente é necessário que ele seja salvo permanentemente em um banco de dados.

Segundo Model (2003) ainda não existe um modelo de dados orientado a objetos padrão, mais é possível encontrar características comuns na maioria dos modelos existentes.

As principais características deste modelo são:

a) objeto, que possui um identificador único (pode ser referenciado), e literal, que não possui identificador (não pode ser referenciado);
b) o estado de um objeto é definido pelos valores de suas propriedades (atributos e relacionamentos);
c) o comportamento de um objeto é determinado pelo conjunto de operações (apenas as assinaturas das operações);
d) tanto objetos quanto literais podem ser de um tipo. Todos os elementos de um determinado tipo possuem um mesmo conjunto de propriedades e comportamento;
e) um modelo específico é construído usando uma *Object Definition Language* (ODL), independente de linguagem ou não.

2.4.4.1.1 Persistência de Objetos

A persistência é uma das principais características de um BDOO. Esta propriedade é responsável pelo armazenamento permanente em disco de um objeto instanciado na memória e pela facilidade e agilidade de recuperação da informação armazenada (KIRSTEN at al 2002).

Suas características costumam variar de acordo com o SBDOO. Segundo a Intersystems (2008), o BDOO Caché da empresa InterSystems adota as seguintes características de persistência:

a) automática: as informações são armazenadas em uma estrutura *Transactional Multidimensional Database Management* (TMDM) no seu modelo de dados.
b) definido pelo usuário: os dados são armazenados em uma estrutura hierárquica de dados, como globais.
c) externa: usando o Caché SQL Gateway, que faz uma ligação externa a outro banco de dados.
d) um modelo específico é construído usando a ODL (independente de linguagem ou não).

3 IMPEDÂNCIA DE DADOS EM BANCO DE DADOS

Uma das grandes dificuldades dos bancos de dados relacionais era o compartilhamento de dados com as linguagens orientadas a objetos como Java ou C++. Simplesmente estender o tipo de comunicação da linguagem orientada a objetos com o banco de dados relacional não é o suficiente (SILBERCHATZ; KORTH; SUDARSHAN, 2006).

> As diferenças entre o sistema de tipo de banco de dados e o sistema de tipo de linguagem de programação tornam o armazenamento e a recuperação de dados mais complicados e precisam ser minimizadas. A necessidade de expressar acesso a banco de dados usando uma linguagem (SQL) que é diferente da linguagem de programação novamente dificulta o trabalho do programador. Para muitas aplicações, é desejável ter construções ou extensões de linguagem de programação que permitam acesso direto a dados no banco de dados, sem ter de usar uma linguagem intermediária como SQL (SILBERCHATZ; KORTH; SUDARSHAN, 2006, p. 241).

Essas diferenças trazem grandes problemas para os analistas de sistemas. Muitos deles gostariam de utilizar tecnologia orientada a objetos como Java, C++ e COM, devido à produtividade e os modelos de dados mais ricos que elas oferecem. Já por outro lado, suas aplicações precisam acessar informações em banco de dados relacionais ou interagir com ferramentas que auxiliam na análise de dados e geração de relatórios, que geralmente utilizam SQL. Este problema é conhecido como diferença de impedância em banco de dados, pois os dados relacionais não se encaixam bem com as linguagens orientadas a objetos (INTERSYSTEMS, 2008).

Como os objetos não podem ser diretamente representados em um banco de dados relacional, existem ferramentas que fazem o papel de comunicação entre o objeto e as tabelas do banco de dados. Elas são conhecidas como ferramentas de mapeamento objeto-relacional ou ferramentas *Object-relational mapping* (ORM).

3.1 OBJECT RELATIONAL MAPPING (ORM)

Em um mundo praticamente dominado pelos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), é uma tendência natural se usar o modelo relacional para armazenar dados persistentes. O modelo orientado a objetos permite uma abrangência maior sobre as aplicações, assim como o modelo relacional permite o armazenamento de dados em tuplas e tabelas.

O modelo relacional se baseia em princípios matemáticos, principalmente na teoria de conjuntos. Já a orientação a objetos se baseia nos princípios de engenharia de software. A orientação a objetos foi criada para criar aplicações com dados e comportamento, enquanto o modelo relacional foi criado apenas para armazenamento de dados (DATE, 2003).

O principal problema ao se usar um banco de dados relacional em uma linguagem orientada a objetos é o problema de casamento de impedância.

Este problema basicamente se refere às dificuldades técnicas/conceituais e a grande perda de tempo para se armazenar dados persistentes em banco de dados relacionais, quando usado uma aplicação desenvolvida em linguagem orientada a objetos com um banco de dados relacional.

O uso de ferramentas ORM pode reduzir o problema de impedância de banco de dados, permitindo a aplicação uma independência de banco de dados, além de mapear todos os tipos de dados nos atributos das classes orientadas a objetos para as tuplas relacionais.

3.2 FERRAMENTAS ORM

As ferramentas ORM têm como objetivo, reduzir a impedância nos sistema de programação orientada a objetos quando utilizada com bancos de dados relacionais. Com sua utilização, o programador não precisa se preocupar com os comandos de linguagem SQL devido ao fato de que a aplicação ORM faz todo o trabalho de persistência de dados.

Uma solução ORM, segundo Bauer e King (2007) consiste basicamente em quatro partes:

a) uma *Application Programming Interface* (API) para realizar operações como *Creat, Retrieve, Update, Delete* (CRUD) básicas em objetos de classes persistentes. Essa API é usada para operações em banco de dados relacionais;
b) API ou linguagem para especificar consultas que se referem às classes ou os atributos das classes;
c) facilidade para especificar o método de mapeamento;
d) funções para que a aplicação ORM interaja com os objetos transacionais verificando sujeira do mapeamento (tecnicamente chamado de *dirty checking*), recuperação de associação ociosa (tecnicamente chamado de *lazy association fetching*) e outras funções de otimização.

Segundo Bauer e King (2007) uma ferramenta ORM trabalha diretamente na "camada de persistência" também podendo ser chamada de "camada de mapeamento", conforme Figura 6. Essa camada de persistência é a base em uma arquitetura em camadas. Em uma aplicação usando uma solução ORM, teríamos as seguintes camadas:

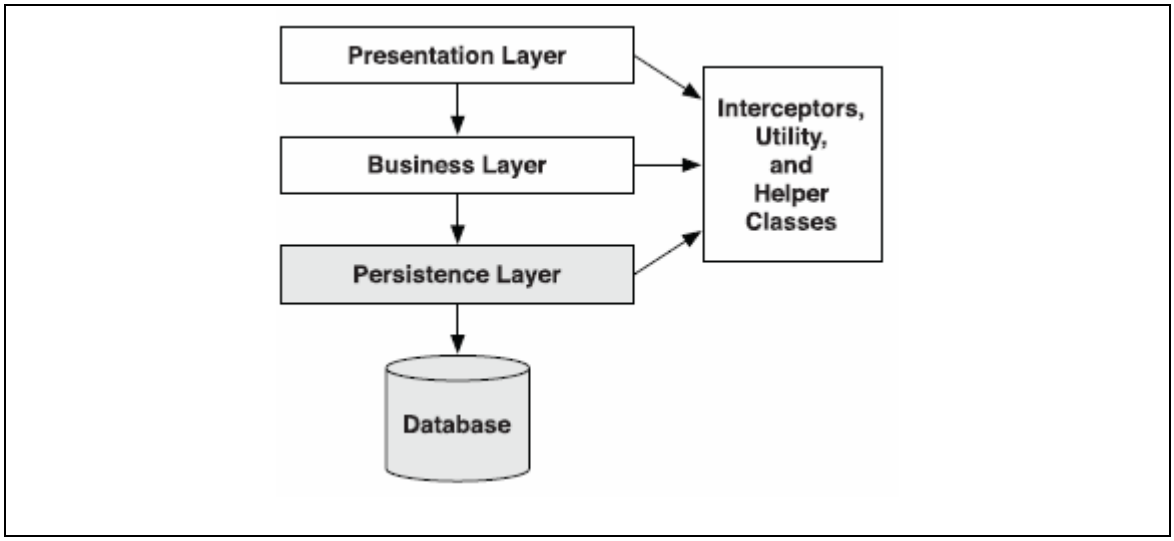


Figura 6. Camada de persistência
Fonte: Bauer, C.; King, G., (2007).

a) camada de apresentação (*presentation layer)*: Lógica da interface com o usuário responsável pela apresentação, controle da página e a navegação;
b) camada de negócio (*business layer*): Esta camada é formada conforme a aplicação. Geralmente, esta camada é responsável por qualquer regra de negócio ou necessidades do sistema entendido como parte do domínio do problema. Algum tipo de controle sempre é atribuído a esta camada, podendo ser uma biblioteca que permite invocar a regra de negócio adequada;
c) camada de persistência (*persistence layer)*: esta camada é um grupo de classes e componentes responsáveis por armazenar dados e recupera-los em um ou mais repositório de dados. Esta entidade inclui o modelo de domínio de negócios. Na Figura 7 pode-se ter uma visão mais completa desta camada;
d) banco de dados (*database)*: o banco de dados existe fora da aplicação em si. É na verdade a representação persistente das classes e objetos no sistema;

e) classes de ajuda e de utilidade (*interceptors, utility and helper class)*: toda a aplicação tem um conjunto de infra-estrutura de classes de ajuda e de utilidade que são usadas em toda a camada de aplicação. Estas funções não formam uma camada, pois não obedecem às regras de dependência entre as camadas.

A utilização dessa ferramenta deixa a "camada de mapeamento" transparente para a aplicação, minimizando o problema.

A Figura 7 ilustra a camada de mapeamento objeto relacional deixando seu uso transparente para a aplicação.

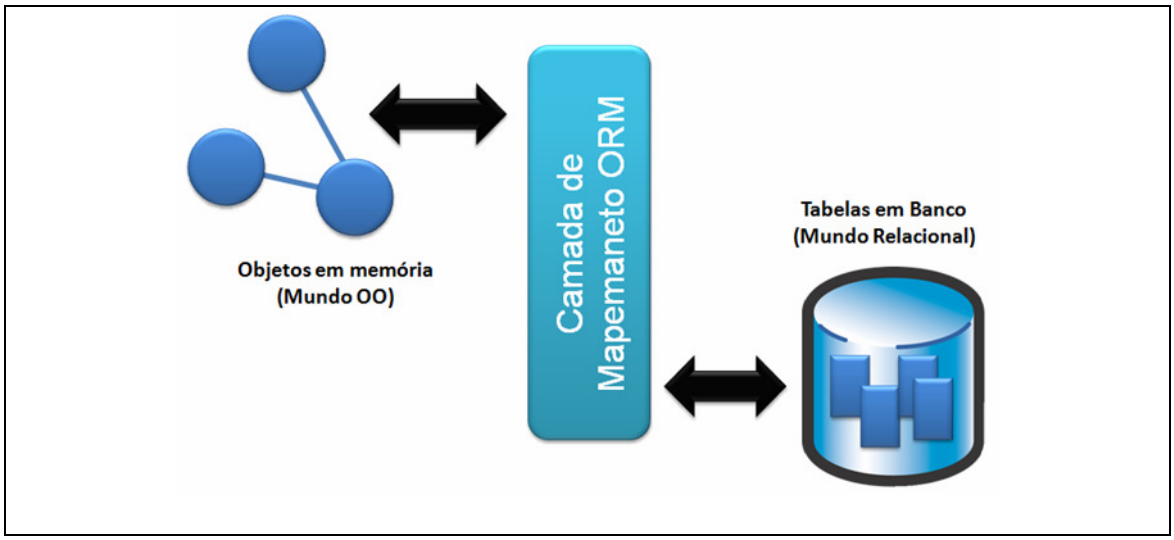


Figura 7. Camada de mapeamento objeto relacional
Fonte: Adaptado HIBERNATE (2008).

Conforme Bernadi (2007) em termos conceituais, uma camada de persistência de objetos é uma biblioteca que permite a realização do processo de persistência. Dentre suas várias vantagens, destaca-se o fato de o analista/programador poder trabalhar como se estive-se uma ferramenta completamente orientada a objetos. Outra vantagem é que os acessos realizados diretamente ao banco de dados da aplicação são isolados e os processos de consulta e operações de manipulação de dados são concentrados em uma camada de objetos inacessível ao desenvolvedor. Essa vantagem

torna as aplicações mais confiáveis, permitindo ao SGBD ou a estrutura das suas tabelas possam ser modificadas sem a necessidade da compilação dos programas.

Já segundo Hibernate (2008) esta camada é responsável por resolver vários problemas do mapeamento objeto relacional, tais como:

a) mapeamento entre entidade x classe (queries x tipos);
b) herança
c) tratamento de chaves (primaria e estrangeira);
d) representação de relacionamentos;
e) campos calculados;
f) diferenças entre tipo SQL e linguagem de programação;
g) *locking* e *isolation level*;
h) *caching*;
i) outros diversos.

Existem várias ferramentas ORM no mercado que implementam a camada ORM e variados outros recursos. Na sua maioria, todos são programas gratuitos e de código fonte aberto.

3.2.1 Aplicações comerciais ORM

O armazenamento objeto relacional acontece por meio de *frameworks* de mapeamento objeto relacional. Os *frameworks* mais comumente usados comercialmente são:

a) Hibernate para Java: considerado o mais usado *framawork* segundo o site do próprio fabricante, desenvolvido sob licença *Lesser General Public License* (LGPL) pela *Red Hat Inc.* para mapeamento objeto relacional

utilizando a LPOO Java. Na sua versão 3.2.6 GA, este *framework* contém suporte as plataformas relacionais MySQL, MS SQL, Interbase/Firebird, PostGreSQL e Oracle;

b) NHibernate para aplicações .NET: versão do Hibernate para aplicações .NET também desenvolvido sob licença LGPL pela *Red Hat Inc.* Na sua versão 1.2.0 GA, este *framework* contém suporte as plataformas relacionais MySQL, MS SQL Server, Interbase/Firebird, PostGreSQL e Oracle;

c) SQLObject para aplicações Python: versão de ferramenta para mapeamento objeto relacional desenvolvida para Python. Desenvolvida sob linceça LGPL, na sua versão 0.10.0 este *framework* possui suporte para as plataformas relacionais MySQL, Postgres, SQLite, Firebird, Sybase, MAX DB e MS SQL Server;

d) *Apache Object Relational Bridge* (OJB) para Java: *framework* desenvolvido para Java pela Apache Software Fundation sob licença *General Public License* (GNU). Na sua versão 2.0 esta ferramenta possui suporte para as plataformas relacionais MS SQL Server, MySQL, DB2, Oracle, MS Access, PostGre, Informix, SyBase, SAP DB e MAX DB;

e) *Java Data Objects* (JDO) para Java: ferramenta desenvolvida para java pela Apache Software Fundation sob licença GNU, na sua versão 0.10.0 este *framework* possui suporte para as plataformas relacionais MySQL, Postgres, SQLite, Firebird, Sybase, MAX DB e MS SQL Server;

O *framework* ORM foco de estudo do mapeamento objeto relacional será a versão do Hibernate Core 3X para Java.

3.2.2 Hibernate

Hibernate é um *framework* ORM de código fonte aberto, componente do projeto *JBoss Enterprise Middleware System* (JEMS), uma divisão da *Hed Hat Inc.*

Este *framework* é responsável pela camada de mapeamento objeto relacional necessário para armazenar objetos Java em uma base de dados relacional, conforme Figura 7 (HIBERNATE, 2008).

Por meio da camada de mapeamento é que a aplicação ORM permite o desenvolvimento de classes orientado a objetos persistentes, podendo-se usar associação, herança, polimorfismo, composição e coleções, como também possibilita executar consultas SQL de extensão *Hibernate Query Language* (HQL*)*, SQL nativa ou pesquisas orientadas a objetos. (HIBERNATE, 2008).

Trabalhando na camada de mapeamento objeto relacional, conforme Figura 7, o Hibernate permite ha aplicação, independência de banco de dados, pois sua arquitetura permite que os atributos e tuplas sejam mapeados a estrutura e não ao tipo de banco de dados.

3.2.2.1 Arquitetura Geral do Hibernate

Segundo Hibernate (2008) a arquitetura mais amplamente vista deste ORM, usa base de dados relacionais configuradas por meio de arquivos XML para realizar o mapeamento objeto relacional, conforme ilustrado na Figura 8.

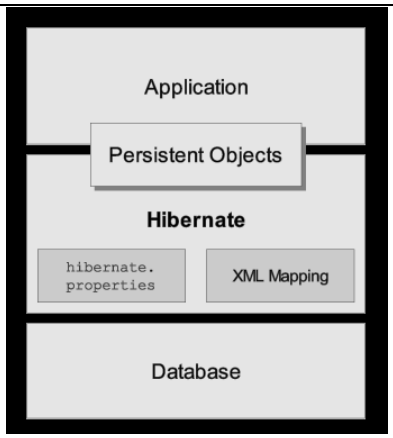


Figura 8. Arquitetura geral do Hibernate
Fonte: Hibernate (2008).

Por meio de arquivos XML, toda a relação do mapeamento objeto relacional é configurada. Na camada de mapeamento, cada objeto persistente é convertido a uma coluna da tabela do campo do banco de dados relacional. Geralmente o mapeamento começa dizendo a ferramenta Hibernate qual tabela e seus respectivos campos a serem mapeados.

Segundo Bernadi (2007) dentre as vantagens do uso desta ferramenta ORM, talvez a mais importante esteja relacionada à persistência de dados. As regras e rotinas para tratamento de persistência podem ser criadas diretamente no banco de dados.

A grande desvantagem do Hibernate, pode ser observada no momento da análise de um sistema a ser implementado, devido a grande dificuldade em se obter a realidade, ou seja, traduzir para meio de tabelas, seus campos, suas chaves primárias e estrangeiras a realidade do mundo real possibilitado pela POO.

Conforme Hibernate (2008) uma visão de sua arquitetura detalhada demonstra como o Hibernate é flexível e possibilita vários enfoques. Analisando os dois extremos da arquitetura do Hibernate, temos a arquitetura mais simples, via JDBC utilizando o mínimo de API's possível, conforme Figura 9.

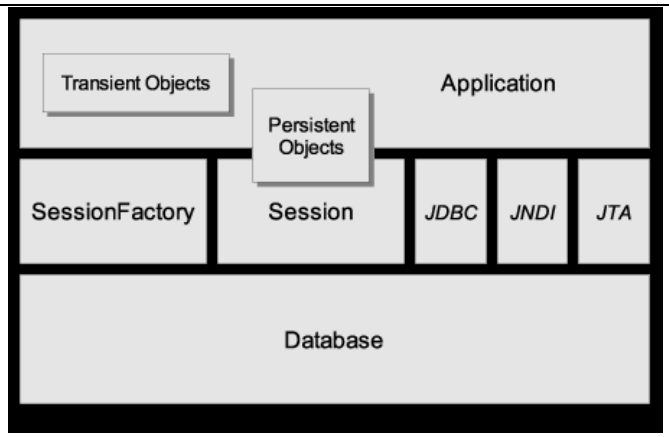


Figura 9. Arquitetura simples do Hibernate.
Fonte: Hibernate (2008).

Com uma arquitetura mais completa, o Hibernate permite controlar as informações diretamente na camada de mapeamento objeto relacional, conforme demonstra a Figura 10.

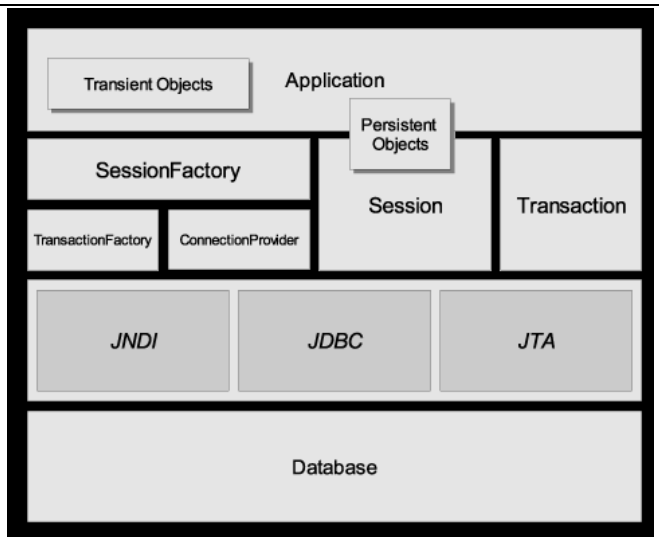


Figura 10. Arquitetura complexa do Hibernate.
Fonte: Hibernate (2008).

3.2.2.2 Regras para o mapeamento de associações entre entidades

As regras de mapeamento têm como objetivo garantir que todas as características do modelo de dados orientado a objetos sejam reproduzidas fielmente pelo banco de dados relacional. Conforme Bernadi (2007) os objetos formam unidades

que armazenam atributos e operações. Os bancos de dados relacionais representam de forma eficiente os atributos, mais são limitados na representação das operações.

Geralmente, para se mapear um objeto em uma tabela ou um campo relacional, os atributos destes objetos são representados por meio dos atributos na tabela. Essa situação não é única, pois há variação entre os tipos de dados, tamanho dos campos e outras propriedades, podendo-se até fazer com que um atributo seja mapeado em vários atributos na tabela, ou então que muitos atributos sejam mapeados em apenas um atributo na tabela.

Ao traduzir um paradigma de modelo de dados orientado a objetos para um relacional, algumas regras de mapeamento devem ser observadas, mas não são necessariamente obrigatórias.

3.2.2.2.1 Atributo *Object Identifier* (OID) para cada tabela

Como foi visto nos capítulos 2.4.3 e 2.4.4, todas as relações devem ter uma chave primária, assim como toda a classe deve ter um objeto que é único. A comunicação entre um e outro é garantida por meio da introdução de um "identificador de objetos" - OID. Essa propriedade é gerada pelo sistema e não pode ser alterada pelo usuário, sendo que seu valor não depende dos valores dos atributos deste objeto. No Hibernate, esta implementação sobre o modelo relacional, deve-se a criação de um atributo OID para cada tabela. A Figura 11 ilustra um exemplo de como seria um mapeamento objeto relacional de um objeto "Pedido" para uma relação "Pedido".

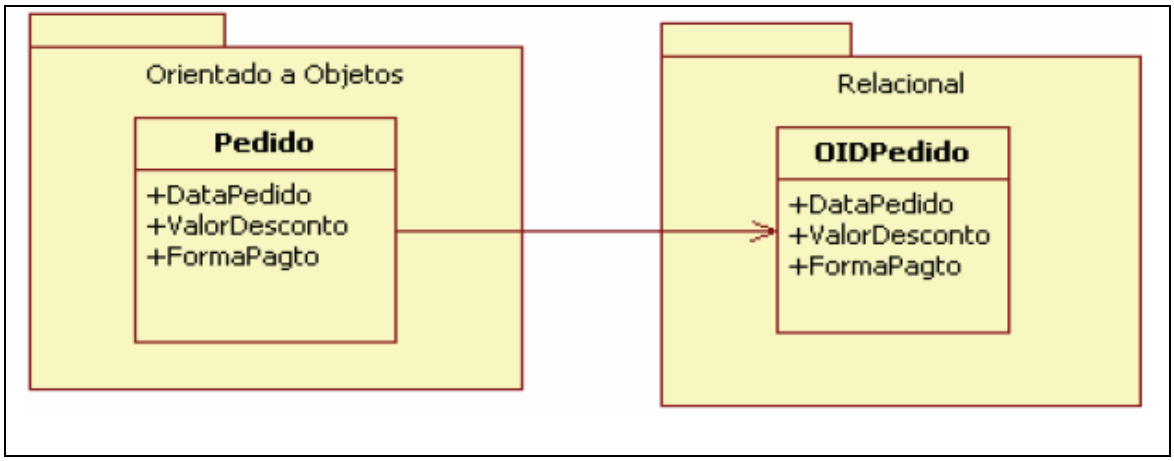


Figura 11. Exemplo de mapeamento objeto relacional por OID para cada tabela
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.2 Mapeamento simples, compostos e multivalorados

Neste tipo de associação existem três tipos de atributos a serem mapeados, os simples, os compostos e os multivalorados. Os simples são mapeados para colunas, os compostos devem ser mapeados em varias colunas e os multivalorados devem ser mapeados em tabelas onde a chave primária da tabela representa a classe que contém o atributo multivalorado e pela chave primária que representa o atributo multivalorado (BERNADI, 2007).

Na Figura 12 pode ser observado o mapeamento dos atributos simples e compostos. Já na Figura 13 o mapeamento dos atributos multivalorados.

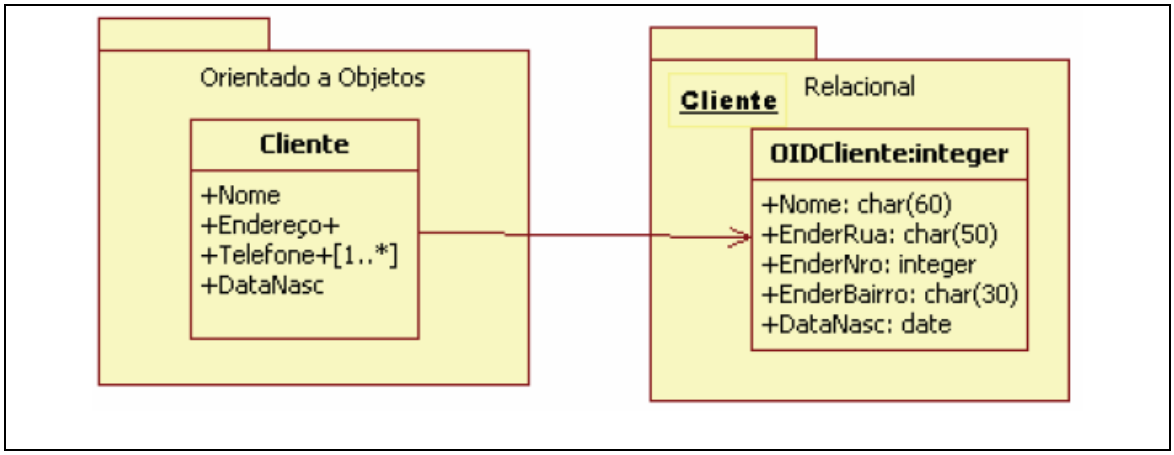


Figura 12. Exemplo de mapeamento de atributos simples e compostos
Fonte: Adaptado de BERNADI, A. (2007)

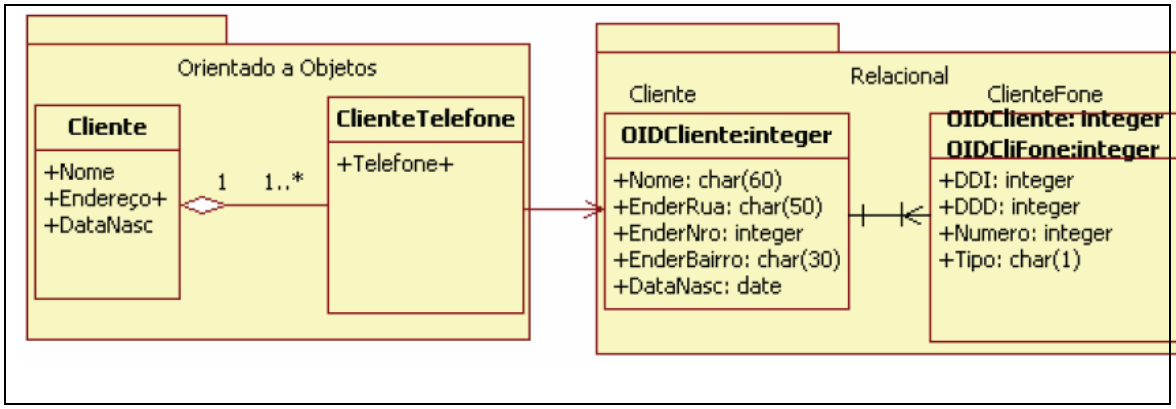


Figura 13. Exemplo de mapeamento de atributos multivalorados
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.3 Herança

Segundo Bernadi (2007) este conceito pode ser mapeado de três maneiras. A primeira é simplesmente criar uma tabela para a classe. Neste mapeamento, os campos da classe são os mesmos campos da tabela com uma coluna de chave estrangeira para armazenar o índice da tabela pai. A Figura 14 demonstra o uso do mapeamento de uma tabela para cada classe.

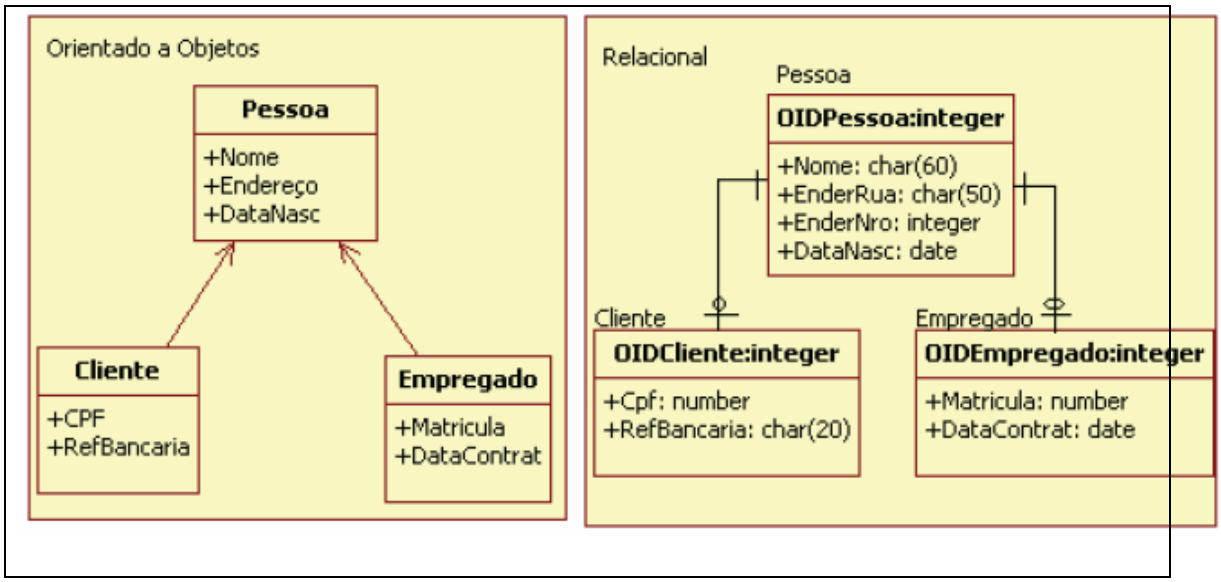


Figura 14. Mapeamento de uma tabela para cada classe
Fonte: Adaptado de BERNADI, A. (2007)

A segunda forma de se mapear a herança é criar uma tabela para toda a

hierarquia de classe. Neste mapeamento, todos os atributos da classe são armazenados em uma única tabela. A Figura 15 ilustra o mapeamento de toda a hierarquia de classes em uma única tabela.

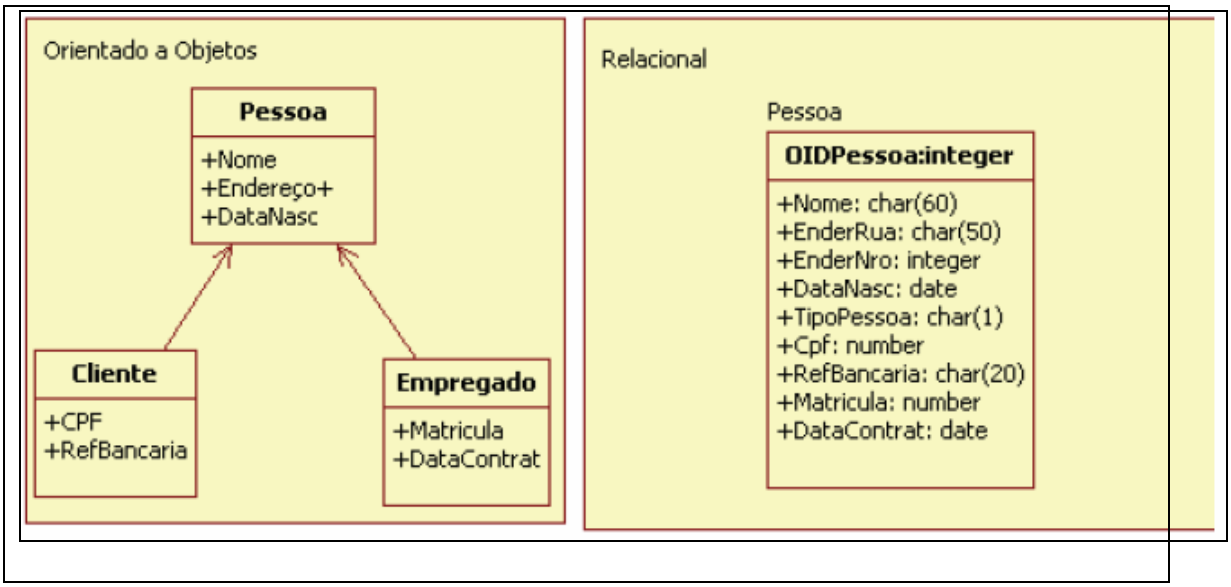


Figura 15. Mapeamento de toda a hierarquia de classe em uma única tabela
Fonte: Adaptado de BERNADI, A. (2007)

Por fim, a terceira maneira é o mapeamento de uma tabela para cada classe concreta. Neste caso, deve-se incluir em cada tabela os atributos específicos, quanto os herdados da classe que ela representa. A Figura 16 demonstra o mapeamento de uma tabela para a classe concreta

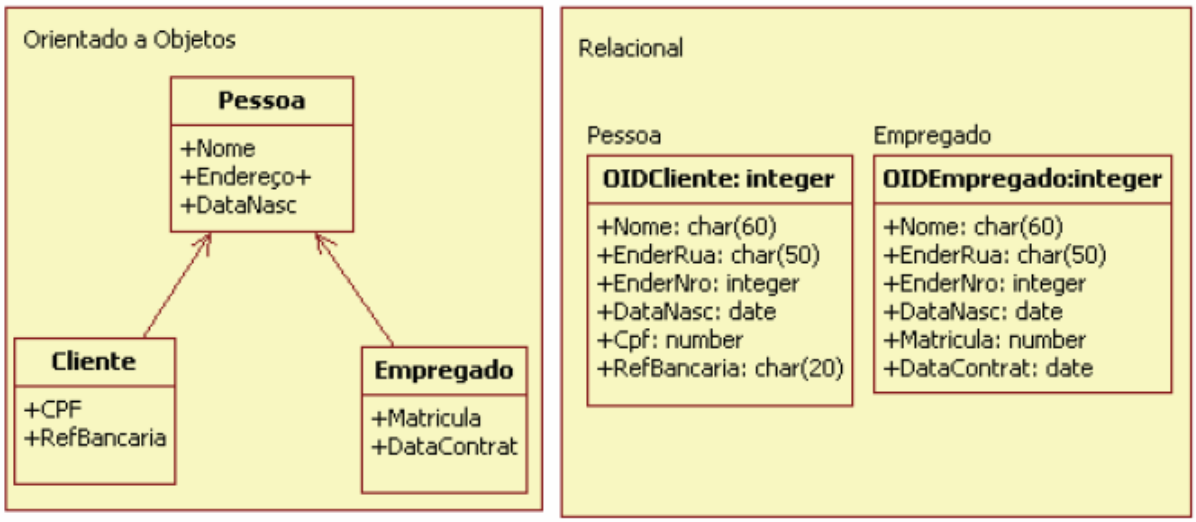


Figura 16. Mapeamento de uma tabela para cada classe concreta

Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.4 Associações muitos-para-muitos

Neste modelo, uma tabela é criada e adicionada ao mapeamento. Nesta tabela, a chave primária é composta pelas chaves primárias das tabelas associadas à entidade. Segundo Bernadi (2007), um exemplo básico é a existência de dois objetos, "Aluno" e "Disciplina" que ao passarem pela camada de mapeamento, se tornaram três tabelas, "Aluno", "Disciplina" e "Freqüenta". A Figura 17 demonstra o uso da associação muitos-para-muitos.

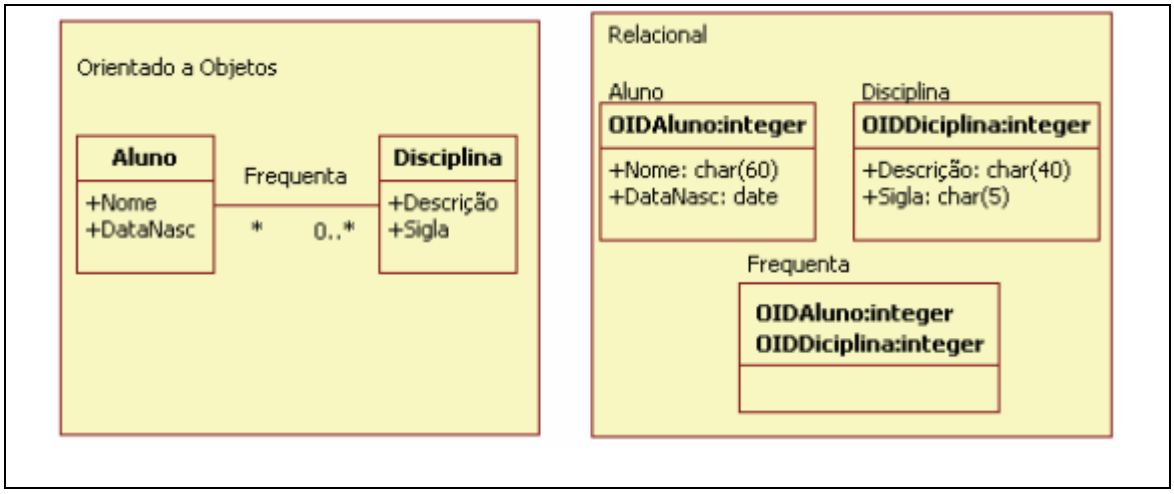


Figura 17. Mapeamento de uma associação muitos-para-muitos.
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.5 Associações muitos-para-muitos com associação

Neste caso, aplica-se a regra anterior, sendo que os atributos da classe associativa permanecerão na tabela que é usada para gerar o mapeamento da associação. A Figura 18 ilustra como pode ser feito o mapeamento de uma associação de muitos-para-muitos.

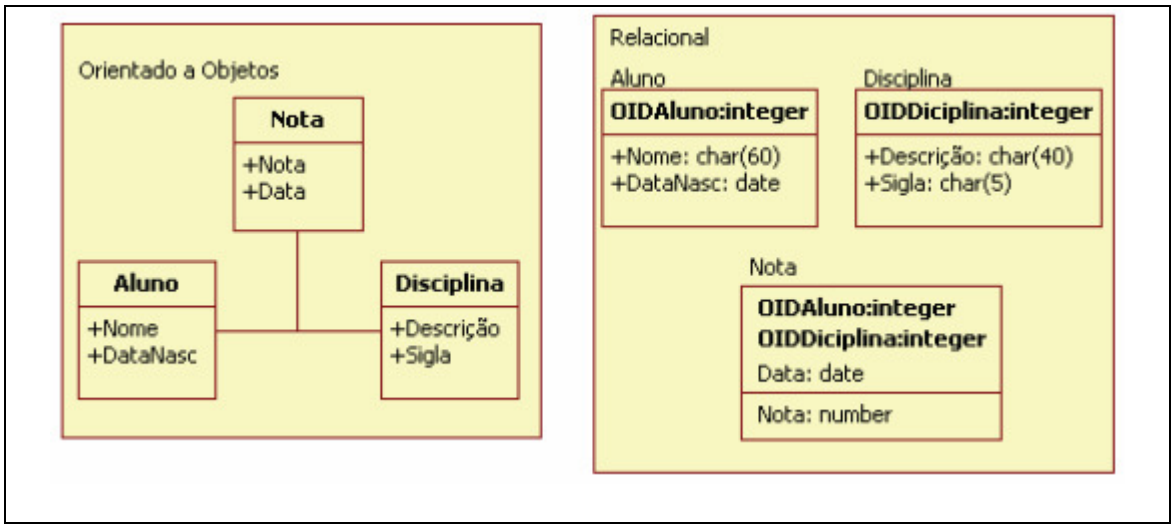


Figura 18. Mapeamento muitos-para-muitos com agregação
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.6 Associação um-para-muitos

Neste modelo, a tabela cujos registros podem ser endereçados diversas vezes é a que herda a referência da tabela cuja correspondência é unitária. A Figura 19 exemplifica uma tabela "Pedido" que faz referência a tabela "Cliente" por meio de uma chave estrangeira "FK".

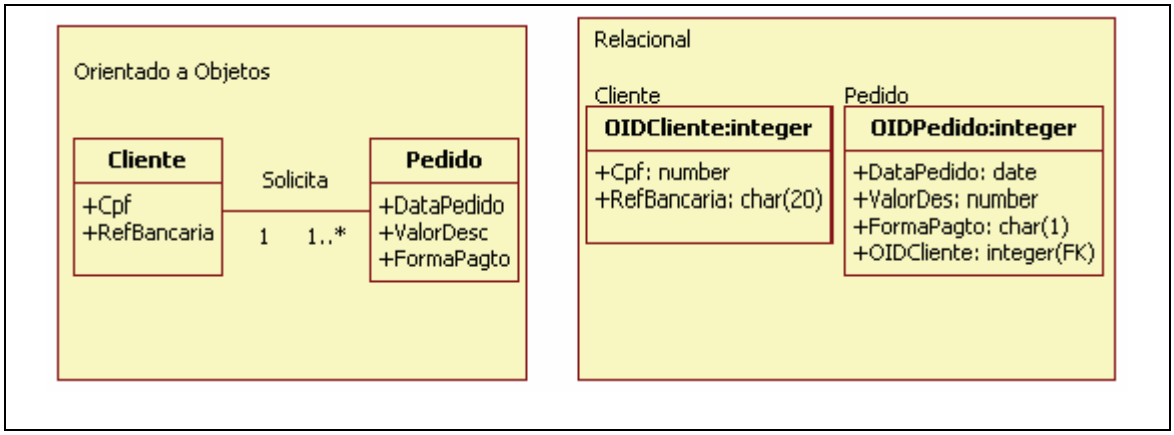


Figura 19. Associação um-para-muitos
Fonte: Adaptado de BERNADI, A. (2007)

Sétima regra, associação um-para-muitos com associações: neste caso, aplica-se a sexta regra sendo que os atributos da classe são herdados como atributos normais pela tabela que herda a chave estrangeira. A Figura 20 demonstra o uso da

associação um-para-muitos com associações.

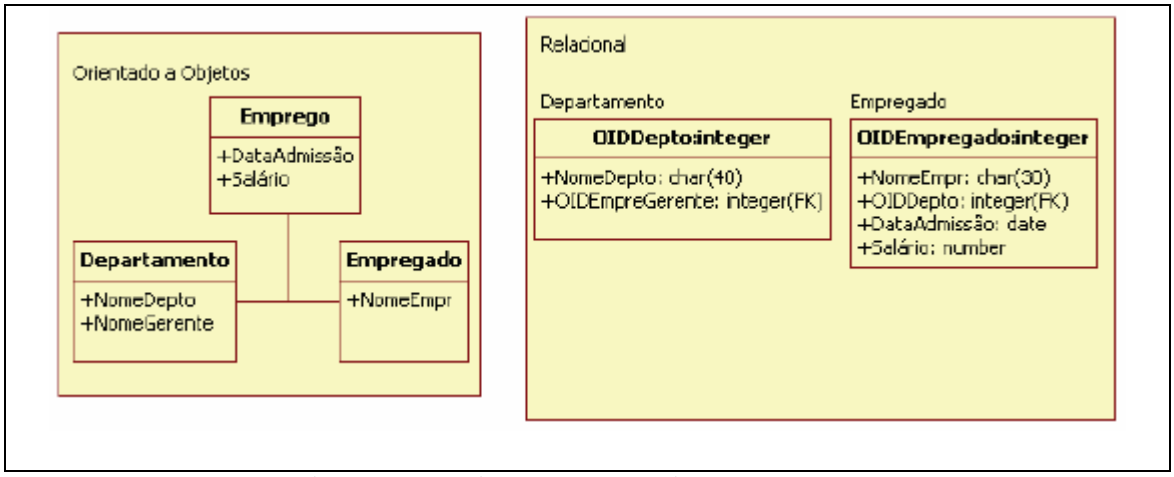


Figura 20. Associação um-para-muitos com agregação
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.2.7 Associação um-para-um

Neste modelo, o programador deve optar por gerar duas opções. A primeira é gerar uma única tabela relacional. Os atributos da classe mapeada devem ser inseridos na tabela mapeada. O objeto mapeado é automaticamente excluído quando o objeto da classe for eliminado, não havendo necessidade de “triggers” ou rotinas especiais na aplicação para garantir a consistência do banco de dados. Na Figura 21 pode ser observado um exemplo da utilização da técnica de geração de uma única tabela, sendo utilizado o prefixo “PagCh” para distinguir os atributos. Os objetos “Pedido” e “Pagto em cheque” são mapeados na tabela “Pedido”.

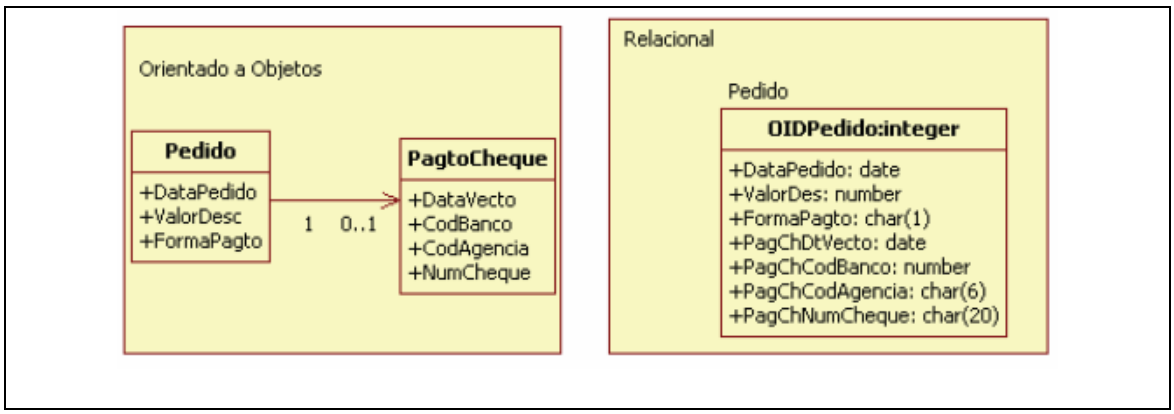


Figura 21. Associação um-para-um com única tabela relacional

Fonte: Adaptado de BERNADI, A. (2007)

Na segunda opção, o programador deve gerar uma tabela para cada classe. Uma delas deve herdar com um atributo comum, a chave estrangeira referenciando a tabela relacional associada à Figura 22 demonstra o uso desta técnica por meio da geração das tabelas de acordo com a quantidade de classes. Nesta ilustração, os objetos "Pedido" e "Pagto em cheque" são mapeados nas tabelas "Pagamento" e "PagamentoCheque".

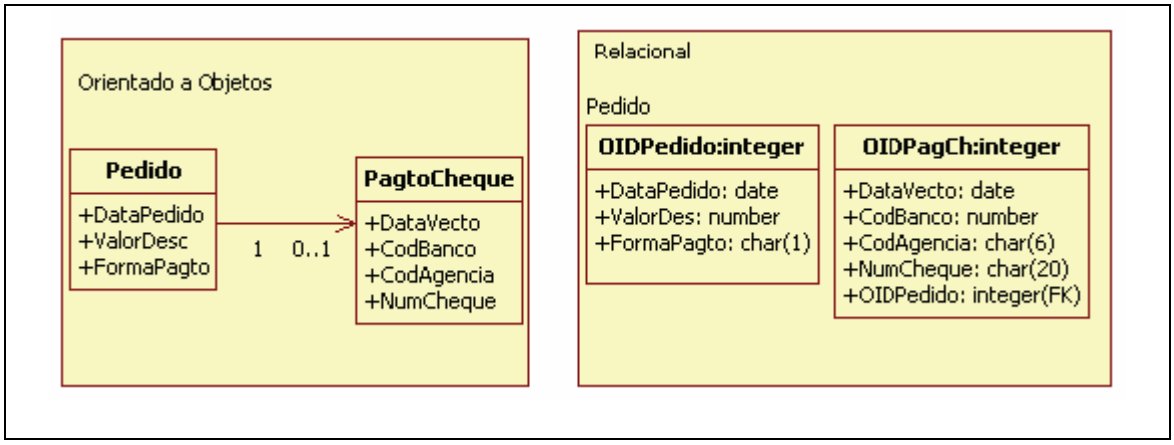


Figura 22. Associação um-para-um com uma tabela para cada classe
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.3 Exemplo de mapeamento

Para demonstrar o uso das regras demonstradas nas regras de mapeamento, a Figura 23 ilustra como seria o mapeamento de um sistema simplificado de controle de uma biblioteca. Vale lembrar que essas regras não são obrigatórias e podem ter variação de acordo com o tipo de sistema implementado.

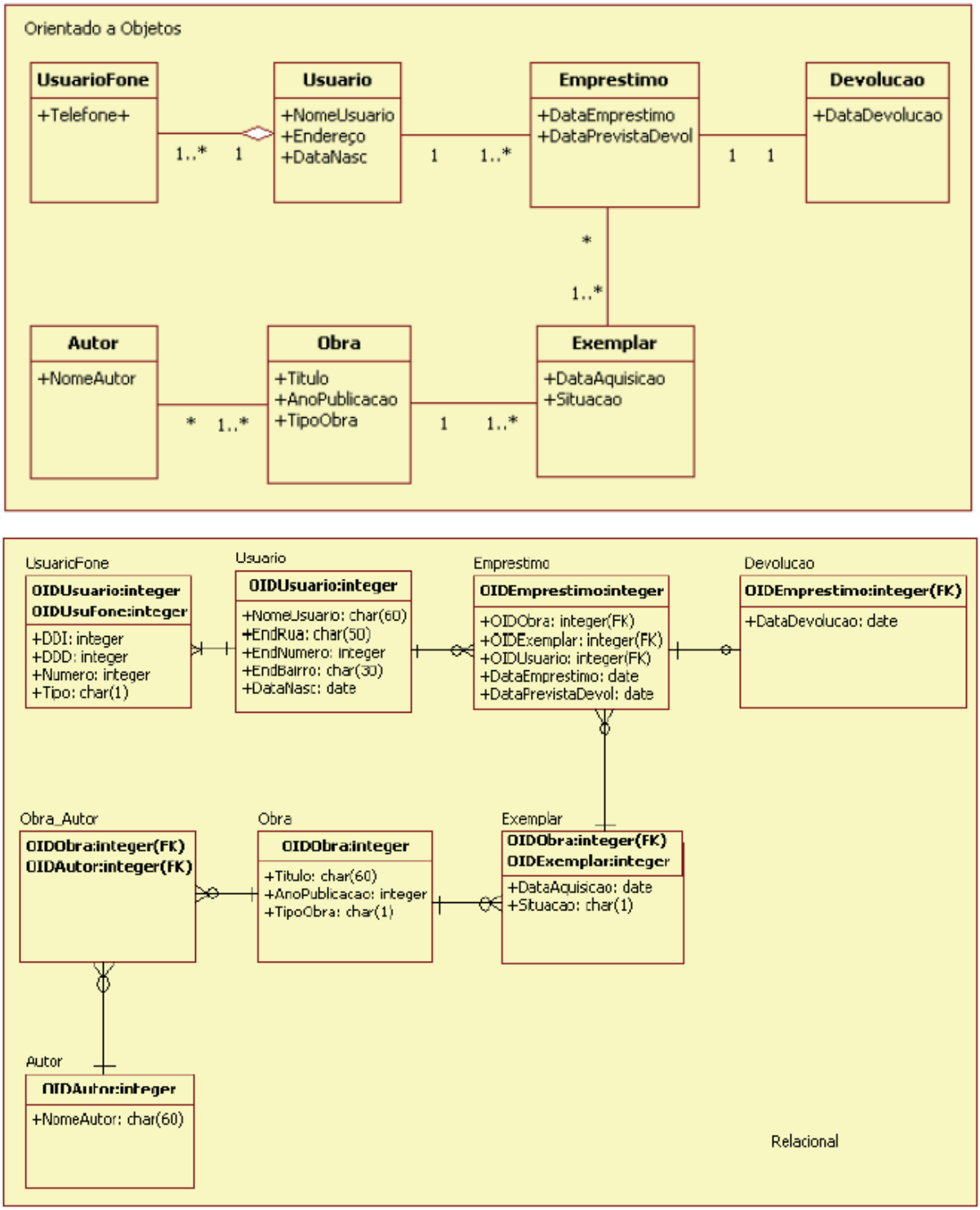


Figura 23. Exemplo de mapeamento
Fonte: Adaptado de BERNADI, A. (2007)

3.2.2.4 Tipos de testes com Hibernate

Testar é provavelmente a atividade mais importante que um desenvolvedor se engaja durante um dia do trabalho. Segundo Bauer e King (2007) testar determina a exatidão do sistema a partir de um ponto de vista funcional sempre em busca de performance e escalabilidade. Testes executados com êxito significam que todos os componentes e as camadas da aplicação interagem corretamente.

No caso do mapeamento objeto relacional, existe diferentes categorias de testes: teste unitário funcional, de integração e independente, todos têm um diferente objetivo e propósito, sendo necessário saber quando cada estratégia de teste é apropriada. Estes testes podem ser introduzidos no *framework TestNG* (ferramenta de testes para aplicações em Java).

3.2.2.4.1 Teste de aceitação

Segundo Bauer e King (2007) este tipo de teste não é necessariamente automatizado não sendo responsabilidade do desenvolvedor de aplicação ou projetista de sistemas. A aceitação é o estagio final do teste de um sistema, onde se verifica se o sistema satisfaz os requerimentos do projeto inicial. Este teste pode incluir qualquer método, deste a funcionalidade, performance e usabilidade.

3.2.2.4.2 Teste de performance

Bauer e King (2007) explicam que o teste de performance ou de carga sobre o sistema, com um alto número de usuários concorrentes, principalmente com uma carga igual ou maior da que é esperada uma vez que o *software* roda em produção.

3.2.2.4.3 Teste unitário de lógica

Conforme Bauer e King (2007) estes testes consideram um único pedaço de funcionalidade, frequentemente somente um método de negocio (por exemplo, se o lance mais alto ganha o leilão). Neste caso, se uma rotina é testada como uma única unidade, ela é testada independentemente de qualquer outra rotina do sistema.

3.2.2.4.4 Teste unitário de integração

Segundo Bauer e King (2007) o teste de integração determina se a comunicação entre os componentes, serviços e subsistemas do software funcionam como esperado. Num contexto de gerenciamento de transação e de gerenciamento de informações, o teste unitário de integração pode testar se o software funciona corretamente com o banco de dados (por exemplo, se um lance recém feito esta salvo corretamente no banco de dados).

3.2.2.4.5 Teste unitário funcional

Conforme Bauer e King (2007) o teste funcional verifica todo o caso de uso de todos os componentes da aplicação necessários para completar esse caso de uso em particular. Este teste pode incluir fluxo de trabalho da aplicação e a interface com o usuário (por exemplo, simular como o usuário deve ser logado antes de dar um novo lance para um item de um leilão).

4 BANCO DE DADOS ORIENTADO A OBJETOS

O mapeamento objeto-relacional nada mais é do que a tradução entre os dois modelos de LPOO e SGBDR, devido ao fato de que eles são diferentes.

Como foi visto no inicio do Capítulo 3, em todas as traduções, existem perdas de paradigmas de programação OO ou o programador acaba perdendo muito mais tempo implementando as classes de mapeamento objeto relacional.

A Figura 24 ilustra o uso do mapeamento objeto relacional quando é usada uma linguagem OO e um banco SGBDR, fazendo um comparativo de o uso de uma linguagem OO com um banco SGBDOO.

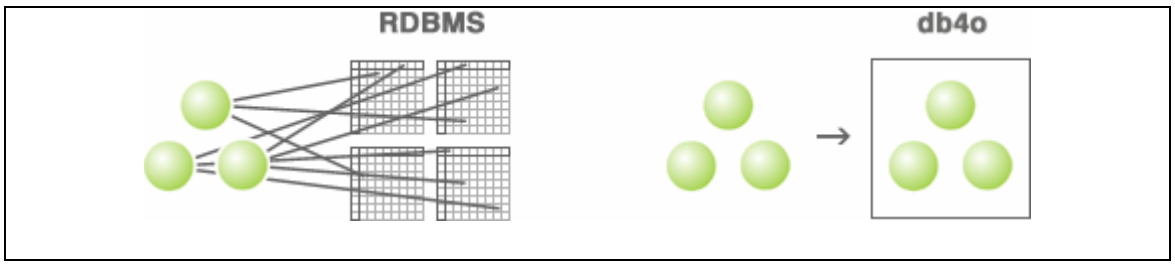


Figura 24. Comparação entre um mapeamento objeto relacional com dados persistentes em um SGBDO
Fonte: DB4Objects (2008)

Segundo Lima (2007) parte do problema de se ter que usar o mapeamento objeto relacional poderia ser resolvido ao se usar uma linguagem de programação orientada a objetos com um banco de dados orientados a objetos.

Existem bancos de dados orientados a objetos que mudam a visão de armazenamento e busca de informação, como o banco de dados Caché da Intersystems e o banco de dados db4o *open source object database* sob licença GPL. Ambos os banco de dados reduzem drasticamente o esforço do programador para se armazenar dados no banco de dados. O armazenamento é quase que instantâneo quando comparado com o armazenamento em um banco de dados relacional.

4.1 BANCO DE DADOS CACHÉ – INTERSYSTEMS

O Caché é um banco de dados pós-relacional de alta performance da empresa Intersystems. Foi introduzido no mercado no final do ano de 1997. Segundo Intersystems (2008), este SGDBOO destina-se a transcender as limitações do modelo de banco de dados relacional, proporcionando uma ferramenta como forma de atualização para as milhares de aplicações que usam banco de dados relacionais, bem como o suporte para ferramentas de relatórios via SQL.

Este banco de dados permite a manipulação tanto de dados relacionais como de dados orientado a objetos. Conforme Intersystems (2008) o "relacional" parte do "pós-relacional", refere-se ao fato de o banco de dados Caché atender a todas as necessidades de um banco de dados relacional. Uma base de dados Caché pode ser acessada via e manipulada por padrões SQL via ODBC, JDBC, objeto ou métodos. Já o "pós" parte do "pós-relacional" refere-se ao fato que este SGDBOO oferece um conjunto de recursos que vão alem dos limites dos bancos de dados relacionais.

Os principais fundamentos do caché são:

a) capacidade de modelar dados como objetos: cada objeto criado e sincronizado automaticamente, de representação relacional, eliminando automaticamente toda o problema de impedância em informações de representação relacional e orientada a objetos para o ambiente de aplicação, bem como para reduzir a complexibilidade da modelagem relacional;
b) tipo de dados definidos pelos usuários;
c) a possibilidade do uso dos métodos de herança, incluindo polimorfismo, nos mecanismos de banco de dados;

d) a possibilidade de usar SQL para manipular entidades de objetos e relacionamentos;

e) a capacidade de intercalar SQL e orientação a objetos dentro de uma única aplicação, utilizando o comando que for mais adequando;

f) controle sobre o armazenamento físico e de *clustering* usado para armazenar informações, garantindo o máximo de desempenho sobre as aplicações.

Outra característica importante do Caché é a sua arquitetura unificada de dados, onde cada estrutura de dados tem uma definição única que funciona tanto como classe de objetos como tabelas relacionais.

4.1.1 Arquitetura Unificada de Dados Caché

A arquitetura unificada de dados do Caché permite modelar componentes como objetos. Os objetos por sua vez são organizados por classes que definem os dados (propriedades) e comportamento (métodos) do objeto.

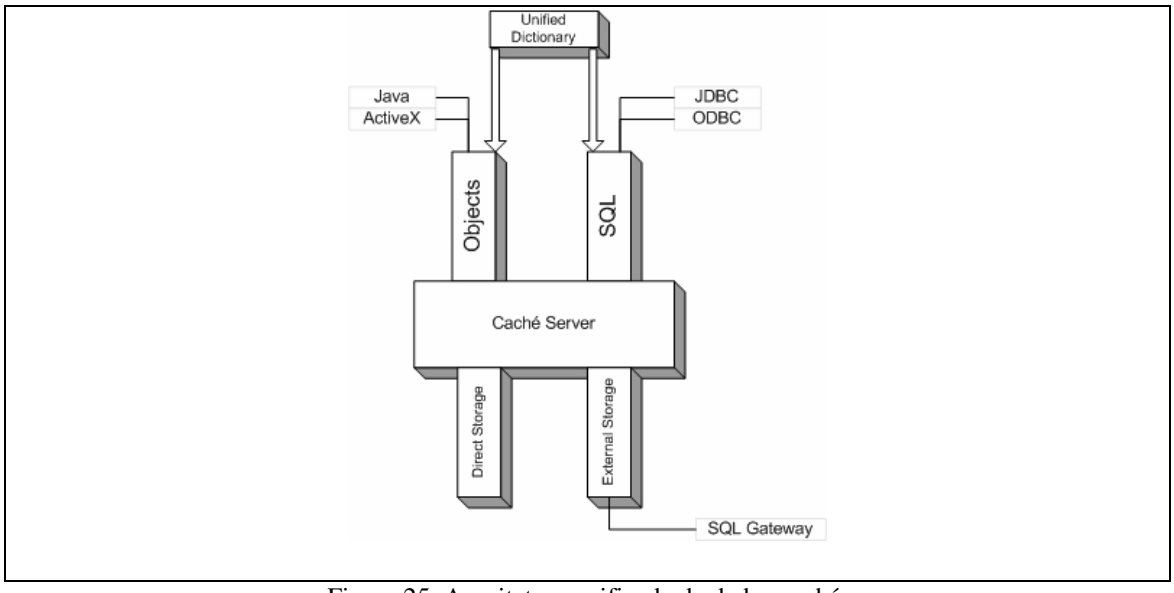


Figura 25. Arquitetura unificada de dados caché.
Fonte: Intersystems (2008).

Conforme definido na Figura 25, a definição de cada classe é armazenada na classe de dicionário caché (*Unifled Dictionary)*. Esta classe é um objeto próprio do banco de dados caché, podendo ser acessada usando programação orientada a objeto. Nesta classe, por meio de uma classe que faz a compilação, define-se a estrutura necessária para armazenamento de objetos persistentes e converte as definições das classes em paralelo com um conjunto de código executáveis que fornece tanto o acesso a objeto como relacional na estrutura de armazenamento, conforme Figura 26.

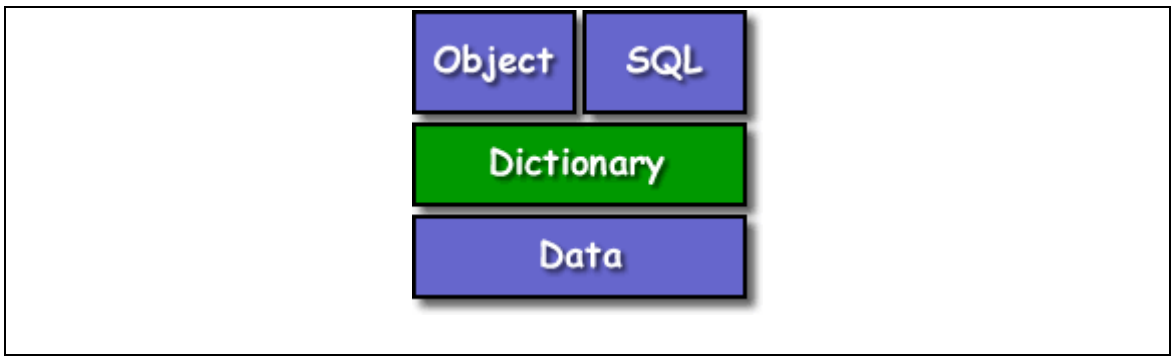


Figura 26. Dados relacionais e orientado a objetos persistentes no caché.
Fonte: Intersystems (2008)

Por meio desta arquitetura, os desenvolvimentos para o caminho objeto e relacional são eficientes e sincronizados um com o outro. Segundo Finn (2002) a estrutura unificada de dados do caché pode reduzir o problema de impedância de dados.

4.1.2 Diferença de impedância em dados no Caché

Segundo Intersystems (2008) a arquitetura unificada de dados do Caché é uma solução para o problema de diferença de impedância. A estrutura dos dados multidimensionais é uma forma natural de armazenar os tipos de dados que são características de banco de dados orientado a objetos, como também projetar estruturas multidimensionais como estrutura bidimensionais (tuplas) via SQL, conforme item 4.1.1.

Assim, tanto objetos como relações podem compartilhar simultaneamente dados multidimensionais sem a necessidade de mapear formato de dados evitando a diferença de impedância de dados, conforme Figura 27.

Figura 27. Acesso a dados no Caché
Fonte: Intersystems (2008)

4.1.3 Hibernate com Caché

Conforme visto no Capitulo 3, o Hibernate realiza o mapeamento objeto-relacional em uma LOO sobre SGBDR. Para muitas aplicações que utilizam banco de dados pós-relacional Caché, seria inútil utilizar uma camada de mapeamento, devido ao fato de que o Caché pode controlar os deferentes tipos de dados tanto em formato relacional como orientado a objetos.

O problema de persistência de objetos ocorre quando o Caché necessita de uma maneira simples de troca de objetos com um banco de dados relacional. Segundo Intersystems (2008) o Caché permite o uso do Hibernate para que a aplicação Java interaja com as classes orientadas a objeto Caché transparentemente entre um ou mais banco de dados relacionais. Neste ambiente, o Hibernate oferece as seguintes vantagens:

a) mapeamento objeto relacional: O Caché pode ser utilizado para persistir objetos em banco de dados relacionais, mais ocorrem problemas quando se utiliza mais de alguns tipos de objetos. Neste caso, o Hibernate automatiza o processo de conversão de formatos entre objeto e relacional, evitando perda de tempo para manutenção de desempenho da aplicação;
b) *Hibernate Query Language:* por meio da HQL, pode-se usar uma linguagem simples tanto de acesso Caché como de banco de dados relacionais. Isso reduz o potencial de problemas causados por variações entre os dialetos SQL. Também possibilita funcionalidades que facilitam a consulta de dados em formato de objetos em um modo natural.

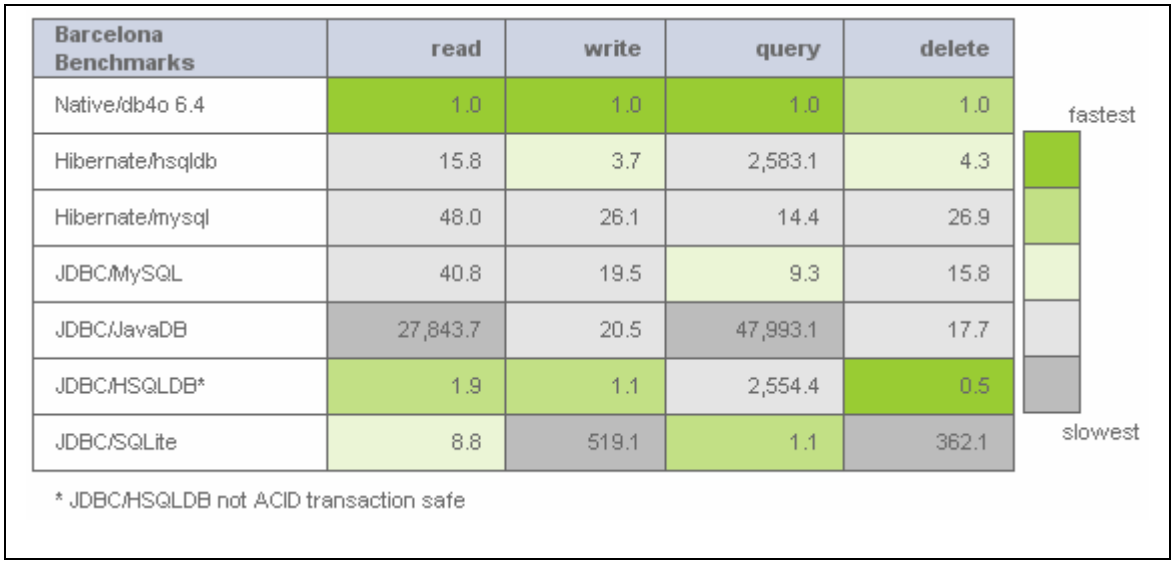
O Hibernate pode ser configurado a parir da versão do Intersystems Caché 2007.

4.2 BANCO DE OBJETOS DB4OBJECTS

DB4O é um banco de objetos gratuito, distribuído sobre licença GPL disponível para usuários Java e .NET. Segundo DB4O (2008) ele possibilita diminuir o tempo e custo de desenvolvimento de aplicações e permite alcançar ótimos níveis de desempenho.

Os objetos são armazenados nativamente, eliminando dificuldades extras de implementação e perda de performance com conversões para SQL. Segundo DB4O o suporte nativo as funcionalidades do banco de dados orientado a objetos, assim como as *querys* nativas e a replicação orientada a objetos, aumenta a performance e os ganhos de desempenho de linguagens orientada a objetos como Java.

Conforme Figura 28, o banco de dados DB4O elimina a troa OO por performance. Segundo DB4O (2008) permite armazenar as mais complexas estruturas de objetos com facilidade em altos níveis de performance. Avaliações demonstram que o DB4O pode ser 44x mais rápido que o Hibernate e MySQL, conforme Figura 28. Neste teste de desempenho foram utilizados 100 pesquisas simultâneas com 30000 objetos e 5 níveis de herança.



Barcelona Benchmarks	read	write	query	delete
Native/db4o 6.4	1.0	1.0	1.0	1.0
Hibernate/hsqldb	15.8	3.7	2,583.1	4.3
Hibernate/mysql	48.0	26.1	14.4	26.9
JDBC/MySQL	40.8	19.5	9.3	15.8
JDBC/JavaDB	27,843.7	20.5	47,993.1	17.7
JDBC/HSQLDB*	1.9	1.1	2,554.4	0.5
JDBC/SQLite	8.8	519.1	1.1	362.1

* JDBC/HSQLDB not ACID transaction safe

Figura 28. Teste de desempenho do banco de objetos DB4O
Fonte: DB4Objects (2008)

4.2.1 Diferença de impedância no DB4O

O DB4O por ser um banco de dados orientado a objetos não encontra problemas de diferença de impedância de dados. Segundo Finn (2002) este problema poderia ser totalmente eliminado armazenando dados em um banco puramente de objetos.

O problema de diferença de impedância ocorre quando se necessita executar uma consulta SQL neste banco de dados. Segundo Finn (2002) o SQL é a linguagem mais utilizada para consulta em banco de dados, sendo seu retorno em tuplas relacionais. Alguns fabricantes de banco de dados orientado a objetos utilizam a

linguagem *Object Query Linguage* (OQL). Oferecer suporte JDBC e ODBC e prever resultados relacionais seria uma solução para este problema. Quando não existe esta solução, utiliza-se mais uma vez o mapeamento, convertendo as consultas SQL para objeto tendo seu retorno como tuplas relacionais.

5 SIMULAÇÃO DA IMPEDÂNCIA DE DADOS

A pesquisa desenvolvida simula o uso da LOO Java com o *framework* ORM Hibernate configurado com os SGBDR's MySQL 5.0 (*Sun Microsystems*) e PostgreSQL 8.2.7 (*PostgreSQL Global Development Group*) mapeando todos os tipos de atributos e tuplas, assim como o uso de SGDBOO Caché 2008.01 (*Intersystems*) e DB4O (*db4objects, Inc.*) com o objetivo de verificar quais as melhores alternativas para evitar o problema de impedância de dados.

Todos os testes foram realizados em um único computador com a configuração conforme Tabela 3.

Item	Configuração
CPU	AMD Sempron™ 3100+ 1.6 Ghz
Memória	512 MB
HD	60 Gb
S.O.	Windows XP SP 2

Tabela 3. Quadro de configurações do computador

No SGBDR MySQL foi criado uma tupla com os seguintes atributos, conforme segue na Figura 29.

```
CREATE TABLE `cliente` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `nome_cliente` varchar(100) NOT NULL,
  `idade` int(11) default NULL,
  `endereco` varchar(100) NOT NULL,
  `bairro` varchar(45) NOT NULL,
  `cidade` varchar(45) NOT NULL,
  `uf` varchar(2) default NULL,
  `cep` int(8) unsigned default NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=13 DEFAULT CHARSET=latin1;
```

Figura 29. Tupla e seus atributos nos SBGDR MySQL

No SGBDR PostGreSQL foi criada uma tupla com os seguintes atributos conforme segue na Figura 30.

```
-- Table: cliente.cliente

-- DROP TABLE cliente.cliente;

CREATE TABLE cliente.cliente
(
  id integer NOT NULL,
  idade integer NOT NULL,
  endereco character(100) NOT NULL,
  nome_cliente character(100) NOT NULL,
  cep integer NOT NULL,
  cidade character(45) NOT NULL,
  uf character(2),
  CONSTRAINT id PRIMARY KEY (id)
)
WITH (OIDS=FALSE);
ALTER TABLE cliente.cliente OWNER TO postgres;
```

Figura 30. Tupla e seus atributos nos SBGDR PostGreSQL

Usando o Hibernate com os banco de dados relacionais MySQL e PostgreSQL pode-se ter uma visão global de testes no contexto de persistência e de gerenciamento de dados objeto relacional. Estes bancos de dados atendem todos os requisitos de mapeamento objeto relacional no Hibernate em seu ponto mais avançado. A escolha dos modelos relacionais deveu-se por ambos serem gratuitos, de licença GPL e ao fato de que são SGBD muito utilizados tanto em plataformas Linux quanto Windows.

O Hibernate Core 3.2.x foi a versão utilizada com os banco de dados relacionais. Segundo Bauer e King (2007) este modelo é o serviço básico para a persistência, com sua API nativa e seus métodos de mapeamento armazenados em arquivos XML. Pode-se ser usado o Hibernate Core independentemente de qualquer outro *framework* ou de qualquer ambiente de tempo de execução em particular com

todas as JDK´s. Com esta estrutura, o conjunto com as outras versões do Hibernate funcionara adequadamente.

5.1 HIBERNATE COM MYSQL

Ao usar o MySQL com Hibernate, toda a configuração básica de mapeamento foi realizada. Nenhum ponto importante foi encontrado nesta etapa do projeto. Os procedimentos foram os seguintes:

a) criação do projeto “teste_hibernate”: Um projeto foi criado, adicionando todas as bibliotecas necessárias da versão do Hibernate Core. As bibliotecas foram: *antlr.jar, asm.jar, asm-attrs.jars, c3p0.jar, cglib.jar, commons-collections.jar, commons-logging.jar, dom4j.jar, hibernate3.jar, jta.jar;*

b) adicionando o conector JDBC do banco de dados MySQL ao projeto: *mysql-connector-java-5.0-prodution-bin.jar*;

c) criado o arquivo *hibernate.cfg.xml* com as propriedades de conexão com o banco de dados MySQL, conforme a Figura 31;

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<hibernate-configuration>
    <session-factory name = "java:/hibernate/HibernateFactory">
        <property name="connection.driver_class">org.gjt.mm.mysql.Driver</property>
         <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/teste</property>
        <property name="connection.username">root</property>
        <property name="hibernate.connection.password">123456</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <mapping resource="xml/Cliente.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Figura 31. Arquivo de configurações do *framework* Hibernate com MySQL

d) criado arquivos de mapeamento objeto relacional, contendo suas classes e seus respectivos mapeamentos, conforme a Figura 32;

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
    <class name="beans.Cliente" table="cliente">
        <id name="id" column="id">
            <!-- Incrementa Chave Primaria -->
            <generator class="increment"/>
        </id>
        <!-- Nome -->
        <property name="nome" type="java.lang.String" column="nome_cliente" />
        <!-- Endereço -->
        <property name="endereco" type="java.lang.String" column="endereco" />
        <!-- Idade -->
        <property name="idade" type="int" column="idade" />
        <!-- CEP -->
        <property name="cep" type="int" column="cep" />
        <!-- Bairro -->
        <property name="bairro" type="java.lang.String" column="bairro" />
        <!-- Cidade -->
        <property name="cidade" type="java.lang.String" column="cidade" />
        <!-- UF -->
        <property name="uf" type="java.lang.String" column="uf" />
    </class>
</hibernate-mapping>
```

Figura 32. Arquivo XML de mapeamento objeto relacional

e) inseridos e escritos demais arquivos para funcionamento do Hibernate.

Aconteceram alguns erros nestas etapas do projeto. Por segurança, Segundo Bauer e King (2007) as bibliotecas do Hibernate Core somente compilam as rotinas após verificam a consistência dos arquivos de configuração XML (Figura 31) e arquivo de mapeamento XML (Figura 32). Na compilação dos códigos fontes é necessário que o programa de desenvolvimento esteja conectado a Internet, para validar o XML seguindo regras específicas do arquivo "*hibernate-mapping-2.0.dtd*" e arquivo "*hibernate-configuration-2.0.dtd*" conforme Figura 31 e 32 respectivamente.

Outros problemas encontrados foram relacionados às transações seguras da aplicação. O Hibernate por padrão, controla todas as transações seguras do banco de dados, exigindo que se termine com segurança todo o processo de persistência da informação, caso contrário o processo é desfeito. A solução para este problema, foi à

inserção de comandos de controle de transações e como conseqüência o pleno funcionamento do Hibernate, conforme Figura 33.

```
package dao;

import beans.Cliente;
import org.hibernate.Session;
/**
 *
 * @author Daniel Plácido
 */
public class ClienteBD {

    /** Creates a new instance of ClienteBD */
    public ClienteBD() {
    }

    public void salvar(Cliente c){
        Session session = ConexaoBancoDeDados.getSession();
        ConexaoBancoDeDados.beginTransaction();
        session.save(c);
        ConexaoBancoDeDados.commitTransaction();
        ConexaoBancoDeDados.closeSession();
    }
}
```

Figura 33. Transações seguras no Hibernate

Com o funcionamento do hibernate, foi desenvolvida uma interface de inserção de dados, conforme Figura 34.

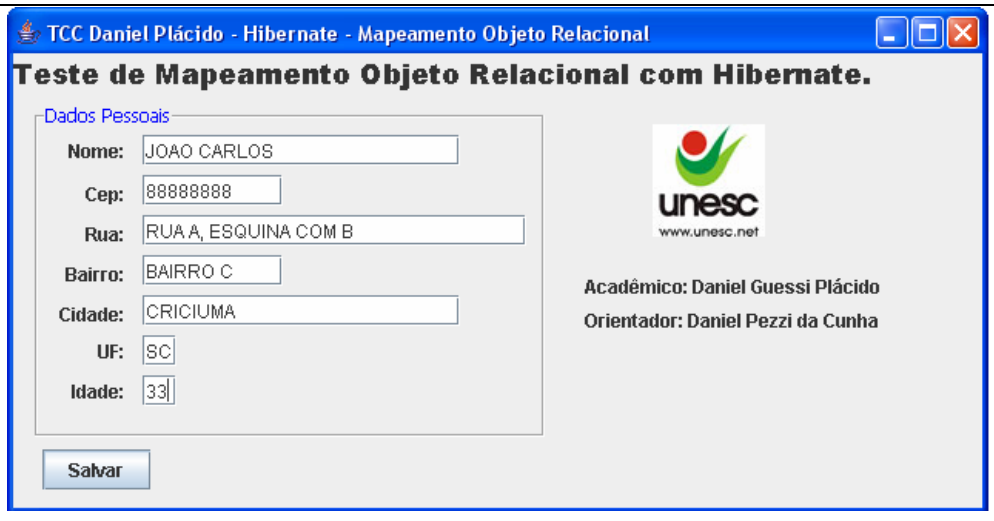


Figura 34. Interface de inserção de dados

Com a aplicação funcionando, foi inserido um registro sendo mapeado para o banco de dados MySQL. Todos os tipos de atributos foram mapeados para a tupla do banco de dados, sendo o registro salvo com sucesso, conforme Figura 35.

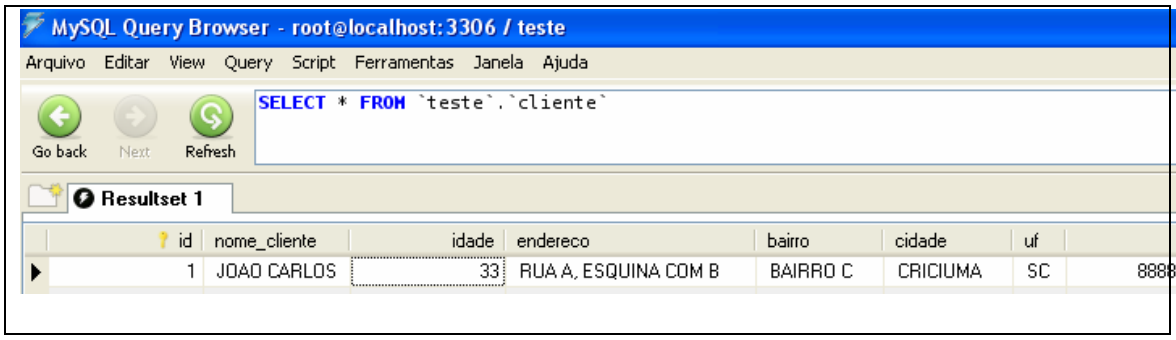


Figura 35. Resultado da inserção de dados no MySQL

5.2 HIBERNATE COM POSTGRESQL

Mudando-se de banco de dados na aplicação sem perder o mapeamento, apenas configurou-se o dialeto no arquivo de configuração do Hibernate "*hibernate.cfg.xml*". Outras propriedades de usuário e senha de banco de dados também precisam ser alteradas, conforme Figura 36.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
    PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-2.0.dtd">

<!--<hibernate-configuration>
    <session-factory name = "java:/hibernate/HibernateFactory">
        <property name="connection.driver_class">org.gjt.mm.mysql.Driver</property>
         <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/teste</property>
        <property name="connection.username">root</property>
        <property name="hibernate.connection.password">123456</property>
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <mapping resource="xml/Cliente.hbm.xml"/>
    </session-factory>
</hibernate-configuration>-->

<hibernate-configuration>
    <session-factory name = "java:/hibernate/HibernateFactory">
        <property name="connection.driver_class">org.postgresql.Driver</property>
         <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/teste</property>
        <property name="connection.username">postgres</property>
        <property name="hibernate.connection.password">123456</property>
        <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
        <mapping resource="xml/Cliente.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Figura 36. Configurando conexão com PostGreSQL

Com a configuração alterada, o Hibernate mapeia todos os atributos da tupla de acordo com o dialeto usado. A inserção dos dados no PostGreSQL também foi de acordo com a Figura 34, tendo como retorno a inserção correta dos dados, conforme Figura 37.

	id [PK] integer	nome_cliente character(100)	endereco character(100)	idade integer	cep integer	cidade character(45	uf character(2)	bairro character(45
1	1	JOAO CARLOS	RUA A, ESQUINA COM B	33	88888888	CRICIUMA	SC	BAIRRO C
*								

Figura 37. Resultado da inserção de dados no PostGreSQL

Com o funcionamento do Hibernate, observou-se a independência do tipo de banco de dados para a aplicação. Todas as tuplas e seus atributos foram mapeados com sucesso. Com o mapeamento objeto relacional nestas situações evita-se o problema de impedância de dados em banco de dados.

5.3 O USO DO CACHE INTERSYSTEMS

Para testes do Caché Intersystems com a LOO Java, foi utilizada a biblioteca *Japaleño* para armazenamento de objetos *Plain Old Java Objects* (POJO), sem mapeamento e de fácil acesso a partir da aplicação Java.

Foi criada uma classe caché “basic.Contact” com os campos *concactType* e *name* conforme Figura 38.

```
Class basic.Contact Extends %Persistent [ SqlTableName = Contact ]
{

Property concactType As %Library.String(JAVATYPE = "java.lang.String", MAXLEN = 4096);

Property name As %Library.String(JAVATYPE = "java.lang.String", MAXLEN = 4096);

XData JavaBlock
  {
<JavaBlock>
   <Package implementation="basic.cache" pojo="basic"></Package>
   <UseSameNames>false</UseSameNames>
   <Name implementation="Contact" pojo="Contact"></Name>
   <ResolveNameCollisions>false</ResolveNameCollisions>
   <EagerFetchRequired>true</EagerFetchRequired>
  </JavaBlock>
  }

}
```

Figura 38. Classe Caché Contact

Cada tipo de dado dos atributos da LOO Java pode ser atribuídos nas propriedades da classe “basic.Contact”, conforme pode ser observado na Figura 38. Toda a conexão com o banco de dados acontece na própria linguagem Java via JDBC. Foram criadas classes Java para persistir a informação no banco de dados conforme Figura 39 e Figura 40.

```
package cache4;
import com.jalapeno.annotations.*;
public class Contact {
    public String concactType = null;
    public String name = null;
    public String showProps () throws Exception {
        return " Contato:" + name + " Tipo de Contato:" + concactType;
    }
}
```

Figura 39. Classe POJO responsável por estruturar persistência

```
package cache4;
import com.jalapeno.ApplicationContext;
import com.jalapeno.ObjectManager;
import com.jalapeno.annotations.Transient;
@Transient
public class tinyPojo {
public static ObjectManager objManager;
public static String url="jdbc:Cache://localhost:1972/TESTE";
public static String user = "_SYSTEM";
public static String pwd = "SYS";
public static void main( String[] args ) throws Exception
{
    objManager = ApplicationContext.createObjectManager(url,user,pwd);

    Contact newEmp = new Contact ();
    objManager.startTransaction ();
    Object myId = null;

    /* Create an Employee object in memory */
    newEmp.name = "Daniel Plácido";
    newEmp.concactType = "Comercial";
    myId = objManager.getId(newEmp);
    objManager.insert(newEmp, true);
    newEmp = (Contact)objManager.openById(Contact.class, myId);
    objManager.save (newEmp, true);
    objManager.commit();
    objManager.close();
}
}
```

Figura 40. Classe Java de persistência de dados

A persistência para os tipos de dados dos atributos orientado a objetos funciona perfeitamente quando se é usado uma linguagem de programação orientada a objetos no Caché. A programação fica mais completa, sendo simples e de fácil entendimento conforme Figura 40. Como resultado desta execução, obteve-se a persistência do objetos na propriedade das classes do banco de dados Caché conforme Figura 41.

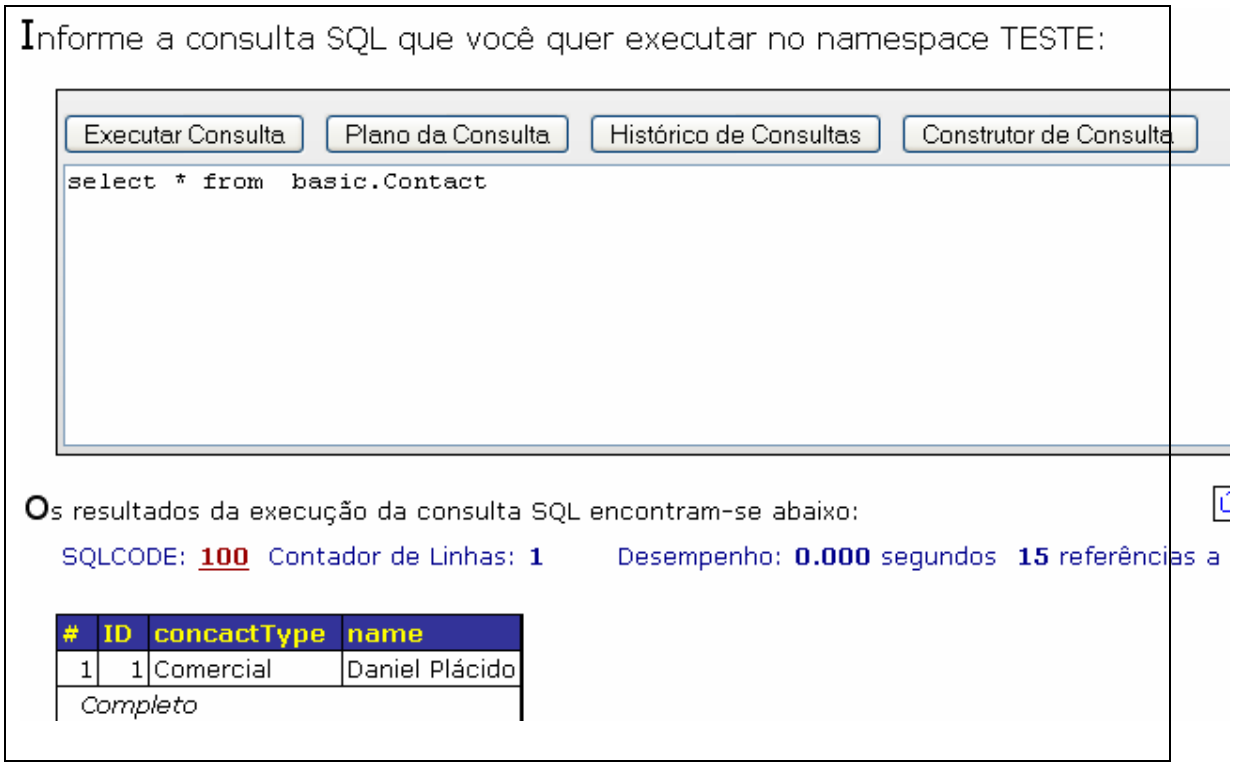


Figura 41. Resultado da persistência de dados no Caché

Vale lembrar que toda a persistência de dados no caché assim como o limite de usuários no sistema é controlado por sessão, então é importante sempre fechar a mesma no fim da execução do programa, conforme Figura 41.

5.4 O USO DO DB4O

Neste banco de dados não existe um processo formal de criação de banco de dados. À medida que as classes na linguagem Java forem sendo instanciadas, o DB4O cria a persistência das informações. Todas as funções de manipulação deste banco de dados são utilizadas por meio de um método chamado "*ObjectContainer*", responsável por atualizar, excluir, salvar e pesquisar informações no banco de dados.

Para os testes foi criada uma classe "Cliente" de persistência de informação conforme Figura 42.

```
package db4o;
public class Cliente{
    private String nome;
    private String endereco;
    public Cliente(String nome,String endereco) {
        this.nome=nome;
        this.endereco=endereco;
    }
    public String getName() {
        return nome;
    }
    public String getEndereco() {
        return endereco;
    }
    public String toString() {
        return nome+"/"+endereco;
    }
}
```

Figura 42. Classe Cliente no DB4O

Para conexão com o banco de dados, foi criada uma classe “acessoDb4o”, onde é aberto o arquivo do banco de dados. Se este arquivo não existir, ele é criado automaticamente. Logo em seguida foi criada uma classe “salvarCliente” de persistência no banco de dados. A persistência é instantânea conforme Figura 43.

```
public static void accessoDb4o(){
    ObjectContainer db=Db4o.openFile(Util.DB4OCLIENTE);
    try {
        salvarCliente(db);
    }
    finally {
        db.close();
    }
}
public static void salvarCliente(ObjectContainer db) {
    Cliente clientel=new Cliente("Daniel Plácido","Rua Julio Werner");
    db.store(clientel);
    System.out.println("Salvo "+clientel);
}
```

Figura 43. Classes de conexão e persistência do DB4O

A estrutura da classe Cliente cria a organização dos objetos instanciados e persistidos no arquivo “DB4OCLIENTE”, tornando o processo dinâmico e fácil de ser utilizado.

Com toda a facilidade para persistir dados, notou-se que neste banco de dados não há um controle de consistência de informações, como chave primárias e estrangeiras. São necessários métodos paralelos para controle de consistência, dificultando o trabalho de desenvolvimento e coerência das informações.

Mesmo com a integridade podendo ser afetada, pode-se controlar perfeitamente por meio de métodos de verificação de objetos, sendo que a LOO permite um dinamismo amplo das regras de negócio.

Os tipos de dados se adaptam perfeitamente a estrutura da LOO Java, sendo que cada tipo de dado que se é usado na linguagem pode ser reutilizado como estrutura do tipo de objeto.

No decorrer dos testes com o banco de dados não se encontrou nenhum problema para persistir dados.

5.5 TABELA COMPARATIVA

Na Tabela 4 pode-se observar uma comparação entre os bancos de dados testados em relação ao problema de impedância de dados.

	Bancos de Dados			
Comparações	**MySQL**	**PostGreSQL**	**Caché**	**DB4Objects**
Atende requisitos OO			X	X
Atende requisitos da LOO			X	X
Não possui problema de impedância				X
Necessita de mapeamento	X	X		
Padronização dos dados			X	X
Problemas encontrados	X	X		X

Tabela 4. Comparação entre bancos de dados em relação ao problema de impedância de dados

CONCLUSÕES

Com o objeto de estudo dessa pesquisa testou-se o uso de ferramentas de mapeamento objeto-relacional com LOO e bancos de dados relacionais, assim como o teste de LOO com banco de dados orientado a objetos e objeto-relacional, verificando como se comporta cada tipo de banco de dados com o problema de impedância de informações.

Pode-se observar que o uso de LOO com bancos de dados relacionais limita uma aplicação a estrutura do banco de dados utilizado. Neste caso, o uso de uma ferramenta de mapeamento objeto relacional se faz necessária, mapeando cada tipo de dado da LOO ao atributo no banco de dados. Com isso, temos uma ferramenta independente da tecnologia de banco de dados relacional adotada, facilitando mudanças quando necessário. Mesmo assim, notou-se que o desenvolvedor perde um bom tempo de desenvolvimento mapeando classes e objetos para tuplas e atributos bem como reescrevendo algumas regras de negócio. Este fator deve ser considerando ao projetar um sistema e deverá estar no cronograma de desenvolvimento para evitar possíveis problemas de tempo de construção da aplicação.

Já nos testes com banco de dados orientado e LOO pode-se observar que não houve dificuldades de persistir o mesmo tipo de dado da linguagem com o banco de dados. Especificamente o DB4O utiliza o mesmo tipo de dado da linguagem Java, tornando a diferença de dados nula. Contudo, neste banco de dados não é possível executar comandos SQL, sendo que estes comandos são sem duvida os mais utilizados em pesquisas nos bancos de dados, tornando o uso do DB4O limitado.

Com o banco de dados pós-relacional Caché, notou-se uma grande diferença no controle do problema de impedância de dados. Com sua arquitetura unificada de dados é possível trabalhar com objetos e atributos atendendo todos os seus tipos ao

mesmo tempo, sendo que a tradução entre eles é automatizada, além de permitir consultas SQL. O Caché ainda permite a configuração do Hibernate sobre o banco de dados, podendo-se converter dados de objetos para relacional eliminando o problema de impedância de dados em banco de dados.

Vale lembrar que a escolha correta do tipo do banco de dados usado em uma determinada aplicação, geralmente não cabe a equipe de desenvolvimento e sim ao projetista do sistema. Selecionar o banco de dados correto para o tipo de aplicação, seja ele relacional ou orientado a objeto pode apresentar características de sucesso da ferramenta, mesmo que ainda haja problemas de padronização de dados.

TRABALHOS FUTUROS

Com o estudo realizado sobre os bancos de dados, todas as características de manipulação dos tipos de dados, podem-se perceber alguns pontos fundamentais para futuras pesquisas em persistência, padronização e solução de impedância de dados tais como:

a) comparação de desempenho ao se usar uma determinada aplicação construída com linguagem orientada a objetos com banco de dados relacionais x linguagem orientada a objetos com uma ferramenta de mapeamento objeto relacional;
b) utilização do banco de dados caché com suporte a ferramenta de mapeamento objeto relacional Hibernate;
c) estudo do comportamento de mapeamento objeto relacional em diversas linguagens;
d) testes de associações em ferramentas de mapeamento objeto relacional;

e) benefícios do uso de banco de dados pós relacional do tratamento de impedância de dados em banco de dados.

REFERÊNCIAS

BERNARDI, Diogo Alencar. **SQL Magazine – Diga adeus a SQL com a nova JPA**, Rio de Janeiro, Edição 40, ano 5, p. 46-51, Abril/2007.

CARUSO NETO, José André; MORAIS, Gustavo de Almeida. **Processamento de dados:** para as áreas de ciência da computação e tecnologia em processamento de dados. São Paulo: Érica, 1999.

DATE, C. J. **Introdução a sistemas de bancos de dados.** 4. ed. Rio de Janeiro: Campus, 1990.

DATE, C. J. **Introdução a sistemas de bancos de dados.** 8. ed. Rio de Janeiro: Elsevier, 2003.

DB4O. **DB4O Object**. Disponível em: <http://www.db4o.com>. Acesso em: 5 mar. 2008.

DOEDERLEIN, Osvaldo. Persistência Turbinada. **Java magazine**, Rio de Janeiro, Edição 25 ano 3, p. 28-41, Junho/2005.

FINN, Mary. **Como Resolver a Impedância em Banco de Dados**. Disponível em: <http://www.intersystems.com.br> Acessado em: 3 ago. 2007.

HARRINGTON, Jan L. **Projetos de bancos de dados relacionais: teoria e pratica.** 2. ed. Rio de Janeiro: Campus, 2002.

HERNANDEZ, Michael J. **Aprenda a Projetar seu próprio banco de dados.** São Paulo: Makron books, 2000.

HIBERNATE. **Hibernate Mapping Object Relacional**. Disponível em: <http://www.hibernate.org>. Acesso em: 5 mar. 2008.

INTERSYSTEMSBrasil. **InterSystems do Brasil**. Disponível em: <http://www.intersystems.com.br>. Acesso em: 1 mar. 2008

MACHADO, Felipe Nery Rodrigues; ABREU, Maurício Pereira de. **Projeto de banco de dados: uma visão prática.** 8.ed São Paulo: Érica, 2002.

MARTIN, James. **Princípios de Análise e projeto baseados em objetos.** 4. ed. Rio de Janeiro: Campus, 1994.

MARTIN, James; ODELL, James J. **Análise e projeto orientados a objetos.** São Paulo: Makron Books, 1996.

ODBMS. **Object Database Management Systems:** The Resource Portal for Education and Research. Disponível em: <http://www.odbms.org>. Acesso em: 10 mai. 2007.

ODMG. **Object Data Management Group.** Disponível em: <http://www.odmg.org>. Acesso em: 10 mar. 2008.

OMG. **Object Management Group.** Disponível em: <http://www.omg.org>. Acesso em: 10 mar. 2008.

TAKAI, Osvaldo Kotaro; ITALIANO, Isabel Cristina; FERREIRA, João Eduardo. **Introdução a banco de dados**. Disponível em: <http://www.ime.usp.br/~jef/apostila.pdf> Acessado em: 5 fev. 2008.

PENDER, Tom. **UML a Bíblia.** Rio de Janeiro: Elsevier, 2004.

PFLEEGER, Shari Lawrence. **Engenharia de software:** teoria e prática. 2. ed. São Paulo: Prentice Hall, 2004.

PREISS, Bruno R. **Estruturas de dados e algoritmos: Padrões de projetos orientados a objeto com Java.** Rio de Janeiro: Elsevier, 2000.

RAO, Bindu R. **Object-Oriented databases:** technology, applications, and products. Singapore: McGRAW-HILL, 1995.

SETZER, Waldemar W; SILVA, Flavio Soares Correa da. **Banco de dados.** São Paulo: Ed. Edgard Blucher, 2005.

SILBERSCHATZ, Abraham; KORTH, Henry. **Sistemas de Banco de Dados**. São Paulo: Ed. Makron Books, 1995.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados.** 3. ed. São Paulo: Makron Books, 1999.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados**. Rio de Janeiro: Elsevier, 2006.

SILVA, Arídio. **Dominando a tecnologia de objetos.** Rio de Janeiro: BookExpress, 2002.

SUN. **Lesson: Object-Oriented Programming Concepts (The Java™ Tutorials > Learning the Java Language).** Disponível em: <http://java.sun.com/docs/books/tutorial/java/concepts/>. Acesso em: 8 dez. 2007.

KIRSTEN, W et al. **Object-Oriented Application Development.** Using the Caché Postrelational Database. 2. ed. New York: Springer, 2002.

YOURDON, Edward; ARGILA, Carl. **Análise e Projeto Orientados a Objetos.** São Paulo: Makron Books, 1999.