Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

LUCAS SANTOS

BANCO DE DADOS ORIENTADO A OBJETOS: UMA BOA OPÇÃO

ASSIS

2012

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

BANCO DE DADOS ORIENTADO A OBJETOS: UMA BOA OPÇÃO

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal do Ensino Superior de Assis – IMESA e Fundação Educacional do Município de Assis – FEMA, como requisito para a obtenção do Certificado de Conclusão.

Orientador: Prof. Fernando César de Lima

Área de Concentração: Sistemas de Bancos de Dados

Assis

2012

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634
Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

FICHA CATALOGRÁFICA

SANTOS, Lucas

Banco de Dados Orientado a Objetos: Uma Boa Opção / Lucas Santos. Fundação Educacional do Município de Assis – FEMA – Assis, 2012.

35p.

Orientador: Prof. Fernando César de Lima

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1.DB4O 2. Orientação a objetos

CDD: 001.6
Biblioteca FEMA

Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

BANCO DE DADOS ORIENTADO A OBJETOS: UMA BOA OPÇÃO

LUCAS SANTOS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Instituto Municipal do Ensino Superior de Assis – IMESA e Fundação Educacional do Município de Assis – FEMA, como requisito para a obtenção do Certificado de Conclusão.

Orientador: Prof. Fernando César de Lima

Analisador: Prof. Dr. Alex Sandro Romeo de Souza Poletto

Assis

2012

AGRADECIMENTOS

Agradeço a minha família pelo apoio durante esses anos de curso.
Ao professor Fernando César de Lima pela Orientação e paciência.
A todos os professores do curso de Ciência da Computação que me ensinaram muito durante todo o curso.
A todos os meus amigos David, Rodolfo, Guilherme, Rafael, Rodrigo, Marcell, Valter, Gustavo, Marcos, Ana Luisa, Luís Felipe e muitos outros que conheci durante minha formação e que me apoiaram sempre que precisei.
Agradeço minha namorada Bruna que me ajudou e teve paciência para comigo durante a preparação desse trabalho.
E agradeço principalmente a Deus que sem ele nada disso seria possível.

RESUMO

A Orientação a Objetos (OO) já não é mais um novo conceito, e está sendo muito utilizada na área de programação, mas ainda não é muito bem vista na área de armazenamento de dados. Neste trabalho foi utilizado o banco de dados DB4O e teve como objetivo mostrar como o banco de dados OO é tão confiável quanto o relacional, fácil de usar, com alto desempenho e outros pontos positivos.

Palavras-chave: DB4O, Banco de dados, Orientação a Objetos.

ABSTRACT

The Object Orientation (OO) is no longer a new concept, and is being widely used in the programming area, but still not very well regarded in the area of data storage. In this study we used the database DB4O and aimed to show how the OO database is as reliable as the relational, easy to use, high performance and other positive points.

Keywords: DB4O, Database, Object Orientation.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS E ABREVIATURAS

BDOO	Banco de Dados Orientado ao Objeto
BDOR	Banco de Dados Objeto Relacional
DBA	Database Administrator
IDE	Integrated Development Invironment
NQ	Natives Queries
OME	Object Manager Enterprise Overview
OO	Orientação a Objeto
QbE	Querie by example
SGBD	Sistema Gerenciador de Banco de Dados
SGBDR	Sistema Gerenciador de Banco de Dados Relacional
SGBDOO	Sistema Gerenciador de Banco de Dados Orientado a Objetos
SGDBOR	Sistema Gerenciador de Banco de Dados Objeto Relacional
SQL	Structured Query Language

SUMÁRIO

1. INTRODUÇÃO/CONTEXTUALIZAÇÃO

A orientação a objetos hoje em dia é uma das formas de programação mais utilizada, porém a utilização dela no armazenamento de dados não, mantendo assim uma disparidade entre a programação e o armazenamento de dados. No entanto a necessidade de se armazenar dados complexos o Sistema Gerenciador de Banco de Dados Relacional (SGBDR) já não fornecia funcionalidades suficientes, então, essa necessidade fez com que a orientação a objetos chegasse aos Sistemas Gerenciadores de Banco de Dados (SGBD). O Sistema Gerenciador de Banco de Dados Orientado a Objetos (SGBDOO), também acabou com a disparidade que havia entre a aplicação e o Sistema Gerenciador de Banco de dados.

1.1 OBJETIVO

O objetivo deste trabalho foi a realização do estudo de um banco de dados orientado a objetos (DB4O), assim analisando suas vantagens no momento do desenvolvimento de um projeto, desde a parte do preparo do ambiente até programação.
O foco do trabalho foi em cima das consultas fornecidas pelo banco DB4O, realizando um sistema inteiramente orientado a objetos.

1.2 JUSTIFICATIVA

Banco de Dados Orientado a Objetos já está bem desenvolvido, porém ainda não é muito utilizado nos sistemas atuais, talvez por medo da mudança ou falta de conhecimento, é nesse sentido que esse trabalho poderá ajudar esclarecer algumas duvidas e mostrar como o SGBDOO também é confiável e preparado para esses sistemas.

1.3 MOTIVAÇÃO

Empresas que armazenam dados complexos necessitam de um banco orientado a objetos, já que o banco de dados relacional não é capaz de armazenar esses dados, logo esse tipo de banco é cada vez mais requisitado.
Esse trabalho servirá como material prático, pois o já existente para trabalhar com esse tipo de banco de dados não é muito encontrado.

1.4 ESTRUTURA DO TRABALHO

Este trabalho está organizado em cinco capítulos. Além deste capítulo, Introdução, existem os capítulos descritos a seguir:
Capítulo 2 - Banco de Dados - uma apresentação de algumas estruturas de banco de dados existentes hoje, assim como o Banco de dados Orientado a Objetos, que será o foco desse trabalho.
Capítulo 3 - Db4o - uma breve apresentação do banco de dados que será usado e suas ferramentas.
Capítulo 4 - Estudo de caso – o desenvolvimento do trabalho, com exemplos práticos e explicações.
Capítulo 5 – Conclusão – considerações finais sobre o trabalho desenvolvido.

2. SISTEMA DE GERENCIAMENTO DE BANCO DE DADOS

O conceito que motivou o surgimento de bancos de dados é quase tão antigo quantos os primeiros sistemas computacionais para aplicações não-numéricas, remontando ao final da década de 1950. Entretanto, o surgimento de sistemas gerenciadores de banco de dados sofreu influências também de outras áreas de aplicação, notadamente da área de defesa e de aplicações militares (Ricarte, 1998).

Um Sistema Gerenciador de Banco de Dados gerencia uma grande quantidade de dados, armazenados em um Banco de Dados. Tem como objetivo “fornecer uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente” (Silberschatz *et al.,* 2006).

2.1. MODELO RELACIONAL

O sistema de gerenciamento de banco de dados relacional (SGBDR) já esta muito bem consolidado no mercado. Foi o primeiro modelo de dados para aplicações comerciais, e atende muito bem para o que vem sendo utilizado. Com sua estrutura de tabelas/entidades, regras, chave primária, chave estrangeira, etc. (Figura 1). Durante muito tempo será difícil abandoná-lo, pelo seu legado, em que muitas empresas dependem de dados antigos já armazenados em bancos relacionais e seria muito custosa essa transição.

Seu objetivo é gerar esquemas de relações que permitam armazenar informações sem redundância desnecessária com desempenho, rapidez no acesso aos dados e consultas sofisticadas, mas um ponto negativo para o banco de dados relacional é sua falta de suporte para dados mais complexos.

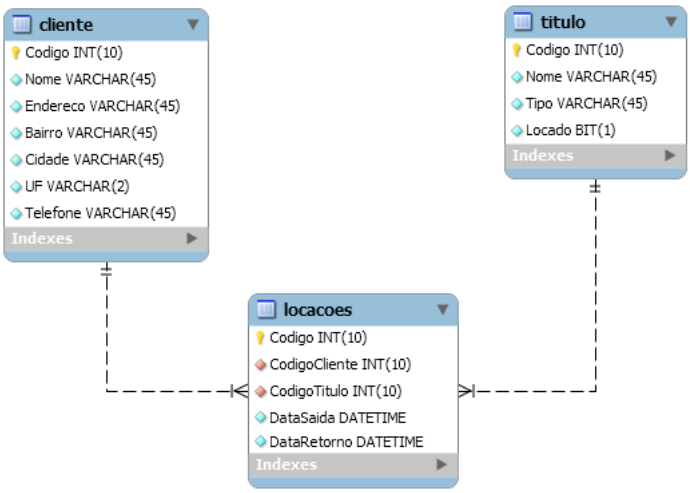

Figura 1. Modelo de Dados Relacional

2.2. MODELO ORIENTADO A OBJETOS

Mesmo com o banco de dados relacional já bem estruturado no mercado, algumas empresas sentiram que suas aplicações estavam sendo limitadas por restrições impostas, pois necessitavam de um armazenamento para dados complexo, como foi o caso da empresa Genethon laboratories, que em 1992 começou a desenvolver o primeiro protótipo, IDB, para o projeto “Genome View Project”. Esse fato ajudou o estudo para o desenvolvimento de um banco de dados orientado a objetos (BDOO), que possuísse segurança e formas de consultas avançadas. A segunda versão do banco lançada em 1994, chamou-se EyeDB, e vem sendo utilizado em vários projetos de bioinformática.

Com o BDOO é possível fazer o armazenamento de dados complexos e com isso a representação do mundo real é muito mais próxima. Além disso, a dificuldade de se mapear uma linguagem orientada a objetos para um banco relacional não existe, economizando tempo na produção de um software. Mais algumas diferenças do BDOO são:

- A eliminação de um campo código/id, já que não será mais necessário esse tipo de relação, pois o objeto realmente será gravado dentro de outro objeto.
- A eliminação das tabelas, chaves primarias e estrangeiras, o que deixa de fora o trabalho da criação do banco antes da programação.

Na parte de consulta, a orientação a objetos faz a diferença na questão de “caminhar” entre os objetos onde deixa a consulta bem mais simples do que no banco relacional.

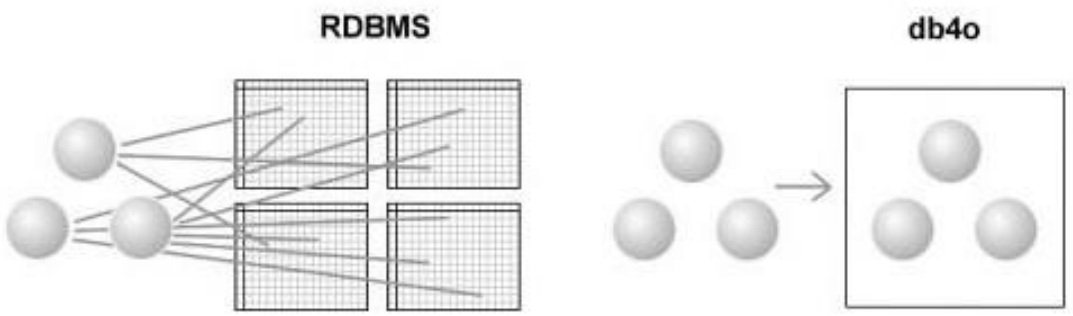

Figura 2. Salvando Objetos no Banco

Como visto na Figura 2, trabalhando com o banco de dados relacional, os objetos são divididos e interligados, se necessário, entre tabelas, o que implica fazer o mapeamento ou trabalhar com as consultas SQLs na programação. Já no caso do banco orientado a objetos isso não acontece, pois os objetos são salvos exatamente como estão, basta carregar um objeto, trabalhar com ele e salvá-lo.

2.3. MODELO OBJETO-RELACIONAL

Vendo que o banco de dados relacional (BDOR) tem um bom desempenho e que o banco de dados orientado a objetos tem a capacidade de armazenar dados complexos e trabalhar com herança, polimorfismo, etc., surge então o banco de dados objeto relacional, que tenta unir as vantagens dos dois bancos.

Esse banco estende o modelo relacional fornecendo um tipo de sistema mais rico, acrescentando estruturas a linguagens de consultas relacionais para tratar os tipos de dados acrescentados.

No banco de dados relacional também temos, por exemplo, o entity framework para C# ou hibernate para java que são exemplos de mapeamento objeto-relacional. Com esse mapeamento é possível se trabalhar com os objetos, mas guardá-los em tabelas, unindo assim o já conhecido modelo relacional com a programação orientada a objetos, porém ainda assim não será possível salvar dados complexos, como é feito no BDOO e no BDOR.

3. DB4O

Db4o é um banco de dados orientado a objetos open source, leve, seguro, compatível com .NET e JAVA. Ele é um .jar ou .dll adicionado ao projeto, logo, não precisa que um banco seja instalado separadamente quando o software for instalado, facilitando assim a implantação do mesmo.

Usando db4o o programador ganha tempo ao não precisar fazer todo o mapeamento de uma linguagem orientada a objetos para um banco relacional.

O acesso nativo SQL é rápido, porém, muito trabalhoso e com mapeamento objeto-relacional se perde desempenho, então com db4o há a união dos fatores “pouco trabalho” e “desempenho”, pois não terá a camada de mapeamento para fazer a conversão das linguagens, simplesmente serão gravados os dados por mais complexos que sejam. Além disso, apresenta custo reduzido no projeto, já que não será preciso de um DBA.

Algumas funcionalidades disponíveis no db4o são:

- Sessões – Início e fim.
- Arquivos de base de dados – Criar, Abrir, Fechar e Apagar.
- Transações – Commit e Rollback
- Objetos – Armazenar, recuperar, alterar (inclusive cascata), replicar, apagar (inclusive cascata).

Recursos chave	Benefícios chave
O banco-de-uma-linha-de-código Uma linha de código armazena qualquer objeto Modelo de Classes = esquema de objetos Processo de produção suave	**Corta 90% do custo** para desenvolver persistência **10% mais rápido para comercializar** sua aplicação
Embarcável Administração zero Versionamento automático de esquema 400 KB de tamanho	Roda até **44x mais rápido** que aplicações tradicionais Distribuível em grandes volumes sem administração local
Suporte a múltiplas plataformas Nativo para Java e .NET CPU embarcadas, dispositivos móveis, desktop e servidores Execução *cross-platform*	Construa Sistemas leves e realmente orientados a objeto Construa aplicações distribuídas e totalmente sincronizadas
Traz mais OO para o banco de dados Replicação orienta a objetos Queries nativas ObjectManager browser	Menos erros, melhor refatoramento e longevidade do software.

Figura 3. Recursos e Benefícios do Banco

3.1 CONSULTAS

Db4o trabalha com três tipos de consultas: NQ (Native Queries), QbE (Querie by Example) e S.O.D.A.

É o primeiro banco de dados OO a fornecer NQ que funciona como uma consulta ao banco de dados com a mesma semântica da linguagem que esta sendo utilizada em .NET pode ser utilizado com o LINQ da Microsoft. Essa consulta é checada em tempo de compilação, é 100% reformulável e fornece muita produtividade ao desenvolvedor. É recomendada para todo tipo de consulta, por ser precisa e de rápida escrita.

QbE é recomendado para consultas de poucas linhas de código. Fornece ao desenvolvedor, consultas de forma simples, recupera os dados por um objeto exemplo, porém há algumas restrições nesse tipo de consulta, como:

- Não executa consultas com expressões avançadas: AND, OR, NOT, etc.
- Não restringe valores como 0, strings vazias ou valores nulos.
- É necessário um construtor para criar objetos sem campos inicializados.

S.O.D.A é uma forma de consulta poderosa e de criação de consultas dinâmicas. Até o surgimento da NQ era a mais recomendada. É a API de nível mais baixo do DB4O, não é checada em tempo de compilação e é um pouco mais trabalhosa de se escrever. Sua utilização é recomendada em consultas que precisam de rapidez.

Em um projeto deve-se usar os três tipos de consulta, de acordo com a necessidade do programador. Esses três tipos de consulta serão apresentados em prática no próximo capítulo.

3.2 OBJECT MANAGER ENTERPRISE OVERVIEW

No Object Manager Enterprise Overview (OME) pode-se trabalhar diretamente com o banco criado, basta instalá-lo na IDE eclipse, que assim como o banco, é de fácil instalação, e aproveitar tudo que ele tem a oferecer. OME trabalha com a parte visual do banco de dados gerado, onde pode-se consultar todos os dados. Ele trabalha com quatro abas, que são:

- DB4O Browser – lista as classes do banco.

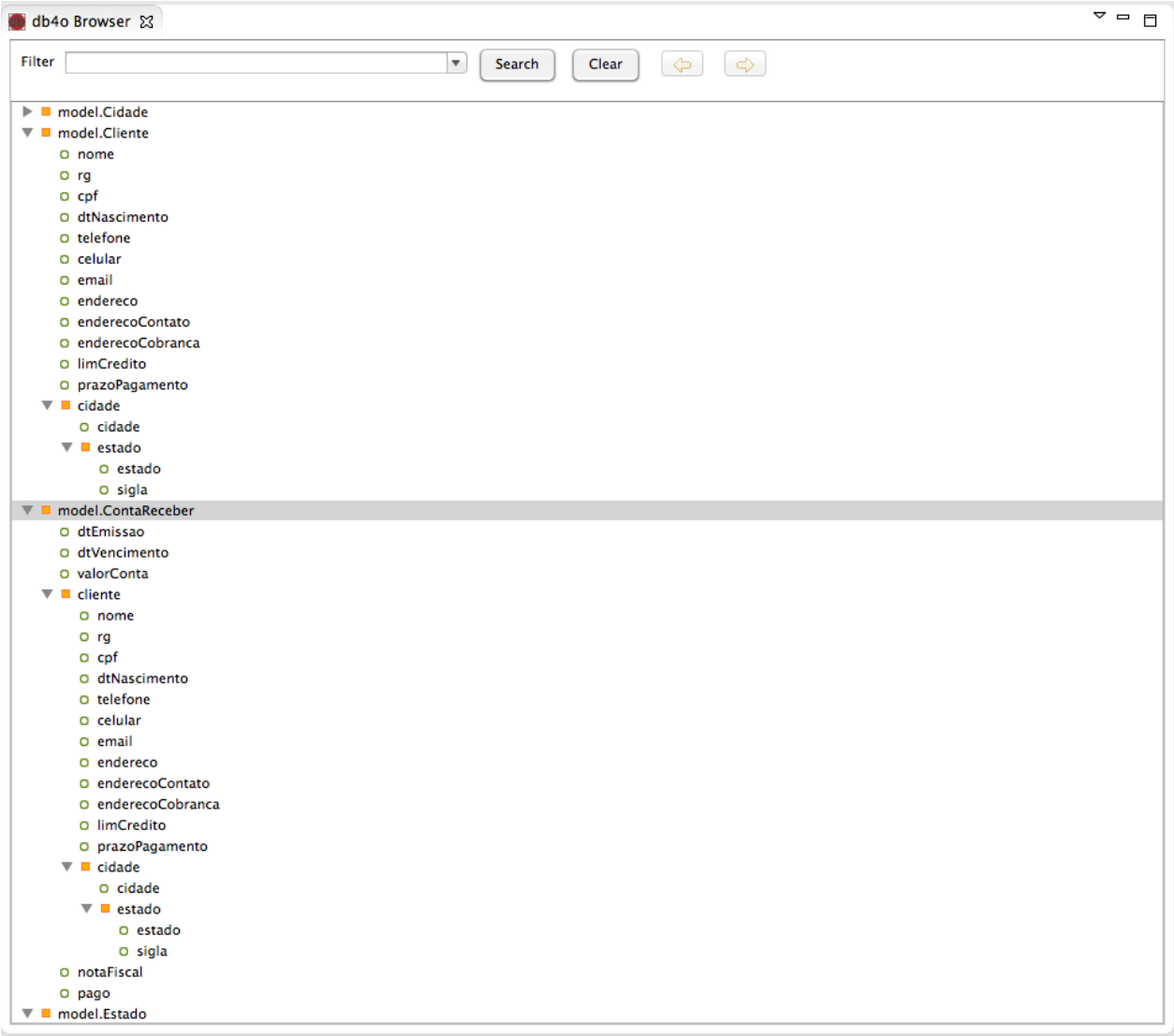

Figura 4. DB4O Browser

Nessa lista aparecerá todas as classes e seus relacionamentos que foram estabelecidos na programação. Clicando com o botão direito pode-se listar todo o conteúdo da classe na aba query results, que veremos logo a frente.

- Build Query – nela pode-se executar uma consulta especifica.

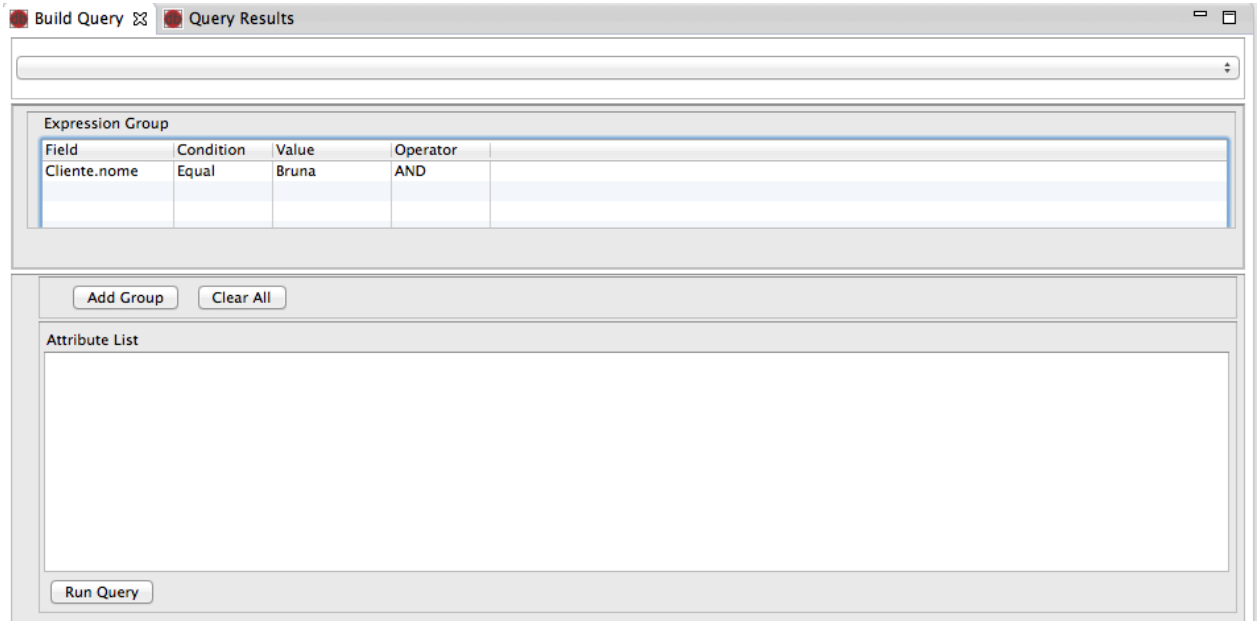

Figura 5. Build Query

No build query basta arrastar uma classe da aba DB4O Browser para Expression Group e especificar o tipo de consulta, com condição, valor a ser pesquisado e o operador, executando um ou mais grupos de consulta. O resultado dessa consulta será retornado na aba Query Results.

- Query Results – lista os resultados da pesquisa.

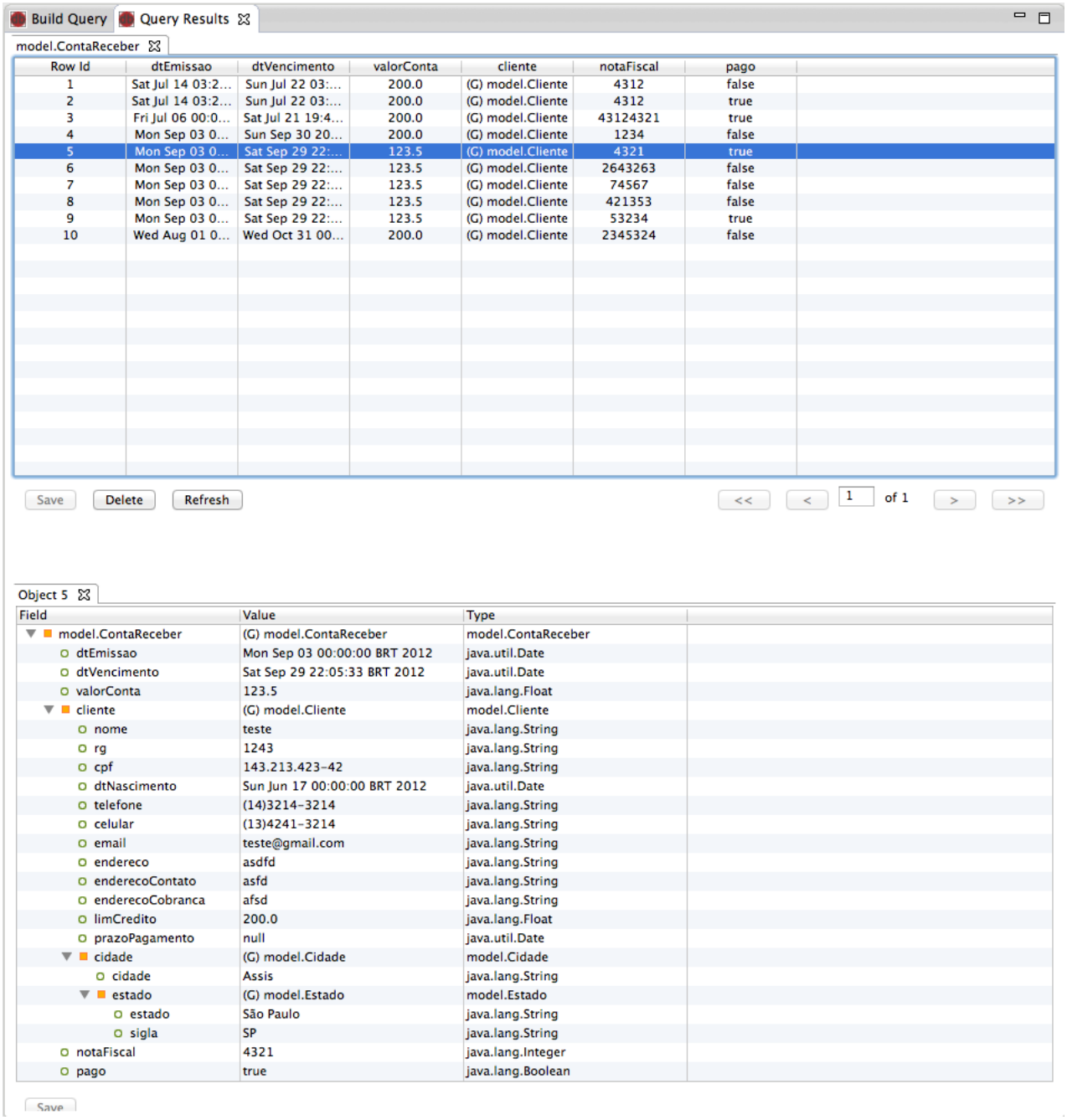

Figura 6. Query Results

Query Results é a mais interessante, nela pode-se visualizar, alterar o objeto e os objetos relacionado a ele, tudo visualmente, sem nenhuma linha de código. Para isso basta clicar em cima do objeto no qual se deseja visualizar os dados, que ele será mostrado logo abaixo. Assim como está na imagem é possível desmembrar o objeto e todos os seus relacionamentos para visualizar seu conteúdo e/ou alterá-los.

- Property Viewer – propriedades dos dados armazenados no banco.

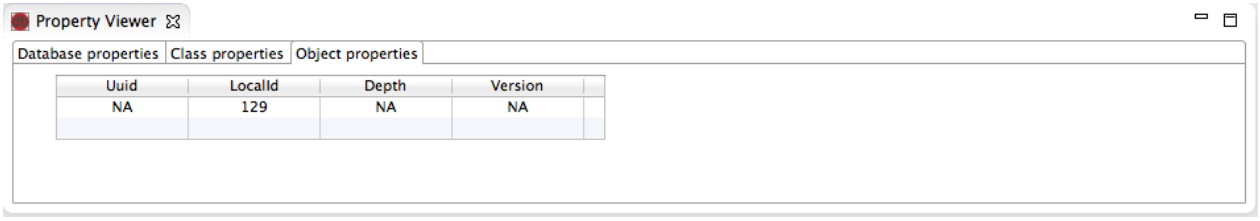

Figura 7. Property Viewer

No Property Viewer é possível visualizar as propriedades do banco, propriedades da classe e propriedades do objeto, podendo exibir assim:

- No banco – seu tamanho, número de classes e espaço livre.
- Na classe – os campos, o tipo de cada campo, se o campo é publico ou não e se o campo é indexado, podendo alterar a opção indexado para true ou false.
- No objeto – Uuid, LocalId, Depth e a Versão.

O OME então veio para facilitar a vida do programador, já que o próprio não necessitará fazer consultas pesadas e extensas para tratar relacionamentos. Também é uma grande vantagem para empresa em questão de custos, pois não teria a necessidade de se contratar um DBA.

4. ESTUDO DE CASO

O projeto desenvolvido nesse trabalho é simples, pois o objetivo é apenas utilizar os métodos fornecidos pelo DB4O, para se demonstrar como é produtivo desenvolver um software com o mesmo.

4.1 CASO DE USO

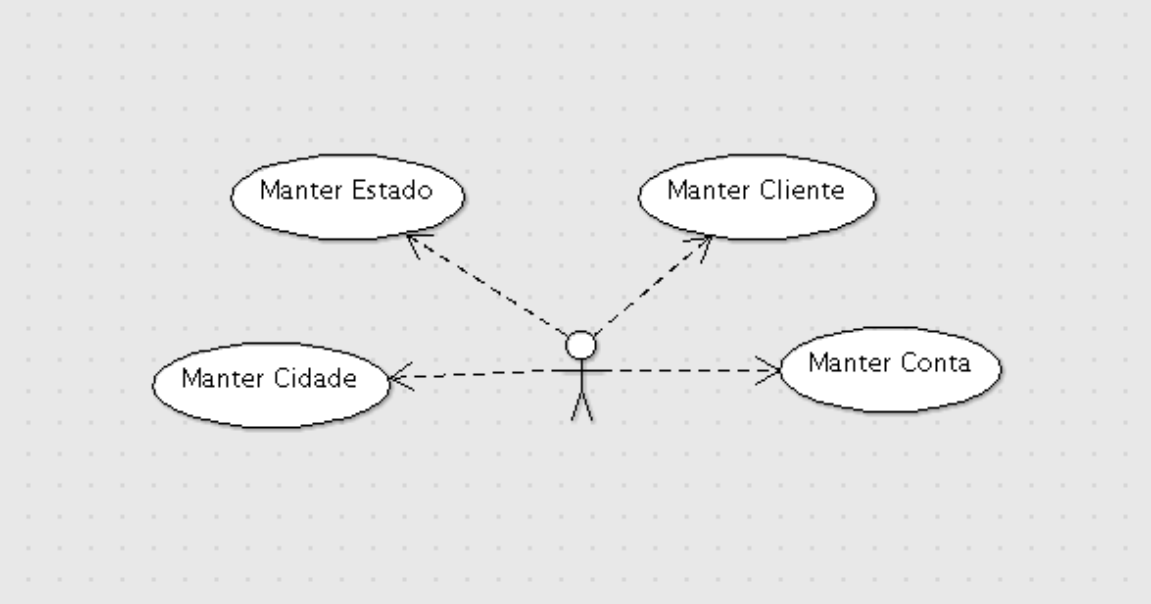

Figura 8. Caso de Uso

4.2 DIAGRAMA DE CLASSES

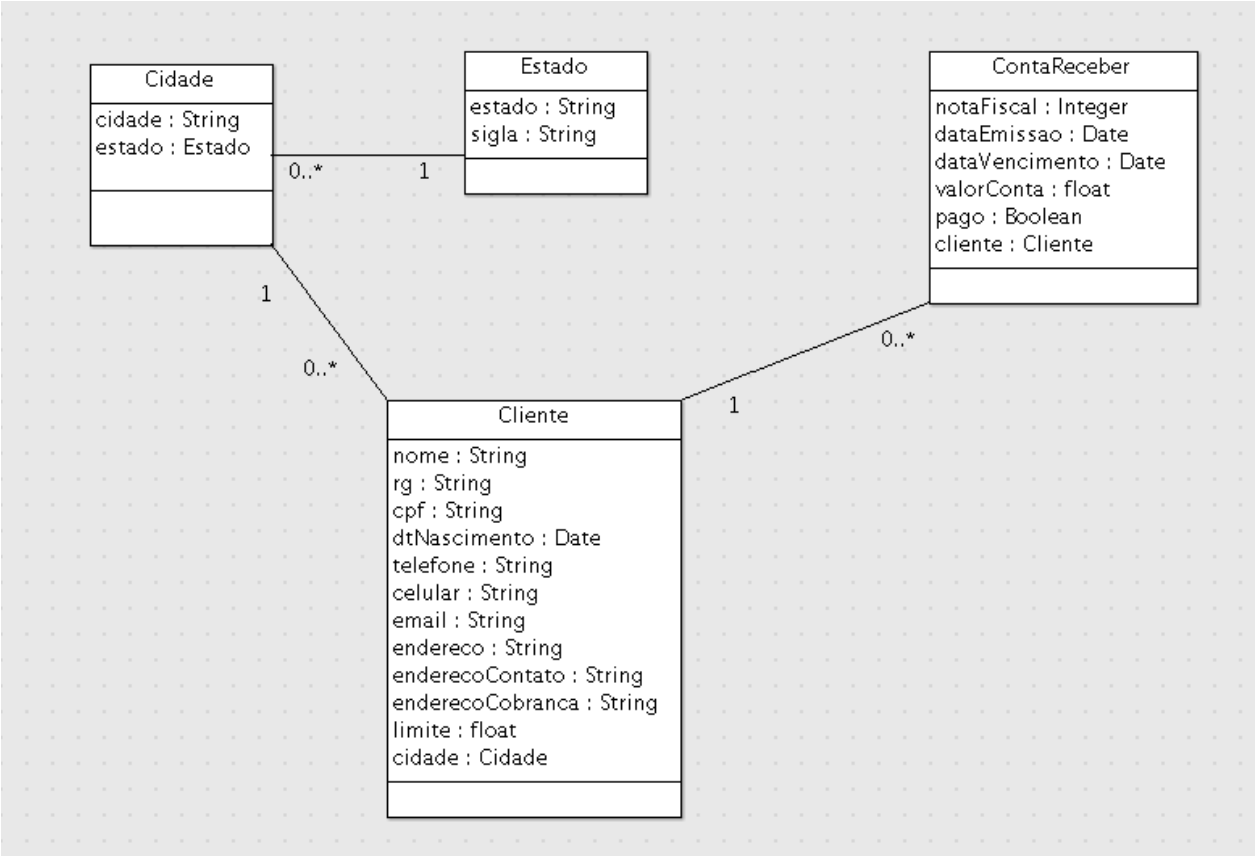

Figura 9. Diagrama de Classes

4.3 PREPARANDO O AMBIENTE DE DESENVOLVIMENTO

Na preparação do ambiente de desenvolvimento é onde mais se economiza tempo e trabalho na produção de um software utilizando o banco de dados DB4O. Logo de início se tem a facilidade de implantação do banco, já que basta adicionar o .jar ao java ou a .dll ao c# que já está instalado e pronto para ser utilizado.

No quesito trabalhar com objetos, com o banco relacional teria que ser construído todo o banco de dados, e na programação, o mapeamento, usando algum tipo de persistência de dados. A camada de persistência fica encarregada de converter os

objetos em atributos das tabelas do banco e de converter os atributos das tabelas do banco em objetos (Figura 10), tudo para que o programador tenha a sensação de estar trabalhando diretamente com os objetos ou se deixar de fora o mapeamento, usar SQLs para inserção no banco. Já no Banco de Dados Orientado a Objetos todo o trabalho de construção do banco e a parte de mapeamento seriam descartados, pois o banco realmente trabalhará diretamente com objetos (Figura 10).

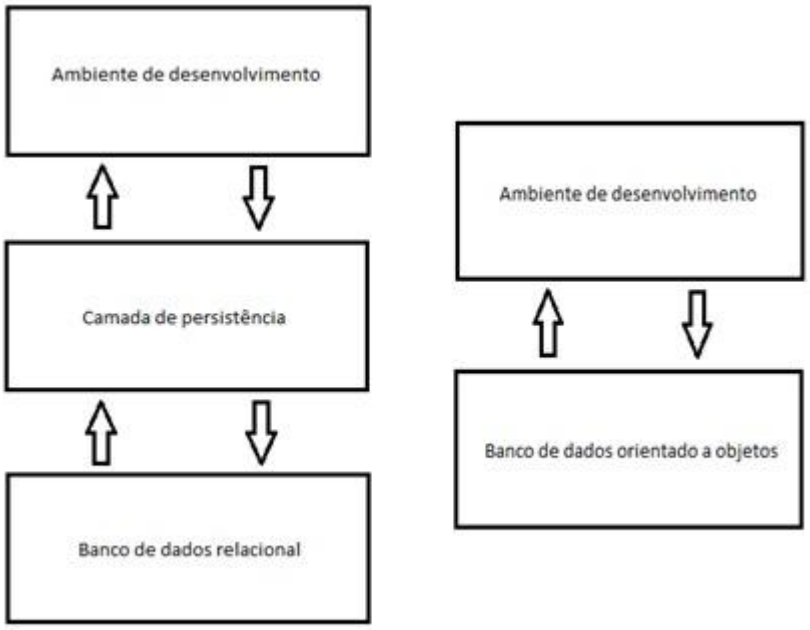

Figura 10. Gravando Objetos

4.4 CONEXÃO

Após adicionar o .jar a aplicação o próximo passo para se utilizar o banco é a conexão, que assim como a implantação do banco é muito simples. Como no exemplo abaixo:

```
public static ObjectContainer db;
        static{
                EmbeddedConfiguration config = Db4oEmbedded.newConfiguration();
                db = Db4oEmbedded.openFile(config, "../tcc1.3/bancoDB4O");
        }
```

Em uma classe destinada a conexão, cria-se um ObjectContainer chamado db. Em seguida cria-se um EmbeddedConfiguration chamado config, que provê vários métodos de configuração para solicitar algum comportamento especial, onde db receberá a conexão contendo como parâmetros o objeto config e o local com o nome do arquivo que será salvo.

4.5 PROGRAMAÇÃO

Criando a classe do objeto já percebe-se uma diferença entre o SGBDR e o SGBDOO, onde no primeiro é necessário uma variável de identificação que ajuda a identificá-lo no banco de dados, em um relacionamento com outro objeto. No SGBDOO isso não será necessário, pois os objetos serão gravados exatamente como estão.

- Classe Estado trabalhando com banco de dados relacional.

```
public class Estado {
        private Integer id;
        private String estado;
        private String sigla;
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getEstado() {
```

```
                return estado;
        }
        public void setEstado(String estado) {
                this.estado = estado;
        }
        public String getSigla() {
                return sigla;
        }
        public void setSigla(String sigla) {
                this.sigla = sigla;
        }
}
```

- Classe Estado trabalhando com DB4O

```
public class Estado {
        private String estado;
        private String sigla;
        public String getEstado() {
                return estado;
        }
        public void setEstado(String estado) {
                this.estado = estado;
        }
        public String getSigla() {
                return sigla;
        }
        public void setSigla(String sigla) {
                this.sigla = sigla;
        }
}
```

Um dos benefícios do SGBDOO é de se trabalhar com os objetos diretamente, gravá-los, deletá-los e recuperá-los sem muito trabalho. E como pode-se notar nos exemplos a seguir, em toda parte teremos ganho de produtividade:

- Gravando Objeto

```
private void cadastrarContaReceber(){
                ClienteDao clienteDao = new ClienteDao();
                ContaReceber contaReceber = new ContaReceber();
                ContaReceberDao contaReceberDao = new ContaReceberDao();
                try {
        contaReceber.setCliente(clienteDao.obterPorNomeSODA(ddlCliente.getSelected
Item().toString()));
        contaReceber.setNotaFiscal(Integer.parseInt(txtNotaFiscal.getText()));
                        contaReceber.setValorConta(Float.parseFloat(txtValor.getText()))
                        contaReceber.setDtEmissao(jDateChooser.getDate());
                        contaReceber.setDtVencimento(jDateChooser2.getDate());
                        contaReceber.setPago(cbPago.isSelected());
                        contaReceberDao.addContaReceber(contaReceber);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
public void addContaReceber(ContaReceber contaReceber)throws Exception{
      Banco.db.store(contaReceber);
      Banco.db.commit();
        }
```

Como observado no exemplo acima temos dois métodos. Um método da classe ContaReceber e outro da classe ContaReceberDao, onde cadastrarContaReceber recebe os dados do formulário e insere no objeto contaReceber e logo em seguida chama addContaReceber passando o objeto como parâmetro. Já no addContaReceber que é o método encarregado de salvar o objeto no banco com apenas duas linhas, uma para gravar (store) e outra pra efetuar o commit, se executa a tarefa.

- Recuperando Objetos

```
public List<ContaReceber> getAllSODA()throws Exception{
        Query query= Banco.db.query();
        query.constrain(ContaReceber.class);
        return query.execute();
}
public List<ContaReceber> getAllQBE()throws Exception{
        return Banco.db.queryByExample(ContaReceber.class);
}
```

Aqui temos dois tipos de consultas, ambas retornando todas as Contas a Receber cadastradas no banco por S.O.D.A. e QBE. Note como é fácil implementar consultas, especialmente na QBE que recupera os dados em apenas uma linha de código.

```
public List<Cliente> consulta(String busca)throws Exception{
                List<Cliente> listCliente = null;
                Query query= Banco.db.query();
                query.constrain(Cliente.class);
                query.descend("nome").constrain(busca).like().or(query.descend("email").c
        onstrain(busca).like());
                listCliente = query.execute();
                return listCliente;
        }
```

Nesse exemplo temos uma consulta, utilizando S.O.D.A., um pouco mais sofisticada, onde trará todos os clientes que tenham nome com a string busca ou e-mail que contenha a string busca. Também é muito simples, basta indicar a Classe que será usada na consulta, e em seguida informar as queries a serem executadas. O resultado da pesquisa resultará uma lista de objetos da classe utilizada.

```
contaReceberLista = Banco.db.query(new Predicate<ContaReceber>() {
        private static final long serialVersionUID = 1L;
        public boolean match(ContaReceber contaReceber) {
                boolean resultado =
contaReceber.getCliente().getNome().contains(txtCliente.getText()) &&
contaReceber.getPago().equals(false);
                return resultado;
                }
        });
```

Nesse caso usa-se a NQ, onde pode-se observar a utilização da própria linguagem para se efetuar as consultas. O que este tipo de consulta realiza é, caso o retorno do método match seja true o objeto será adicionado a lista contaReceberLista, ou seja, ele vai passar por todos os objetos da classe e retornar apenas aqueles que estarão de acordo com os requisitos da consulta.

```
public Cliente obterPorNomeSODA(String busca)throws Exception{
                ObjectSet<Cliente> clienteList = null;
                Query query= Banco.db.query();
                query.constrain(Cliente.class);
                query.descend("cpf").constrain(busca);
                clienteList = query.execute();
                if (clienteList.isEmpty()){
                        return null;
                }else{
                        return clienteList.iterator().next();
                }
```

Agora nesse tipo de consulta o objetivo é de retornar apenas um objeto em especifico. Então usando a consulta por S.O.D.A., cria-se um ObjectSet de objetos no qual se

espera que tenha apenas o objeto previsto ou nenhum. Em seguida, caso o ObjectSet contenha algum objeto, se executa os métodos .iterator().next(), que faz a busca pelo próximo objeto e o retorna.

- Atualizando Objeto

```
public void addCliente(Cliente cliente)throws Exception{
        Banco.db.store(cliente);
        Banco.db.commit();
          }
```

O mesmo método usado para salvar o objeto pela primeira vez é usado também para atualizá-lo. Basta recuperar o objeto a ser atualizado e passá-lo com as devidas alterações.

- Deletando Objeto

```
public void delCliente(Cliente cliente)throws Exception{
          Banco.db.delete(cliente);
          }
```

Como observado acima, pode-se dizer que deletar um objeto é uma das tarefas mais fáceis, basta passar o objeto que deverá ser deletado para o método delete e pronto, com apenas uma linha de código o objeto foi excluído.

5. CONCLUSÃO

Um dos desafios deste trabalho foi aprender a utilizar os métodos do banco e se acostumar a não trabalhar com id e tabelas de banco, que é um método tão natural hoje em dia.

A intenção desse trabalho de conclusão de curso é demonstrar como um banco de dados orientado a objeto é produtivo, e incentivar o uso do mesmo. Foi levado em consideração a implantação do banco até o uso na programação. Com isso temos uma grande vantagem ao se desenvolver com esse banco (DB4O), minimizando o trabalho em ambos os casos.

O estudo desse banco contribui diretamente como material para alunos que pretendam utilizar um BDOO para futuros projetos.

Pode-se concluir que o banco orientado a objetos veio para aumentar e facilitar a produtividade do desenvolvimento de software e que o tempo de estudo do banco será diretamente compensado no desenvolvimento dos projetos. É claro que uma empresa dificilmente passará um BDR para um BDOO de um software já implantado e funcionando corretamente, seria muito custoso, mas poderia aproveitar essa tecnologia para novos projetos.

REFERÊNCIAS BIBLIOGRÁFICAS

BOSCARIOLI, Clodis *et al.* **Uma reflexão sobre Banco de Dados Orientados a Objetos**. 12p. Universidade Estadual do Oeste do Paraná (UNIOESTE).

CERÍCOLA, V. O. **Banco de Dados Relacional e Distribuído**. 448p. São Paulo: Makron Books, 1995.

DB4Objects. **DB4O api**. Disponível em: <http://community.versant.com/documentation/reference/db4o-8.0/java/api/>. Acesso em: 23 março 2012.

DB4Objects. **DB4O tutorial**. Disponível em: <http://community.versant.com/documentation/reference/db4o-8.0/java/tutorial/>. Acesso em: 23 março 2012.

DB4O **Product Information**. Disponível em: <http://www.db4o.com/about/productinformation/db4o%20Product%20Information%20V8.0.pdf>. Acesso em: 13 Julho 2012.

EyeDB. **Eyedb history**. Disponível em: <http://www.eyedb.org/history/>. Acesso em: 21 Abril 2012.

GALANTE, A. C.; MOREIRA, E. L. R.; BRANDÃO, F. C. **Banco de dados orientado a objetos: uma realidade**. 15p. Faculdade Salesiana Maria Auxiliadora.

KHOSHAFIAN, S. **Banco de Dados Orientado a Objeto**. 353p. Rio de Janeiro: Infobook, 1994.

NEVES, N. A.; ROCHA, G. A.; SEGUNDO, A. O. **Banco de Dados Objeto-Relacional (BDOR)**. 14p. Universidade Federal da Bahia (UFBA).

PATERSON, J. *et al.* **The Definitive Guide to db4o.** 2006.

PINHEIRO, D. R. S. *et al.* **Comparativo entre Banco de Dados Orientado a Objetos (BDOO) e Bancos de Dados Objeto Relacional (BDOR)**. 8p.

RICARTE, I. L. M. **Sistemas de Bancos de Dados Orientados a Objetos**. 1998. 41p. Universidade Estadual de Campinas.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados – 5ª edição**. 805p. Rio de Janeiro: Elsevier, 2006.