Persistência de Objetos Utilizando JAVA e DB4O: Uma Alternativa à Modelagem Objeto-Relacional

Walisson Pereira de Sousa[1], Virginia de Sousa Venega[1]

[1]Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (IFTO)
Av. Joaquim Teotônio Segurado – 77.020-450 – Palmas – TO – Brasil

walisson.sousa@ifto.edu.br,virginia.venega@ifto.edu.br

Abstract. *This paper intend to show an alternative to using of Hibernate, for persistence of information in related databases, with object oriented languages. For that, DB4O is going to be used. With that, it's possible to persist objects without "dismount" orientation, eliminating bad uses of the font code and getting more speed in the software development process.*

Resumo. *O presente artigo visa mostrar uma alternativa ao uso do Hibernate, na persistência de informações em bancos de dados relacionais, em conjunto com linguagens orientadas a objetos. Para tal, será utilizado o DB4O. Com ele, torna-se possível a persistência de objetos, sem ter que se desfazer da Orientação, eliminando improvisos no código fonte e ganhando maior velocidade no processo de desenvolvimento de softwares.*

1. Introdução

Desde a época em que eram utilizados arquivos de texto para armazenamento dos dados – meados dos anos 1960 – até a utilização de Sistemas Gerenciadores de Banco de Dados (SGBD) para alocação e gerenciamento de informações, houve um avanço considerável de conceitos e métodos relacionados ao armazenamento de informações. Surgiram diversas tecnologias que vieram a aprimorar o armazenamento bem como o processo de persistência de dados.

Codd (1970) desenvolveu o que veio a ser o primeiro banco de dados relacional, batizado de Sistema R. Através de seus estudos, ele percebeu que o aumento incremental das informações poderia criar um gargalo nos sistemas gerenciadores atuais. Logo, foi proposto a criação de um sistema que agrupasse maior quantidade de dados em bases compartilhadas e que facilitasse, quando necessário, a recuperação dessas informações.

Foi apresentado por Peter Chen (1976) o Modelo Entidade-Relacionamento, que propunha métodos de modelagem de software através de atributos, entidades e relacionamentos. Através destes, poder-se-ia modelar um sistema no chamado Diagrama Entidade-Relacionamento (DER). Esse método tem facilitado a compreensão e abstração do sistema para que ele tenha sua persistência facilitada ao banco de dados. Chen define entidades como "coisas" que podem ser classificadas, agrupadas e distinguidas – citando como exemplos uma pessoa, uma empresa ou um evento qualquer –, e relacionamentos como associações entre entidades. Já os atributos, são descritos como detentores de características das entidades. O autor defende que o mundo é composto por entidades e relacionamentos, dando uma visão mais real (e natural) do

sistema a ser construído. Dessa forma, a criação do modelo em questão tem propiciado aos desenvolvedores um avanço considerável na criação e manutenção das diversas aplicações existentes.

Modelos de programação também têm sua importância; eles definem o estilo no qual o código será escrito. O paradigma procedural, que é caracterizado por executar procedimentos, iterações e decisões, foi o precursor do paradigma Orientado a Objetos (OO) [JUNGTHON e GOULART 2007]. O conceito de orientação a Objetos não é novo. Em meados da década de 60, criada por Nygaard e Dahl, a linguagem Simula 67 já utilizava alguns dos conceitos desse modelo (classes, herança). Em 1972, a linguagem Smalltalk, a primeira totalmente orientada a objetos, foi lançada. Mas foi com a linguagem Java que esse paradigma ganhou força e notoriedade. Para Mendes (1998, p. 18):

> O paradigma de orientação a objetos traz um enfoque diferente da programação estruturada, no sentido de adotar formas mais próximas do mecanismo humano para gerenciar a complexidade de um sistema. Nesse paradigma, o mundo real é visto como sendo constituído de objetos autônomos, concorrentes, que interagem entre si, e cada objeto tem seu próprio estado (atributos) e comportamento (métodos), semelhante a seu correspondente no mundo real.

No entanto, a grande maioria dos SGBDs atuais não dá suporte à orientação a objetos, utilizando-se do Modelo Relacional para armazenamento de informações. Como boa parte das linguagens de programação, hoje em dia, é orientada a objetos (ou suporta), caso o programador queira simplificar a persistência de dados nesses servidores, é necessário utilizar o mapeamento Objeto-Relacional. Para a linguagem Java existe o Hibernate.

Nesse framework, o desenvolvedor deverá especificar as classes em arquivos *eXtensible Markup Language* (XML) para coincidir com as tabelas do banco de dados. Em suma, ele garante métodos de acesso à base de dados (inserção, remoção, alteração e consulta) sem que o programador tenha que desmanchar a programação orientada a objetos, utilizada durante a criação do programa. A abstração fica a cargo dos métodos disponíveis, porém com alguns efeitos colaterais. A Figura 1 explicita o funcionamento do framework, onde as bolinhas flutuantes representam os objetos criados durante a execução da aplicação. O Hibernate funciona como uma camada intermediária, de alto nível e prática.

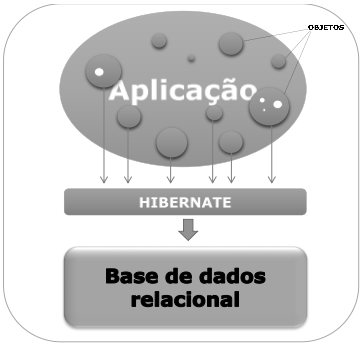

Figura 1. Funcionamento do Hibernate[1]

[1] Fonte: Figura elaborada pelo autor.

Apesar da melhoria visual no código, no mapeamento automatizado (possível através de alguns plugins), acarreta um leve aumento de tempo nas operações realizadas no banco. Mesmo assim, muitos programadores optam por utilizá-lo. Logo, o presente artigo tem por objetivo mostrar uma alternativa ao uso do Hibernate para aqueles que utilizam o paradigma orientado a objetos em conjunto com esse framework, garantindo melhor desempenho, velocidade e qualidade no processo de desenvolvimento de software.

2. Banco de Objetos

Utilizar linguagem de programação orientada a objetos em conjunto com banco de dados relacional não é uma tarefa simples. É necessário desfazer a orientação a objetos na hora da utilização do banco ou utilizar mapeadores objeto-relacional a fim de realizar a persistências dos dados. A primeira opção é mais eficaz, já que a utilização do *Structured Query Language* (SLQ) nativo garante menor tempo de resposta no acesso ao banco que o mapeamento, porém deixa o código fora dos padrões de OO. Com a segunda alternativa é possível manter todos esses requisitos intactos. Porém, ao utilizar ferramentas de mapeamento (o Hibernate pode ser citado aqui por ser o mais famoso mapeador), por mais que facilite e simplifique o processo de persistência, coincidindo as classes OO em tabelas SQL, tem-se um aumento significativo nas operações realizadas no banco de dados. A utilização do DB4O como alternativa a esse modelo visa simplificar o processo de persistência de dados.

O guia definitivo do DB4O [Paterson, Edlich, Hörning e Hörning 2006], mostra que ele se trata de um banco de objetos, de código aberto e gratuito, onde os mesmos são armazenados sem a necessidade de desfazer a orientação a objetos ou utilizar mecanismos de adaptação. Ele não é um Sistema Gerenciador de Banco de Dados (SGBD) pois não traz mecanismos de gerenciamento, como: segurança, controle de usuários, visões, etc. Os objetos são alocados em *pools* [2]e são resgatados de acordo com a necessidade, facilitando o trabalho do programador. O funcionamento do DB4O pode ser conferido na Figura 2.

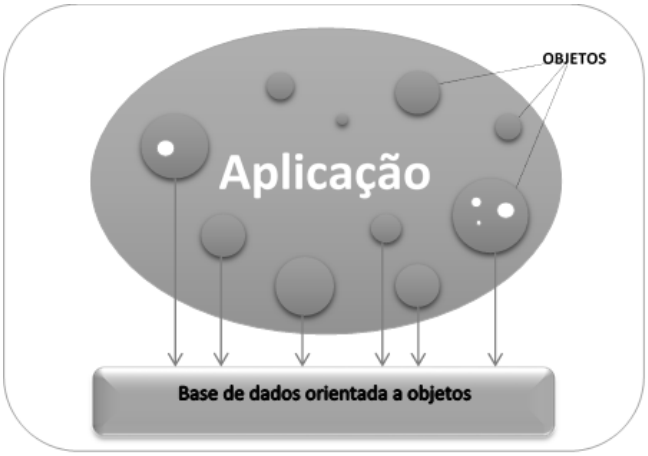

Figura 2. Funcionamento do DB4O[3]

[2] *Pools*: “Piscinas” ou conjuntos de objetos.
[3] Fonte: Figura elaborada pelo autor.

A instalação da base consiste em apenas adicionar um arquivo **J**ava **Ar**chive (JAR) ao projeto, o que garantirá acesso à poderosa *Application Programming Interface* (API). A mesma possui um conjunto de métodos que permitem ao programador realizar as diversas operações presentes nos bancos de dados relacionais. Outro ponto positivo é a possibilidade de se usar *queries* nativas nas consultas ao banco de dados. Utilizando-se o DB4O, o banco e o programa são executados em um mesmo processo, o que torna ideal o uso em aplicações com recursos limitados. As informações são armazenadas em arquivo em diretório especificado no momento da criação do mesmo.

Utilizando como exemplo um sistema genérico, será mostrado a seguir o simples funcionamento do DB4O através de um esqueleto de um software de controle de clientes. Para tal, foram criadas as classes java – linguagem escolhida devido à difusão e facilidade de entendimento da mesma – Usuario, Cliente e Endereco, ignorando as formalidades necessárias:

Usuario.java

```
1.public class Usuario {
2.int id;
3.String nome, usuario, senha;
4.public Usuario() { }
5.public Usuario(Integer id) {
6.this.id = id;
7.}
8.public Usuario(Integer id, String nome, String usuario, String senha) {
9.this.id = id;
10.this.nome = nome;
11.this.usuario = usuario;
12.this.senha = senha;
13.}
14.}
```

Cliente.java

```
1.public class Cliente {
2.    int id;
3.    Usuario usuario;
4.    String nome;
5.    ArrayList<Endereco> enderecos;
6.    ArrayList<Telefone> telefones;
7.
8.    Cliente(){}
9.
10.    Cliente(int id){
11.        this.id=id;
12.    }
13.
14.    Cliente(int id, Usuario usuario, String nome, ArrayList<Endereco> enderecos,
       ArrayList<Telefone> telefones){
15.        this.id=id;
16.        this.usuario=usuario;
17.        this.enderecos=enderecos;
```

```
18.       this.telefones=telefones;
19.    }
20.}
```

Telefone.java

```
1.public class Telefone {
2.    String descricao, fone;
3.    Telefone(){}
4.
5.    Telefone(String descricao, String fone){
6.       this.descricao=descricao;
7.       this.fone=fone;
8.    }
9.
10.}
```

A Figura 3 mostra, através do Diagrama de Entidade-Relacionamento, como as classes transformadas em tabelas interagem entre si, onde 1 usuário cadastra N clientes, 1 cliente possui N endereços e N telefones:

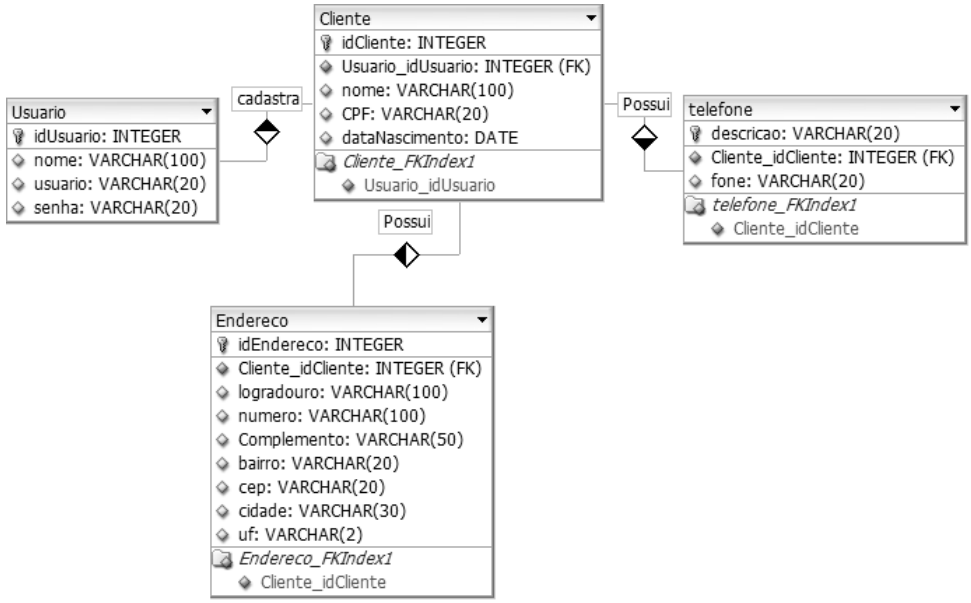

Figura 3. Diagrama de Entidade-Relacionamento

Para utilizar o banco, é feito uso dos objetos dos tipos: ObjectContainer e ObjectSet. O primeiro permite a criação, acesso e execução de operações na base (armazenar, consultar, atualizar, excluir) e o segundo é utilizado para recuperar os objetos após uma consulta em uma lista. Para exemplificar essas operações, foi criada a classe Banco.java:

Banco.java

```
1.import com.db4o.*;
2.public class Banco{
3.final static String DB4OFILENAME = System.getProperty("user.home") + "/banco.db4o";
4.static ObjectContainer db;
```

```
5.public void accessarDB4O() {
6.db = Db4oEmbedded.openFile(Db4oEmbedded.newConfiguration(), DB4OFILENAME);
7.}
8.public void fecharDB4O(){
9.db.close();
10.}
11.public void armazenar(Object o){
12.try {
13.db.store(o);
14.} finally {}
15.}
16.public ObjectSet buscar(Object o){
17.ObjectSet result = db.queryByExample(o);
18.return result;
19.}
20.public void excluir(Object o){
21.ObjectSet result = db.queryByExample(o);
22.Object found = result.next();
23.db.delete(found);
24.System.out.println("Deleted " + found);
25.}
26.public int getProximoId(){
27.accessarDB4O();
28.int t = db.queryByExample(new Object()).size();
29.fecharDB4O();
30.return t+1;
31.}
32.}
```

Nela, estão presentes os principais métodos para realização das operações no banco. Na linha 5, o método *void accessarDB4O()* cria o banco levando como parâmetro o nome dado a ele. Caso o banco já exista, ele somente realiza o acesso. O método *void armazenar* (linha 11), *void buscar* (linha 16) e *excluir* (linha 20) recebem como parâmetro um objeto, e como toda classe em java é filha da classe Object, qualquer dado pode ser utilizado como parâmetro para esses três métodos. Como o DB4O não possui o recurso nativo autoincrement (que realiza o incremento de um campo numérico e que está presente em diversos SGBDs), na linha 26 foi criado o método int getProximoId(), que facilitará a utilização dessa funcionalidade.

Afim de testar a aplicabilidade do DB4O, foi criada uma interface para que os objetos pudessem ser gravados na base. Ao pressionar o botão “Cadastrar”, é executado o seguinte trecho de código:

CadastroCliente.java

```
1.ArrayList <Telefone> telefones = new ArrayList(modeloTelefones.getRowCount());
2.
3.      ArrayList<Endereco> enderecos = new ArrayList(modeloEnderecos.getRowCount());
```

```
4.
5.      for(int i=0;i<modeloTelefones.getRowCount();i++){
6.          telefones.add(new Telefone(modeloTelefones.getValueAt(i,
    0).toString(),modeloTelefones.getValueAt(i, 1).toString()));
7.      }
8.
9.      for(int i=0;i<modeloEnderecos.getRowCount();i++){
10.         enderecos.add(new Endereco(modeloEnderecos.getValueAt(i,
    0).toString(),modeloEnderecos.getValueAt(i, 1).toString(),
11.                 modeloEnderecos.getValueAt(i, 2).toString(),modeloEnderecos.getValueAt(i,
    3).toString(),
12.                 modeloEnderecos.getValueAt(i, 4).toString(),modeloEnderecos.getValueAt(i,
    5).toString(),
13.                 modeloEnderecos.getValueAt(i, 6).toString()));
14.     }
15.     banco.accessarDB4O();
16.     int id = banco.getProximoId(new Cliente());
17.     Cliente c = new Cliente(id, usuario, nome.getText(), enderecos,telefones);
18.     banco.armazenar(c);
19.     banco.fecharDB4O();
```

São instanciadas (linhas 1 à 3) duas listas para armazenamento dos telefones e endereços do cliente (linhas 5 à 14). Da linha 15 à 19, são realizados, respectivamente, acesso ao banco de dados, aquisição do código do cliente (id), instanciação do objeto Cliente (levando como atributo os parâmetros: id, o usuário que realizou o cadastro, o nome do cliente, a lista de endereços e telefones), armazenamento do objeto e encerramento do acesso ao banco.

Caso seja necessária a recuperação das informações armazenadas, deve-se utilizar o método **buscar**. Para isso é preciso acessar o banco, declarar uma lista (List<Usuario>) para receber os resultados e chamar a função passando um objeto similar. Caso seja necessário encontrar um cliente específico, basta enviar algum dado dentro do construtor do objeto.

```
1.  db.accessarDB4O();
2.  List<Cliente> users =  db.buscar(Cliente.class); /*essa consulta retornará todos os
    objetos clientes cadastrados*/.
3.  db.fecharDB4O();
```

3. Considerações

O DB4O tem se mostrado eficaz em aplicações embarcadas onde há pouca memória disponível, uma vez que a aplicação e o banco são executados em um mesmo processo. Como demostrado no escopo deste artigo, comparado a outras bases de dados, é simples aprender a manusear o DB4O. Ele vem sendo utilizado em aplicações que necessitam escalonamento rápido e acesso simultâneo para o gerenciamento de informações em tempo real. Com ele não é necessário descaracterizar os objetos ao armazená-los, uma vez que os objetos são alocados integralmente no banco.

Todavia, um dos pontos fracos do banco de dados em questão é que o mesmo não possui mecanismos de segurança integrados; essa funcionalidade deverá ser implementada pelo programador, caso necessite. Por outro lado, ele dá suporte às chamadas *queries* nativas, que são funções de consultas com a sintaxe similar à da linguagem de programação utilizada.

4. Referências Bibliográficas

Codd, Edgar Frank. (1970) “A Relational Model of Data for Large Shared Data Banks”, IBM Research Laboratory, San Jose, CA.

Chen, Peter Pin-Shan. (1976) “The Entity-Relationship Model-Toward a Unified View of Data”, Massachusetts Institute of Technology, Cambridge, MA.

Jungthon, G. e Goulart, C. M.(2007) “Paradigmas de Programação”, em Faculdade de Informática de Taquara, Taquara, RS.

Mendes, Douglas Rocha. (2009) “Programação Java com Ênfase em Orientação a Objetos”, Novatec, 1ª edição.

Paterson, J., Edlich, S., Hörning, H. e Hörning, R. (2006) “The Definitive Guide to db4o”, http://www.apress.com, Julho.