Banco de Dados Orientado a Objetos

Jerônimo Teles, Paulo César Gonçalves, Pedro Prado, Rutemberg Araújo

Ciência da Computação – Universidade Federal da Bahia (UFBA)
Salvador – BA – Brasil

{jeronimoteles,pauloc062, rutemberg.aj}@gmail.com, kbca_dm@hotmail.com

Resumo. Este artigo descreve como funciona um banco orientado à objetos (ODBMS), explica como é a linguagem de manipulação e definição dos dados, discute as principais características e compara com o modelo relacional. Além disso, há um trabalho prático implementado em OrientDB para exemplificar o funcionamento de um ODBMS.

1. Introdução

Um banco de dados orientado a objetos (BDOO) é um gerenciador de grandes volumes de informação que tem a estrutura de armazenamento de dados baseado no paradigma de orientação a objetos (OO) [Silberschatz et al. 1999]. A sua criação ocorreu quando houve uma necessidade de se trabalhar com aplicações complexas, implementadas em linguagens OO e com estrutura complexa de armazenamento de dados que possuísse algumas capacidades como continuidade, simultaneidade e transações dos seus ambientes.

Com os BDOO foi possível integrar o banco de dados mais facilmente às aplicações orientadas a objeto, pois a estrutura dos dados ficou semelhante às classes implementadas no software. Além disso, os conceitos de objetos, métodos, herança e polimorfismo, que são essenciais para linguagens OO, também fazem parte das características deste tipo de banco de dados [Ricarte 1998].

2. Características e Comparação com Modelo Relacional

Em sistemas computacionais, os bancos de dados orientados a objetos podem atuar como uma alternativa aos bancos de dados relacionais tradicionais para preservação de informações. A principal diferença está na forma da representação da informação que vai ser preservada, como também, recuperada. Ou seja, na interface que oferece ao desenvolvedor de sistemas. Enquanto os relacionais fazem uso de tabelas, os que implementam a tecnologia orientada a objetos valem-se da própria concepção e definição dos objetos. Ambas as maneiras, apesar de distintas, são válidas para representação e modelagem de situações do mundo real. No entanto, há de se considerar o impacto dessa divergência de modelagem nos diversos tipos de sistemas que necessitam preservar dados e informações em média e pequena escala.

A tecnologia de programação orientada a objetos está fundamentada no seguinte tripé: encapsulamento, herança e polimorfismo. No entanto, nesse contexto, uma das mais importantes características desse paradigma é a modularização como essência. Pode-se afirmar que a principal vantagem da modularização, dentre as demais, é que alterações podem ser elaboradas sem que isso afete inadvertidamente e inapropriadamente outras partes do programa.

É claro que os objetos serão as informações que serão preservadas neste cenário em questão. Há, contudo, um questionamento sempre válido neste momento: Qual é a tecnologia mais adequada para a preservação dos dados desses tipos específicos de informações?

O modelo relacional é simples e fundamentalmente diferente do modelo orientado a objetos. No modelo de tecnologia relacional, os repositórios não foram criados visando a persistência dos dados na forma de objetos, mesmo porque a informação, nos dias de hoje, não é constituída de informações facilmente tratáveis. Atualmente, a informação é apresentada em diferentes formatos, quais sejam: vídeo, gráfico, áudio e fotos que são considerados tipos complexos de dados, os quais os SGBDs relacionais não são próprios para suportar e gerenciar. Ademais, a criação de uma interface satisfatória para armazenar objetos em tabelas também um outro obstáculo encontrado nos repositórios que implementam o modelo relacional de dados.

Em projetos de SGBD relacionais, esta representação dos dados é feita em uma divisão das atividades em dois modelos: o modelo do banco de dados e o modelo da aplicação. Por outro lado, com um SGBD orientado a objeto, o programador pode manter a consistência do ambiente de desenvolvimento ao integrar um banco de dados a um paradigma de linguagem de programação e depois apresentar o sistema em um único modelo de projeto.

O modelo mais adequado para superar estas dificuldades é o de sistemas de banco de dados orientado a objetos, porque persistir ou armazenar dados como objetos em um banco de dados relacional ou até mesmo objeto-relacional é uma árdua e não trivial tarefa.

Os modelos acima mencionados são desprovidos de mecanismos necessários para poder representar características básicas e essenciais da programação orientada a objeto, mencionada anteriormente. Nesse contexto, um repositório com a tecnologia relacional não configura qualquer vantagem e por isso a opção por um banco de dados orientado a objetos é mais adequada para essa situação.

Atualmente, as linguagens de programação seguem uma tendência quase que unânime na utilização de orientação a objeto, fazendo com que os sistemas de gerenciamento de banco de dados orientado a objetos, SGBDOO, sejam ideais para os programadores, porque desta maneira eles podem desenvolver, armazenar, replicar e modificar produtos usando a modelagem por objetos desde o banco de dados até a aplicação final.

Os sistemas de banco de dados orientado a objetos passaram por grandes transformações desde seu advento até os dias atuais, atingindo maturidade e performances satisfatórias. Hoje em dia, políticas e novas estratégias de persistência e recuperação de dados possibilitam que esses repositórios tenham um tempo de resposta de consulta geralmente menor em relação às chamadas subseqüentes de um banco de dados relacional, próprias para construir e desfazer objetos, garantindo, assim, maior desempenho.

3. ODMG

O ODMG (Object Database Management Group) é um padrão que surgiu para poder se implementar bancos de dados orientados a objetos. Com as especificações

semelhantes, desenvolvedores de BDOO poderiam manter uma portabilidade entre seus sistemas.

O padrão ODMG especifica dois tipos de produtos: os sistemas de gerenciamento de dados, que armazenam os objetos diretamente no BD, e os mapeamentos de objetos para banco de dados, que convertem os objetos para qualquer outro tipo de BD, como o relacional, por exemplo. Outros fatores importantes deste padrão são: o modelo de objeto, que define estruturas básicas comuns; a linguagem de definição de objetos (ODL), usada para especificar classes, interfaces e outras estruturas necessárias; e a linguagem de consulta de objetos (OQL), utilizada para realizar manipulações sobre os dados no banco [Cattell et al. 2000].

3.1. Modelo de objeto

O modelo de objetos descreve a semântica que determina as características de objetos, os relacionamento inter-objetos e sua forma de identificação. Este modelo define as seguintes estruturas:

- As primitivas básicas de modelagem são: objetos, que possuem um identificador único; e literais ou valores, que não possuem identificadores;
- Objetos e literais podem ser caracterizado por tipos, ou seja, um conjunto comum de estados (propriedades) e comportamentos (operações). Um objeto às vezes é referenciado como uma instância de um tipo;
- O comportamento de um objeto é um conjunto de operações que podem ser executados sobre ou pelo objeto. As operações devem ter parâmetros de entrada e saída de um tipo específico;
- O estado de um objeto é definido pelas propriedades atribuídas a ele, que podem permanecer ou mudar com o tempo. As propriedades podem ser atributos ou relações entre um ou mais objetos;
- O banco de dados orientado a objetos guarda um conjunto de objetos que pode ser compartilhado por múltiplos softwares e usuários.

O tipo tem dois aspectos em sua definição. Uma é a especificação externa e a outra é uma ou mais implementações. A especificação define as características externas visíveis aos usuários, como as propriedades e operações do objeto que podem ser acessadas. Em contrapartida, as implementações definem o funcionamento interno do tipo.

A implementação depende da linguagem escolhida pelo programador. Esta parte é o código dos algoritmos das operações e propriedades, fase é conhecida como *language binding*.

A especificação externa do modelo do objeto consiste de três partes:

- Interface: especificação que define o comportamento abstrato de um tipo de objeto;
- Literal: especificação que define as propriedades abstratas de um tipo de objeto.

- Classe: especificação que define o comportamento abstrato e o estado (propriedades) abstrato de um tipo de objeto;

Com as estruturas citadas acima (interface, classe e literal), pode-se aplicar alguns conceitos de OO, como a herança:

```
class Pessoa {
        attribute string name;
};
class EmpregadoPessoa extends Pessoa : Empregado {
        attribute date dtInicioContrato;
};
class ClientePessoa extends Pessoa : Cliente {
        attribute string cartaoDeCredito;
};
```

O exemplo acima usando o padrão ODMG mostra que Cliente e Empregado herdam de Pessoa visto que ambos possuem o mesmo atributo nome.

3.2. Objetos

Abaixo são apresentados os aspectos dos objetos:

- Criação: os objetos são criados através da chamada de operações de uma interface. Todos os objetos inerentes a um tipo devem seguir a uma interface comum, que contém operações de copia, deleção, controle de concorrência e teste de unidade;
- Identificação: os objetos possuem identificadores que os distinguem uns dos outros em um domínio de armazenamento. O atributo de um objeto que o identifica unicamente é chamado de identificador de objeto, e é levado por todo seu ciclo de vida. Literais não possuem identificadores, portanto não podem se comportar como objetos;
- Nomes: para ser considerado portador de um identificador de objeto, o próprio objeto deve ter um ou mais nomes que sejam significantes e únicos para o programador/usuário. Dessa forma poderão ser feitas referências ao objeto com precisão.
- Tempo de vida: o tempo de vida de um objeto determina como a alocação de memória e armazenamento vão ser gerenciados no momento em que ele é criado. Pode ser transiente ou persistente. Quando é o tempo de vida é transiente, o objeto fica alocado na memória durante a execução e é descartado assim que o procedimento que o utiliza é finalizado. Por outro lado, se o tempo de vida for persistente, o objeto pode ser manipulado durante a execução sem que seja perdido posteriormente, pois é feita uma representação armazenada

do mesmo (banco de dados). Um mesmo objeto pode ter instâncias transientes e persistentes ao mesmo tempo.

- Estrutura: A estrutura do objeto pode ser atômica, ou seja, constituída por um único item definido pelo usuário, ou ser não-atômica. Para estruturas não-atômicas existem várias modalidades, como coleções, sacolas, vetores, etc.

Muitas das características dos objetos citadas acima também são atribuídas aos literais. Basta não levar em consideração as regras que se referem ao identificador de objeto.

3.3. Linguagem de definição de dados (ODL)

A Object Definition Language (ODL) é uma linguagem usada para definir objetos e literais seguindo uma linha comum aos BDOO's e assim facilitar a portabilidade de banco de dados. Esta linguagem não é completa para programação, serve apenas para especificar dados, mas atende bem à suas tarefas e é facilmente extensível.

Segue abaixo uma especificação simplificada da gramática do tipo de objeto (interface e classe) [Cattell et al. 2000]:

```
<interface>              ::=  <interface_dcl>
                              | <interface_forward_dcl>
<interface_dcl>          ::=  <interface_header>
                              { [ <interface_body> ] }
<interface_forward_dcl>  ::=  interface <identifier>
<interface_header>       ::=  interface <identifier>
                              [ <inheritance_spec> ]
<class>                  ::=  <class_dcl> | <class_forward_dcl>
<class_dcl>              ::=  <class_header> { <interface_body> }
<class_forward_dcl>      ::=  class <identifier>
<class_header>           ::=  class <identifier>
                              [ extends <scopedName> ]
                              [ <inheritance_spec> ]
                              [ <type_property_list> ]
```

Uma instância de propriedades corresponde aos atributos e relacionamentos de uma instância de um dado objeto. Segue abaixo a gramática da instância de propriedades:

```
<interface_body>  ::=  <export> | <export> <interface_body>
<export>          ::=  <type_dcl> ;
                       | <const_dcl> ;
                       | <except_dcl> ;
                       | <attr_dcl> ;
                       | <rel_dcl> ;
                       | <op_dcl> ;
```

Atributos são os dados propriamente ditos de um objeto. O seguinte código é a gramática para definição em ODMG:

```
<attr_dcl>          ::= [ readonly ] attribute
                        <domain_type> <attribute_name>
                        [ <fixed_array_size>]
<attribute_name>    ::= <identifier>
<domain_type>       ::= <simple_type_spec>
                        | <struct_type>
                        | <enum_type>
```

O relacionamento (relationship) é a forma que um objeto tem de se ligar com ele mesmo ou outros objetos. Como é visto abaixo, a definição do caminho da relação na gramática também apresenta o caminho inverso.

```
<rel_dcl>                ::= relationship
                             <target_of_path> <identifier>
                             inverse <inverse_traversal_path>
<target_of_path>         ::= <identifier>
                             | <coll_spec> < <identifier> >
<inverse_traversal_path> ::= <identifier> :: <identifier>
```

A especificação da gramática de uma operação é a que segue:

```
<op_dcl>            ::= [ <op_attribute>] <op_type_spec>
                        <identifier> <parameter_dcls>
                        [ <raises_expr> ] [ <context_expr> ]
<op_attribute>      ::= oneway
<op_type_spec>      ::= <simple_type_spec>
                        | void
<parameter_dcls>    ::= ( [ <param_dcl_list> ] )
<param_dcl_list>    ::= <param_dcl>
                        | <param_dcl> , <param_dcl_list>
<param_dcl>         ::= <param_attribute> <simple_type_spec>
                        <declarator>
<param_attribute>   ::= in | out | inout
<raises_expr>       ::= raises ( <scoped_name_list>)
<context_expr>      ::= context ( <string_literal_list> )
<scoped_name_list>  ::= <scoped_name>
                        | <scoped_name> , <scoped_name_list>
<string_literal_list> ::= <string_literal>
```

| <string_literal> **,** <string_literal_list>

3.4. Linguagem de manipulação de dados (OQL)

A Object Query Language (OQL) é a linguagem usada para manipular os dados do banco de dados, executando consultas. Seu design é baseado nas hipóteses e princípios a seguir.

- A OQL é muito semelhante à SQL-92. As extensões da linguagem tratam noções de orientação a objeto, como identidade de objeto, objeto complexo, expressão de caminho, polimorfismo e operações de instanciação;
- A OQL fornece uma primitiva de alto nível para tratar estruturas de coleções como listas e arrays com eficiência;
- A OQL é uma linguagem funcional onde operadores podem ser criados livremente, contanto que respeitem as regras do tipo de sistema;
- A OQL não é computacionalmente completa, é apenas usada para consulta;
- A OQL pode chamar operações;
- A OQL não pode chamar explicitamente operações de atualização, mas pode fazer a atualização indiretamente através de operações definidas no objeto.

Esta linguagem de manipulação faz as pesquisas a partir dos nomes dos tipos de objeto, que podem ser atômicos, estruturas, coleções e literais. Isto pode ser observado nos dois exemplos abaixo:

select distinct x.idade

from Pessoa x

where x.nome = “Joao”

select distinct struct(a: x.idade, s: x.sexo)

from Pessoa x

where x.nome = “Joao”

No primeiro exemplo pode ser visto que foi feita a busca da idade de todas as pessoas que tenham o nome João. Já no segundo caso, é retornada uma coleção de idade e sexo das pessoas também chamadas João.

A parte da estrutura também pode ser omitida. Por exemplo, para selecionar todas as pessoas, pode-se ignorar o *select* e *from* apenas declarando o termo *Pessoas* na consulta.

Para criar um objeto, deve-se chamar a operação construtora com mesmo nome do tipo passando todos os valores dos atributos. Para dar um valor precisa

informar o nome do atributo e o valor recebido. Caso ele não seja chamado no construtor, será preenchido com dados padrões da implementação. No exemplo abaixo pode ser visto como ocorre a inserção:

Pessoa(nome=”Joao”, idade=13, sexo=”masculino”)

Na OQL também é permitido fazer a navegação entre os relacionamentos das consultas de forma simples. Com o operador ‘.’ ou ‘->’ pode-se acessar diretamente objetos sem fazer *join*, que é bastante presente no banco de dados relacional.

select x.endereco.cidade.nome from Pessoa x

No exemplo acima pode ser visto o acesso ao atributo nome do objeto cidade acessado através do objeto endereço, que por sua vez foi acessado pelo objeto pessoa (x).

4. Exemplo

Nesta seção é tratada a parte prática do trabalho da disciplina Banco de Dados, que possui como tarefa criar uma aplicação com BDOO baseada no tema “Copa do Mundo 2014”. Primeiramente, será apresentada a especificação do software, em seguida uma breve descrição do banco de dados utilizado e depois será apresentado o código utilizando o OrientDB.

4.1. Especificação

O aplicativo servirá para administrar um conjunto de lojas no exterior, que venderão pacotes promocionais para a copa do mundo de 2014 no Brasil. Estes pacotes contêm os vôos internacionais de ida e volta marcados e um conjunto de reservas para os jogos.

Cada reserva contém um ingresso para o jogo selecionado e um conjunto de diárias na cidade do mesmo. Vale ressaltar que os pacotes dão direito a 20 diárias em qualquer hotel cadastrado na rede de lojas. Além disso, uma reserva pode alocar um vôo nacional para o cliente chegar ao local do jogo caso seja necessário, caso contrário fica por responsabilidade da pessoa o seu meio de transporte.

Um cliente pode comprar um ou mais pacotes promocionais e pode escolher entre as categorias de pacote Premium, Plus e Standard. O pacote Premium é o mais caro, nele o cliente terá direito a reservas em hotéis cinco estrelas e vôos internacionais de ida e volta na primeira classe. O pacote Plus também tem direito a passagens de primeira classe, porém os hotéis que podem ser reservados vão até no máximo quatro estrelas. Por fim, o pacote Standard dará direito a passagens de ida e volta comerciais e hotéis mais simples ou pousadas como hospedagem.

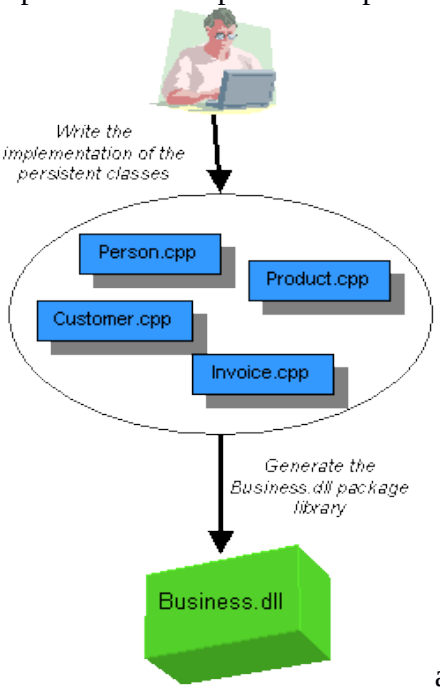
Para efetuar qualquer compra, o cliente deve ter um cadastro na loja com o seu Social Security Number, nome, e-mail, telefone, telefone de emergência, data de nascimento, rua, CEP, bairro e número da moradia.

O gerente de cada loja deverá ter os mesmos dados presentes no cliente, além disso precisa ter também o número do contrato para verificações administrativas.

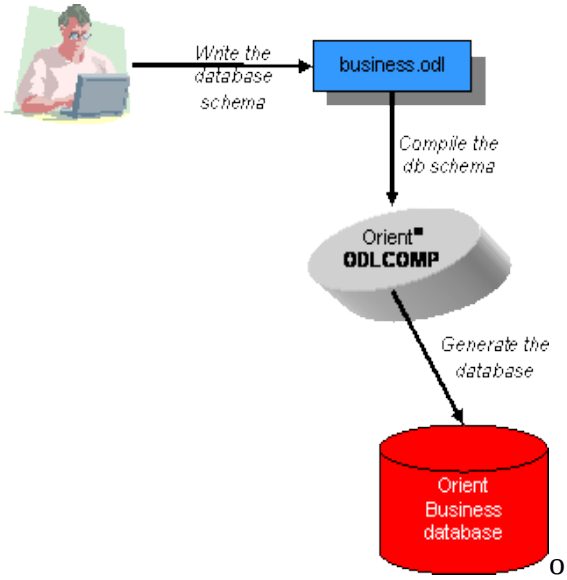
4.2. OrientDB

OrientDB é um banco de dados orientado a objetos que implementa parcialmente o ODMG 3.0, tem suporte a Java e C++ e é uma ferramenta de código-fonte aberto. Apesar de ser um banco de dados com certa maturidade, a documentação não é devidamente adequada, o que acaba por dificultar o trabalho com o mesmo.

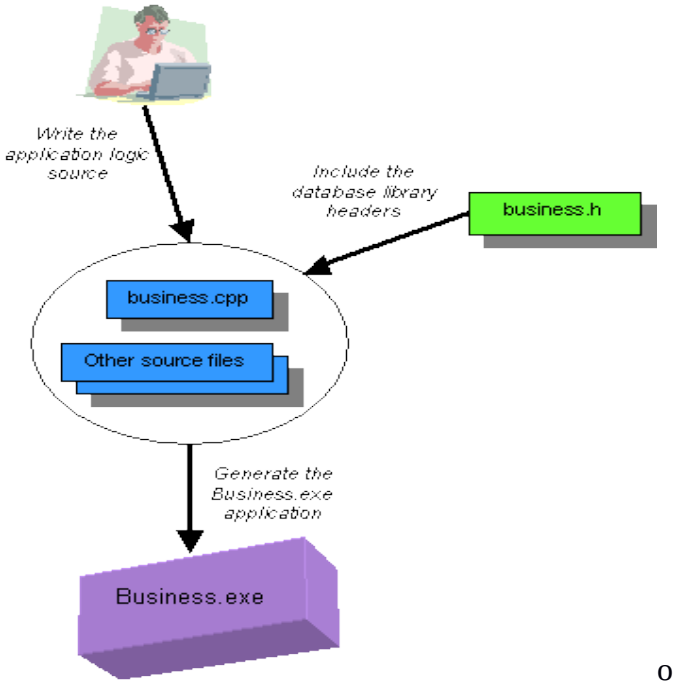
Para implementar um banco de dados no OrientDB primeiramente é necessário criar um arquivo ODL com a linguagem parcial que o banco fornece. Depois o próprio programa já tem um aplicativo que gera o código-fonte em uma linguagem OO (C++ ou Java) das classes criadas na ODL. Com isso, é necessário que o próprio programador crie uma DLL de suporte para o banco de dados, ou seja, exige que o programador integre o sistema do banco de dados aos arquivos gerados. Por fim a DLL pode ser usada para criar aplicativos que interajam com o OrientDB. As imagens



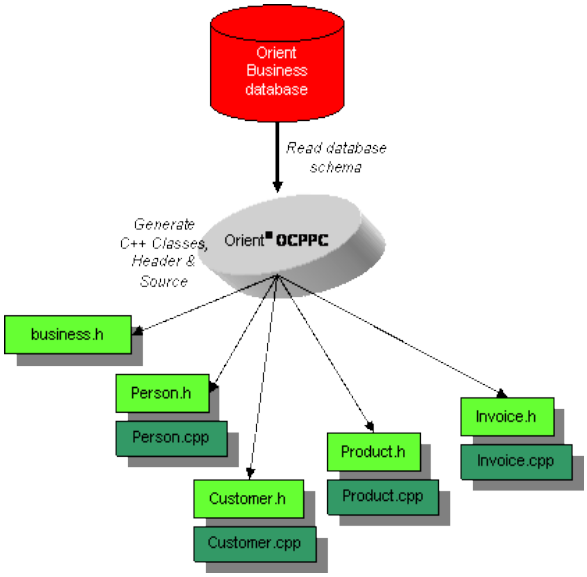
ab aixo demonstram

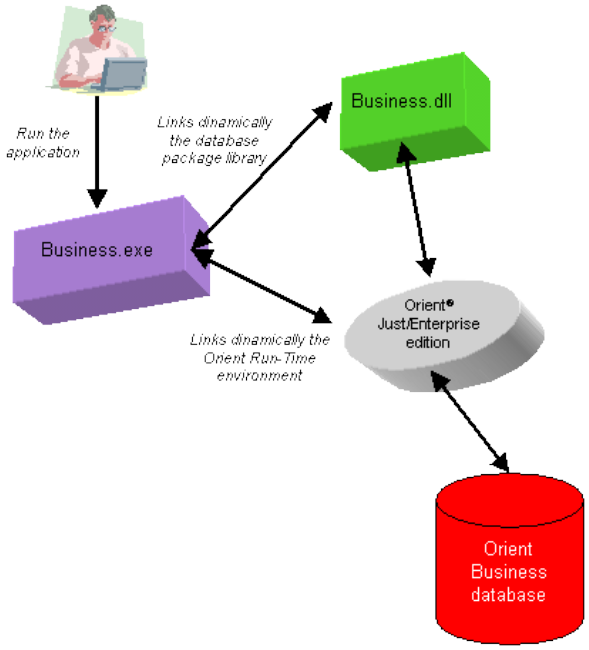


o passo-a-pass



o (da direita para a esquerda).





4.2.1. Código ODL

```
class Endereco
{
        attribute string rua;
        attribute string bairro;
        attribute string numero;
        attribute string cep;
};

class Categoria
{
        attribute unsigned long id;
```

```
        attribute string nome;
};

class Hospedagem
{
        attribute unsigned long id;
        attribute unsigned short vagas;
        attribute string nome;
        attribute string telefone;
        attribute string email;

        attribute Categoria categoria;
        attribute Endereco endereco;
};

class Reserva
{
        attribute unsigned long id;
        attribute date inicio;
        attribute unsigned short diasHospedagem;
        relationship Hospedagem hospedagem;
        relationship Jogo jogo;
};

class Voo
{
        attribute unsigned long numBilhete;
        attribute unsigned long numVoo;
        attribute date data;
        attribute string compArea;
        attribute string origem;
        attribute string destino;
        attribute float preco;
};


class PacoteCopa
{
```

```
        attribute unsigned long id;
        attribute float preco;
        relationship Categoria categoria;
        relationship Voo ida;
        relationship Voo volta;
        relationship set<PacoteJogo> pacoteJogos;
};

class PacoteJogo
{
        attribute unsigned long id;
        relationship Voo vooNacional;
        relationship Reserva reserva;
};

class Jogo
{
        attribute unsigned long id;
        attribute date data;
        attribute float preco;

        attribute string estadio;
        attribute string time1;
        attribute string time2;
        attribute string faseCopa;
};

class Pessoa
{
        attribute unsigned long ssn;
        attribute string nome;
        attribute string telefone;
        attribute string telefoneEmergencia;
        attribute date dataNasc;
        attribute Endereco endereco;
};

class Cliente extends Pessoa
```

```
{
        relationship set<PacoteCopa> pacotesComprados;
};

class Gerente extends Pessoa
{
        attribute unsigned long noContrato;
};

class Loja
{
        attribute unsigned long id;
        attribute string telefone;
        attribute string email;
        attribute Endereco endereco;
        attribute Gerente gerente;
};
```

5. Conclusão

A princípio, a equipe tentou criar o software usando C++. Até o passo de gerar automaticamente os arquivos fontes das classes o OrientDB funcionou perfeitamente, mas para criar a DLL houve várias incompatibilidades da biblioteca padrão do banco de dados com o Visual Studio, que apesar das tentativas de correção, acabaram inviabilizando o projeto via C++.

Após esta falha, a equipe tentou implementar o trabalho usando Java. A etapa inicial de gerar as classes também funcionou, porém houve outra incompatibilidade com as bibliotecas de suporte à comunicação com o banco de dados. Após algum tempo de correções, se conseguiu manter um certa consistência, porém apenas as consultas gerais ficaram funcionando. Consultas mais complexas retornavam o erro informando que algumas funcionalidades não tinham sido implementadas. Além disso, inserções no banco de dados também não funcionavam.

Referências

Ricarte, Ivan L.M. (1998) Sistemas de Bancos de Dados Orientados a Objetos. Campinas: DCA-UNICAMP.

Silberschatz, A., Korth, H. F., and Sudarshan, S. (1999). Sistemas de Bancos de Dados. Pearson Education.

Cattell, R. G. G., Barry, Douglas K., Berler, M., Eastman, J., Jordan, D. (2000) The object data standard: ODMG 3.0, Morgan Kaufmann Publishers Inc., San Francisco, CA

Atkinson, M. P., Bancilhon, F., DeWitt, D. J., Dittrich, K. R., Maier, D., and Zdonik, S. B. (1990) The object-oriented database system manifesto. In SIGMOD Conference, p.395.

Khatchadourian R. (2006) Object Databases: an Analytical Approach.

Object Database Management Systems. “Object Database Vendors”, http://www.odbms.org.

Galante, A., Moreira, E., and Brandão, F. (1999). BANCO DE DADOS ORIENTADO A OBJETOS: UMA REALIDADE. http://www.fsma.edu.br/si/edicao3/banco_de_dados_orientado_a_objetos.pdf.