BDOO: Uma Visão Geral

HEITOR AUGUSTUS XAVIER COSTA[1]
[1]DCC - Departamento de Ciência da Computação
UFLA - Universidade Federal de Lavras
Caixa Postal 37 - CEP 37.200-000 - Lavras (MG)
heitor@esal.ufla.br

Resumo: Com o advento do paradigma de orientação a objetos, um sensível aumento no desenvolvimento de sistemas utilizando este conceito é notado nas organizações. Apesar de ainda estar muito constante o uso de SGBDR para o armazenamento de objetos, o SGBDOO facilita a tarefa de persistir os objetos, diminuindo a complexidade e possibilitando o desenvolvimento de aplicações ainda mais complexas.

Palavras-Chave: Banco de Dados Orientado a Objetos, Características de BDOO, Armazenamento de Objetos.

1 Introdução

A memória volátil do computador (RAM – *Random Access Memory*), embora seja de acesso rápido, tem capacidade de armazenamento limitada sendo bem menor do que a memória não volátil (disco). A limitação faz com que raramente seja possível armazenar todas as informações necessárias para a aplicação trabalhar. Devido a essa limitação e a necessidade de dar continuidade ao trabalho, utilizando as informações processadas anteriormente, surge a necessidade de utilizar o armazenamento de informações em disco. Toda vez que a aplicação precisar de informações que não estejam em memória RAM é utilizada uma das políticas de gerenciamento de memória do sistema operacional para recuperá-las do disco e alocá-las à memória.

A persistência[1] de informações presentes na memória RAM do computador se faz desejada para que se possa trabalhar com elas no futuro, seja com a mesma aplicação dando continuidade ao trabalho outrora terminado, seja com uma nova aplicação.

Estudos para solucionar a questão da persistência de maneira eficiente e eficaz, quando são empregados moldes clássicos[2], para o desenvolvimento de produtos de software, já estão bastante avançados, existindo diversas referências na literatura que apontam boas soluções [5, 9 e 12].

Com o surgimento do paradigma de orientação a objetos, o desenvolvimento de produtos de software deixou os moldes clássicos para incorporar o conceito de objetos.

O desenvolvimento de produtos de software, utilizando os conceitos de orientação a objetos, incentivou o estudo da propriedade de persistência dos objetos, que, dependendo do meio persistente sendo utilizado, pode ser uma característica fácil de implementar. Pesquisadores apontam soluções para a persistência de objetos utilizando um SGBDR (Sistema Gerenciador de Banco de Dados Relacional) [1, 4 e 6], indicando como podem ser implementadas as associações entre as classes e o mecanismo de herança, sem violar a propriedade de encapsulamento. Essa tradução, por vezes, pode ser complexa, fazendo com que seja mais rápido e vantajoso o uso de um SGBDOO (Sistema Gerenciador de Banco de Dados Orientado a Objetos), pois incorpora diversas características facilitadoras para realizar a persistência de objetos [1, 2 e 3].

Logo, se faz necessário a persistência de informações, que neste trabalho consiste de objetos do conceito de orientação a objetos. Os objetos podem estar armazenados em, basicamente, dois tipos de meio persistente: i) arquivo, do sistema operacional, e ii) banco de dados. Atualmente o meio de persistência que está sendo mais amplamente utilizado é o banco de dados. O mecanismo que administra este meio persistente é chamado de Sistema Gerenciador de Banco de Dados (SGBD). Na literatura encontram-se diversas referências que tratam a persistência de objetos utilizando três tipos de SGBD's: Sistema Gerenciador de Banco de Dados Relacional (SGBDR), Sistema Gerenciador de Banco de Dados Orientado a Objetos (SGBDOO) e Sistema Gerenciador de Banco de Dados Objeto-Relacional (SGBDOR). O SGBDOR

[1] Persistência é a estratégia de armazenar informações que futuramente possam ser reutilizadas.
[2] Análise e Projeto Estruturados

compreende características de SGBDR e de SGBDOO, permitindo armazenar dados e objetos.

O SGBDR é o mais utilizado para armazenar os objetos de uma aplicação e é muito comum encontrar nas organizações devido a vários aspectos, entre eles: grande investimento na tecnologia relacional, presença de dados armazenados por sistemas legados não orientados a objetos, características das aplicações (de um modo geral) são transacionais, confiabilidade, longo tempo presente nas organizações, mão-de-obra especializada, o pleno entendimento de como funciona a tecnologia relacional e aplicações legadas (que não utilizam o paradigma de orientação a objetos) precisam acessar os dados

Por outro lado, o uso de um SGBDOO tende a diminuir consideravelmente a complexidade da persistência de objetos, possibilitando o desenvolvedor das aplicações dar maior enfoque à complexidade intrínseca do problema a ser automatizado do que com o armazenamento dos objetos em um meio persistente.

O presente trabalho está organizado da seguinte forma: a seção 2 apresenta o SGBDOO e as suas características, a seção 3 descreve como um SGBDOO suporta os conceitos presentes no paradigma de orientação a objetos e a seção 4 coloca alguns comentários do trabalho. Logo a seguir, o apêndice apresenta um quadro comparativo de algumas características (propriedades) entre quatro SGBDOO.

2 SGBDOO

Um dos tipos mais difundidos de meio persistente é o banco de dados. Segundo [5],

> "Banco de Dados é um repositório para armazenar dados, podendo ser integrado ou compartilhado. Um banco de dados é dito integrado quando existe unificação de diversos arquivos de dados com nenhuma redundância entre eles. Um banco de dados é dito compartilhado quando partes dos dados podem ser compartilhados entre diversos usuários."

[7 e 8] apontam o surgimento dos primeiros BDOO's (Banco de Dados Orientado a Objetos) em meados da década de 80. Em um BDOO, diferentemente de um BDR (Banco de Dados Relacional) que armazena apenas os dados, são armazenados objetos, ou seja, os dados são armazenados juntamente com os métodos que acessam e manipulam esses dados.

Os BDOO's surgiram com o objetivo de suprir a necessidade que as linguagens de programação orientadas a objetos tinham para armazenar os objetos.

Determinados recursos de SGBD's tradicionais, isto é, SGBD's que não possuem características de orientação a objetos, estão presentes, e são muito importantes, em um SGBDOO como: bloqueio para controles de transações concorrentes, recuperação, gravação em duas fases (*two phases commit*), segurança, integridade dos dados e distribuição dos dados.

2.1 Definição

Um SGBDOO é um SGBD que suporta a modelagem e a criação de dados como objetos, ou seja, adiciona funcionalidades de SGBD às linguagens de programação orientadas a objetos. Isto inclui todo tipo de suporte para as classes de objetos, para as associações entre as classes e para o mecanismo de herança entre as classes.

Um dos grandes benefícios dessa composição é a unificação do desenvolvimento da aplicação e do banco de dados em um mesmo modelo de dados e ambiente de programação. Essa unificação colabora no sentido de: i) necessitar de menos código na construção das aplicações; ii) possibilitar o uso, de modo natural, do modelo de dados; e iii) escrever, com mais facilidade, aplicações de banco de dados mais complexas.

Segundo o ODMG (*Object Data Management Group*), ao usar um SGBDOO não há *overhead* de desempenho no armazenamento e na recuperação dos objetos, ou seja, um objeto das linguagens de programação orientadas a objetos corresponde a um elemento básico a ser armazenado por um BDOO, o que não acontece com um SGBDR. O *overhead* existente consiste no mapeamento dos objetos em um outro modelo, que no caso de usar um SGBDR é o mapeamento do modelo de objetos para o modelo relacional (*Impedance Mismatch*).

[2] define SGBDOO,

> "Um SGBDOO deve satisfazer dois critérios: deve ser um SGBD e um sistema orientado a objetos, isto é, deve ser compatível com as linguagens de programação orientadas a objetos. O primeiro critério consiste em incorporar cinco características: persistência, gerenciamento de memória secundária, concorrência, recuperação e facilidade de consulta. O segundo critério consiste em incorporar oito características: objetos complexos, identidade do objeto, encapsulamento, tipos ou classes, herança, *overriding* combinado com *late binding*, extensibilidade e completude computacional."

2.2 Características

Em [1, 2 e 3] são apresentadas características que um SGBDOO deve suportar. Elas estão divididas em três categorias: obrigatórias (mandatórias), desejáveis e abertas.

2.2.1 Características Obrigatórias

As características obrigatórias são características que um SGBD deve satisfazer para que seja considerado um SGBDOO. São elas:

- **Objetos Complexos**: possibilitar a construção de objetos complexos a partir de objetos simples e/ou complexos;
- **Identidade de Objetos**: os objetos devem ter identidade independentemente dos seus valores, ou seja, os objetos devem ter um único identificador que não tenha nenhum significado para a lógica de negócio;
- **Encapsulamento**: armazenar os dados de um objeto (estado do objeto), como também as suas funcionalidades, escondendo a implementação dos dados e da funcionalidade e oferecendo uma interface bem definida;
- **Tipos e Classes Herdados**: assim como a linguagem de programação orientada a objeto suporta herança, o SGBDOO deve suportar.
- ***Overriding, Overloading* e *Late Binding***: permitir a redefinição da implementação da operação (*overriding*) e o resultado da redefinição associar em um único nome (*overloading*). O sistema não pode associar os nomes das operações ao programa em tempo de execução, isto é, o nome das operações deve ser resolvido em tempo de execução (*late binding*);
- **Completude Computacional**: expressar qualquer função computável usando a linguagem de manipulação do SGBD;
- **Extensibilidade**: estender os tipos já definidos pelo sistema de modo transparente para a aplicação e para o desenvolvedor da aplicação;
- **Persistência**: permitir que objetos sejam salvos após a execução de um processo e eventualmente usá-los em um outro processo;
- **Gerência de Armazenamento Secundário**: apoiar um conjunto de mecanismos como: gerência de índice, *clustering* de dados, *buffering* de dados, seleção de caminho de acesso e otimização de consultas;
- **Concorrência**: possibilitar vários usuários acessarem o banco de dados ao mesmo tempo;
- **Recuperação**: suportar mecanismo de falhas de hardware e de software;
- **Facilitar Consultas Simples**: oferecer uma forma rápida e simples de acessar os objetos presentes no banco de dados.

Existe um conjunto de características que ainda não se chegou a um consenso se elas deveriam ser obrigatórias ou desejáveis, são elas: definição de visões e dados derivados, ferramentas de administração de banco de dados, restrição de integridade, facilidade de evoluir o esquema.

2.2.2 Características Desejáveis

As características desejáveis são características que tornam o sistema melhor para trabalhar, contudo não são obrigatórias. São elas:

- **Herança Múltipla**: a partir do momento que se está utilizando uma linguagem de programação orientada a objetos que suporta o uso do mecanismo de herança múltipla, o SGBDOO também deveria suportar tal mecanismo;
- **Checagem e Inferência de Tipos**: não permitir erro de *run-time* após o programa ser aceito pelo compilador. Permitir que somente os tipos base sejam declarados e o sistema deve inferir os outros tipos;
- **Distribuição**: permitir que o sistema de banco de dados seja distribuído;
- **Design para Transação**: apoiar transações de designs, visto que o modelo de transação de sistemas de banco de dados clássico orientado a negócio não é satisfatório, pois tendem a demorar e o critério de seriabilidade não é adequado;
- **Controle de Versão**: fornecer um mecanismo de controle de versão.

2.2.3 Características Abertas

As características abertas são características que possibilitam o projetista fazer escolhas. São elas:

- **Paradigma de Programação**: permitir a escolha de um determinado paradigma de programação (programação lógica, programação funcional, programação imperativa);
- **Sistema de Representação**: é definido pelo conjunto de tipos e conjunto de construções. Dado um conjunto de tipos atômicos e construtores permitir que se estenda de forma diferente;
- **Sistema de Tipos**: há liberdade com respeito aos construtores de tipo. A única facilidade requerida é o encapsulamento;

- **Uniformidade**: definir o conceito de tipo, objeto e método para que exista uniformidade. Neste sentido há três enfoques: i) nível de implementação; ii) nível de linguagem de programação e iii) nível de interface.

3 Armazenamento no BDOO

Para tornar um objeto transiente em um objeto persistente, sendo armazenado em um BDOO, deve-se verificar aspectos presentes no modelo de classes. Estes aspectos consistem em: i) identificar os objetos persistentes e ii) verificar o tratamento das classes e das associações entre as classes no BDOO.

Uma vez que funcionalidades de banco de dados são adicionadas à linguagem de programação orientada a objetos, a identidade dos objetos não precisa ser verificada, pois a linguagem de programação orientada a objetos faz o tratamento necessário. Tão pouco é necessário verificar como o BDOO realiza o tratamento de classes, visto que ela é o seu elemento básico.

3.1 Identificar os objetos persistentes

A identificação destes objetos é realizada quando é estudado o modelo de classes e melhor entendido o domínio da aplicação.

3.2 Tratamento das Associações

a) Associações Simples

Associação 1 para 1

A Figura 1 apresenta uma associação entre classes de cardinalidade 1 para 1. Em cada uma das classes envolvidas é incluído um atributo do tipo ponteiro, representando a associação.

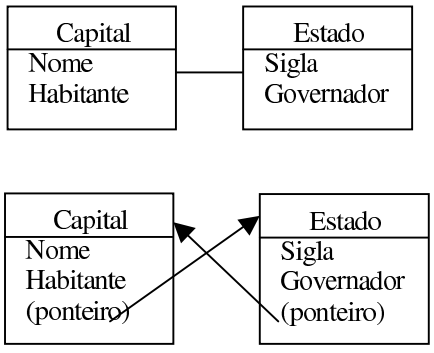

Figura 1 - Associação 1 para 1

Associação 1 para N

A Figura 2 apresenta uma associação entre classes de cardinalidade 1 para N. Em cada uma das classes envolvidas é incluído um atributo. Na classe onde a associação tem cardinalidade N é incluído um ponteiro e na classe onde a associação tem cardinalidade 1, uma coleção é incluída. A composição de ponteiro e coleção, representa a associação. Uma coleção possibilita agregar objetos que podem ser coleções menores ou objetos simples. Uma coleção pode ter características de:

- *Set*: é uma coleção não ordenada de objetos que não permite duplicação;
- *Bag*: é uma coleção não ordenada de objetos que permite duplicação;
- *List*: é uma coleção ordenada de objetos que permite duplicação;
- *Array*: é uma coleção ordenada e indexada de objetos que permite duplicação;
- *Dictionary*: é uma coleção não ordenada de objetos que permite duplicação.

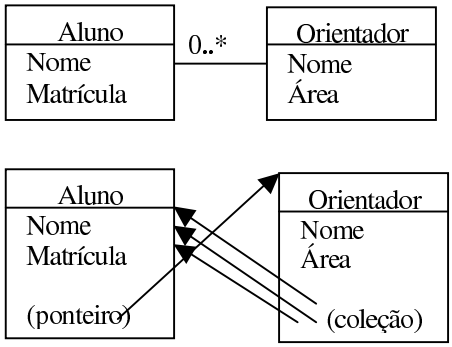

Figura 2 - Associação 1 para N

Associação N para N

A Figura 3 apresenta uma associação entre classes de cardinalidade N para N. Em cada uma das classes envolvidas é incluído um atributo do tipo coleção, representando a associação.

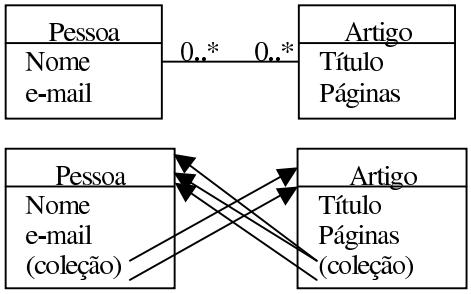

Figura 3 - Associação N para N

b) Associações Avançadas

Classe de Associação

Este tipo de associação é tratado de acordo com a sua cardinalidade. Para associações entre classes de cardinalidade 1 para 1, a classe de associação é tratada como sendo pertencente a uma das classes envolvidas, ou seja, uma das classes incorpora a classe de associação. Para associações entre classes de cardinalidade 1 para N, a classe de associação é tratada como sendo pertencente a classe do lado da associação onde tem cardinalidade 1. Enquanto para associações entre classes de cardinalidade N para N, a classe de associação é tratada como sendo uma classe distinta.

Associação Ternária

As classes envolvidas neste tipo de associação, como regra, são tratadas como sendo classes distintas.

Agregação

Uma agregação de classes pode ser vista como sendo um tipo de associação entre classes, desta forma o tratamento é realizado conforme as associações.

c) Herança

[2] aborda, entre outras características, a herança simples como sendo uma característica obrigatória em um BDOO, pois sem essa característica um banco de dados não é classificado como orientado a objetos. Por outro lado, a herança múltipla é apenas uma característica classificada como desejável, pois não estando presente no banco de dados, ele não deixa de ser classificado com orientado a objetos, mas a sua implementação facilitaria o trabalho do projetista.

Portanto, o tratamento de herança simples é trivial, visto que já está incorporada no BDOO. Contudo, o tratamento de herança múltipla deixa de ser trivial, uma vez que depende que esta característica esteja presente no BDOO sendo utilizado, ou seja, na linguagem de programação orientada a objetos.

4 Conclusão

O desenvolvimento de produtos de software utilizando o paradigma de orientação a objetos faz com que seja necessária a persistência de objetos em um meio persistente. Embora o meio persistente mais utilizado nas organizações seja o SGBDR, devido a diversos aspectos mencionados anteriormente, o SGBDOO cada vez mais está sendo utilizado para realizar esta tarefa. A maior dificuldade em se utilizar um SGBDOO consiste em não existir um consenso de modelo de dados orientado a objetos.

Apêndice A – Comparação entre quatro SGBDOO's: POET, GEMSTONE, JASMINE e ITASCA

A Tabela 1 apresenta quatro SGDBOO's de uma forma comparativa de algumas de suas características. Em [11] é apresentado um quadro comparativo mais completo.

Tabela 1 - Quadro comparativo de quatro SGBDOO

Características	POET	GEMSTONE	JASMINE	ITASCA
Suporte a múltipla herança	✓		✓	✓
Suporte a versão		✓		
Suporte a replicação de dados	✓	✓		
Armazena métodos dos objetos		✓	✓	✓
Suporte a aplicações em Java	✓	✓	✓	✓
Suporte a Object Query Language (OQL)	✓		✓	
Suporte à uma ferramenta CASE orientada a objetos	✓		✓	
Baseado em cliente/servidor	✓	✓	✓	✓
Integração com CORBA	✓	✓	✓	

[10] apresenta um outro quadro comparativo que relaciona dez SGDBOO's. Dentre as várias características apontadas algumas são de cunho acadêmico e outras são de cunho empresarial.

Bibliografia

[1] Ambler, S. W.; **Mapping Object to Relational Databases**; 1998.

[2] Atkinson, M.; Bancilhon, F.; DeWitt, D.; Dittrich, K.; Maier, D.; Zdonik, S.; **The Object-Oriented Database System Manifesto**; http://www.cs.cmu.edu/afs/cs.cmu.edu/user/clamen/OODBMS/Manifesto/htManifesto/Manifesto.html; 1989.

[3] Banchilhon, F.; Delobel, C.; Kanellakis, P.; **Building an Object-Oriented Database System: The Story of O2**; 1992.

[4] Blaha, M.; Premerlani, W.; **Object-Oriented Modelin and Design for Database Aplications**; Prentice Hall; 1998.

[5] Date, C. J.; **Uma Introdução a Sistemas de Bancos de Dados**; Edgard Blücher; 1999.

[6] Heinckiens, P. M.; **Building Scalable Databases Applications: Object-Oriented Design, Architectures, and Implementations**; Addison-Wesley; 1999.

[7] Khoshafian, S.; **Object-Oriented Databases**; Wiley; 1996.

[8] Martin, J.; **Princípios de Análise e Projeto Baseados em Objetos**; Campus; 1994.

[9] Navathe, S. B.; Elmasri, R.; **Fundamentals of Database Systems**; Addison-Wesley; 1999.

[10] **Comparison of ten OODBMS**; http://wwwinfo.cern.ch/asd/cernlib/rd45/evaluations/10odbms.ps; 1994.

[11] **Comparison of four OODBMS**; http://www.iclasb.agh.edu.pl/~jasper/oodb/; 1998.

[12] Silberschatz, A.; Korth, H. F.; Sudarshan, S.; **Sistema de Banco de Dados**; Makron Books; 1999.