BANCO DE DADOS ORIENTADO A OBJETOS: UMA REALIDADE

ALAN CARVALHO GALANTE, Msc
Faculdade Salesiana Maria Auxiliadora
ELVIS LEONARDO RANGEL MOREIRA
Faculdade Salesiana Maria Auxiliadora
FLÁVIO CAMILO BRANDÃO
Faculdade Salesiana Maria Auxiliadora

Resumo

Este artigo tem como objetivo demonstrar que a nova tecnologia de banco de dados orientado a objetos pode ser muito bem utilizada e está totalmente disponível para os desenvolvedores que desejam iniciar no “mundo totalmente OO”. É mostrado quais os conceitos básicos de orientação a objetos, quais os principais modelos de banco de dados e no fim dar-se um foco mais preciso sobre o banco de dados orientado a objetos. Além disso, mostra também o uso do banco de dados orientado a objetos explicitando os mais variados tipos de consultas que se pode realizar e como as realizar, mostrando que a sintaxe OQL não é tão distante da sintaxe da SQL, porém com recursos mais avançados e facilitadores das buscas dos dados. Tudo feito com exemplos práticos e simples de serem entendidos.
Palavras-Chave: Banco de Dados, Orientação a Objetos e OQL.

Abstract

This article aims to demonstrate that the new technology of oriented objects database can be used and is fully available to developers who wish to start in the "entirely OO world”. It is shown that the basic concepts of oriented objects, which the main types of database and in order to give a more precise focus on the database oriented objects. Furthermore, it shows also the use of the database objects aimed at explaining the types of queries that can take place and how the conduct, showing that the OQL syntax is not as far of the syntax of SQL, but with more advanced features and facilitating the tracing data. All done with practical examples and easy to be understood.
Keywords: Database, Oriented Objects, OQL.

1 - Introdução

Nos últimos anos, a procura por ferramentas que facilitem a integração entre o mundo orientado a objetos e o mundo relacional está cada vez mais crescente. Ferramentas de mapeamento objeto / relacional (O/R) nada mais são do que um "tradutor" entre duas linguagens diferentes. Como em todas as traduções, algumas sutilezas da língua acabam sendo perdidas ou utilizam-se muito mais palavras para expressar um conceito que é relativamente simples na língua de origem. No mapeamento O/R não é diferente, acaba-se perdendo uma série de recursos da programação OO, ou tem que escrever muito mais código para simular no banco de dados relacional, algo simples na linguagem. Armazenar objetos em um banco de

dados relacional é uma tarefa difícil pois este modelo não possui mecanismos necessários para representar características básicas do modelo orientado a objeto.

Todos os desenvolvedores das linguagens orientadas a objetos sabem das dificuldades de passar um modelo orientado a objetos para uma persistência relacional. Grande parte do tempo de desenvolvimento de um software é perdida na fase de mapeamento. Utilizando um banco de dados orientado a objetos é possível eliminar ferramentas e códigos para o mapeamento objeto relacional e aproveitar os benefícios do paradigma orientado a objetos sem estar preso pelo banco de dados, permitindo modelos de objetos mais ricos. Nesse cenário, um banco de dados relacional não traz nenhuma vantagem. Talvez a escolha de um modelo Orientado a Objeto seja uma evolução natural.

Hoje, os bancos de dados orientados a objetos são extremamente seguros, e não é necessário um Database Administrator (DBA), o que faz com que o custo do software seja mais barato. Existem vários bancos de dados OO, como o CACHE, ZOPE, GemStone, DB4Objects entre outros.

Com o DB4objects, por exemplo, há a possibilidade de inserir senha e de "criptografar" as informações no banco, caso seja de interesse do usuário. O objetivo deste artigo é mostrar como é a estrutura e uso de banco de dados orientado a objetos. Mostrar quais os principais banco de dados orientado a objetos do mercado.

2 - Linguagem Orientada a Objetos

O paradigma da linguagem orientado a objeto vem ganhando ao longo do tempo muita importância. Nos últimos anos começou a ser bastante empregada e mostra resultados satisfatórios. Passou a ser a linguagem mais utilizada para construção de software. A LOO nada mais é que uma extensão da linguagem modular, e teve seu início na década de 70, com a linguagem SIMULA (Simula Language) que foi desenvolvida na Noruega. A idéia era representar coisas do mundo real, ou seja, fazer simulações. A primeira linguagem foi a SIMULA-68 e logo em seguida nasceu o SMALLTALK, criada pela XEROX, o que popularizou a linguagem orientada a objeto.

Hoje existem diversas linguagens de programação orientada a objeto: Objective-C, C++, CLOS (Common Lisp Object System), Object Pascal, Eiffel, Java, Python, JavaScript, Ruby, C# entre outras.

Para um modelo ser considerado orientado a objeto ele deve conter alguns conceitos básicos: abstração, encapsulamento, herança, e polimorfismo. A abstração é a capacitada de modelar coisas do mundo real, que o programador esteja tentando resolver. Um pequeno exemplo seria uma determinada empresa deseja controlar dados dos seus clientes. Então, o programador colhe os dados genéricos dos clientes e os molda no que se chama de classe. Uma classe descreve um grupo de objetos com propriedades (atributos), similares comportamentos (operações), relacionamentos comuns com outros objetos e uma semântica comum. (ROCHA, 2001)

O encapsulamento tem como principal benefício evitar que interferências externas interfiram na manipulação dos dados. Também protege o mundo externo de alterações na implementação da classe. Com esse princípio, toda ou qualquer transação feita com esses dados só pode ser feita através de procedimentos especificados dentro desse objeto, pelo envio de mensagens.

A herança é o mecanismo que permite a reutilização de códigos. É uma relação entre duas ou mais classes onde existe a Super Classe e a Sub Classe, também conhecidas como classe pai e classe filha respectivamente. Então uma "sub classe" é do tipo "super classe", e contém todos os seus métodos e atributos. Na herança existe a especialização e a generalização. (HAGE, 2005).

O termo polimorfismo quer dizer etimologicamente várias formas. Muitas pessoas acham que sobrecarga e polimorfismo são a mesma coisa, o que não é verdade. Quando um método declarado em uma super classe é herdado para as sub classes e pode ser utilizado em todas as sub classes tendo o mesmo nome e pode atuar em cada sub classe de forma diferente, ou seja, Polimorfismo. Quando um método é herdado e re-escrito para atender necessidades específicas de cada sub classe, se ele ganhar uma nova forma de implementação na sub classe, ou seja, re-escrever um código que já tenha sido declarado e modifica-lo para atender a necessidade de cada classe , isto é caracterizado como Sobrecarga.

3 - Sistema Gerenciador de Banco de Dados (SGBD)

Um SGBD é muito importante para as aplicações nos dias de hoje. Banco de dados são conjuntos de dados estruturados que organizam informação. Para manipular as informações que estão contidas nesse banco de dados, é utilizado um SGBD, que é responsável pelo gerenciamento dos dados. (ELMASRI, 2005)

Um SGBD tem que ter algumas particularidades, e deve facilitar o processo de definir (especificar tipos de dados a serem armazenados), construir (armazenar dados que possam ser manipulados por um SGBD) e manipular (inserir, atualizar e remover base dados de diversas aplicações).

As principais características de um SGBD são:

- **Controle de redundância:** pode-se construir regras para que o gerenciamento seja mais eficaz evitando assim a redundância dos dados e economiza-se espaço em disco. Por exemplo, um aluno só pode ser cadastrado uma única vez em cada curso; cada disciplina só pode ser cadastrada uma vez em um único curso; ou ainda, cada aluno só pode se inscrever uma vez em cada matéria.
- **Restrição a acesso não autorizado:** Em um banco de dados com vários usuários, cada um tem acesso no que lhe é permitido. Com um SGBD é possível restringir os acessos de cada usuário ou grupo de usuários, permitido assim acessos autorizados para cada usuário.
- **Garantia de armazenamento persistente:** Com um SGBD é possível armazenar dados de uma forma organizada. Em muitos SGBD's é necessário fazer a conversão de tipo, porque muitos deles não conhecem o formato dos tipos da linguagem OO, então é necessário fazer a conversão. Umas das vantagens de um banco de dados orientado a objetos é que ele reconhece esse formato não precisando fazer conversão de tipos.
- **Garantia de armazenamento de estruturas para o processamento eficiente de consultas:** Uma outra característica de um SGBD é que além de armazenar dados ele deve prover mecanismo que facilitem a busca, a inserção ou atualização da base de dados.
- **Compartilhamento de dados:** SGBD's multiusuários devem fornecer controle de concorrência para assegurar que atualizações simultâneas resultem em

modificações corretas. Um outro mecanismo que permite a noção de compartilhamento de dados em um SGBD multiusuários é a facilidade de definir visões de usuário, que é usada para especificar a porção da base de dados que é de interesse para um grupo particular de usuários.
- **Fornecimento de múltiplas interfaces:** Devido aos vários tipos de usuários, com variados níveis de conhecimento técnico, um SGBD deve fornecer uma variedade de interfaces para atendê-los. Os tipos de interfaces incluem linguagens de consulta para usuários ocasionais, interfaces de linguagem de programação para programadores de aplicações, formulários e interfaces dirigidas por menus para usuários comuns.
- **Representação de relacionamento complexo entre dados:** Uma base de dados pode possuir uma variedade de dados que estão inter-relacionados de muitas maneiras. Um SGBD deve ter a capacidade de representar uma variedade de relacionamentos complexos entre dados, bem como recuperar e modificar dados relacionados de maneira fácil e eficiente.
- **Backup e restauração:** Garantir backup e restauração de dados é tarefa essencial para qualquer SGBD. Mesmo que as falhas sejam ocasionadas por falhas de software ou hardware ele deve garantir a integridade dos dados.
- **Restrições de integridade:** Num SGBD é possível impor restrições, por exemplo, em uma tabela ALUNO que contém atributos: Nome, CPF, Endereço, Tel, o atributo Nome possa ter no máximo 50 caracteres, e que CPF pode ter 11 caracteres e que Tel pode receber 11 inteiros, ou ainda, a tabela Turma deve ser preenchida com dados da tabela professor e da tabela Aluno etc.

4 - Modelos de Banco de Dados

4.1 - Modelo Hierárquicos

O banco de dados hierárquico foi o primeiro modelo a ser conhecido como modelo de dados. Sua estrutura é do tipo árvore e sua formação se dá através de registros e links, onde cada registro é uma coleção de dados e o link é uma associação entre dois registros. Os registros que precedem outros são chamados de registro pai os demais são chamados de registros filhos. Cada registro tem suas ligações, o registro pai pode ter vários filhos (1:N), o registro filho só pode ter um pai. Caso um determinado registro filho tenha a necessidade de ter dois pais é necessário replicar o registro filho. A replicação possui duas grandes desvantagens: pode causar inconsistência de dados quando houver atualização, e o desperdício de espaço é inevitável.

Para acessar registros em um modelo hierárquico deve obedecer aos padrões desse modelo. A navegação deve começar no topo da árvore e da esquerda para direita. Esse modelo foi muito importante no sistema de banco de dados IMS (Information Management System – é o sistema de banco de dados mais antigo) da IBM. É importante ressaltar que esse modelo era superior a outros modelos da época o que o tornou bem utilizado. Apesar desse, ser o melhor da época ele tem algumas desvantagens como: complexidade dos diagramas de estrutura de árvores, limitações das ligações entre registros - ex.: um filho só pode ter um pai, a navegação é feita através de ponteiros, complexidade na hora das consultas, ausência de facilidades de consultas declarativas, só trabalha com dados primitivos.

4.2 - Modelo em Rede

O modelo em rede surgiu para suprir algumas deficiências do modelo hierárquico. O conceito de hierarquia foi abolido nesse novo modelo, o que permitiu que um registro estivesse envolvido em várias associações, ou seja, esse modelo aceitar relacionamentos (N:M), um filho pode ter mais de um pai, ele só trabalha com dados primitivos. Uma outra característica que difere esse modelo do hierárquico é que ele utiliza grafos ao invés de árvores.

A Data Base Task Group da CODASYL estabeleceu uma norma para esse modelo, uma linguagem própria para definição e manipulação de dados. As ferramentas geradoras de relatório da CODASYL padronizaram dois aspectos importantes dos sistemas gerenciadores de dados: concorrência e segurança. Com isso, o mecanismo de segurança já permitiu que uma determinada área de banco de dados fosse bloqueada para evitar acessos simultâneos, quando necessário.

No modelo em rede qualquer nó pode ser acessado sem precisar passar pelo nó raiz. O sistema mais conhecido dessa implementação é o IDMS da Computer Associates. O diagrama para essa estrutura é formado por registros e links.

4.3 – Modelo Relacional

O modelo relacional surgiu com o propósito de aumentar a independência dos dados nos sistemas gerenciadores de banco de dados; disponibilizar um conjunto de funções apoiadas em álgebra relacional para armazenar e recuperar dados; permitir processamento ad hoc.

A representação do banco de dados desse modelo é feito através de coleções de tabelas. Então quando parte para essa visão, é possível ter tabelas de valores, onde cada tabela tem um nome, e dentro de cada tabela temos as tuplas que são as linhas da tabela, e em cada tabela temos um domínio que é valor atômico, ou seja, são valores indivisíveis no que diz respeito ao modelo relacional. Cada domínio possui um formato de dados.

O modelo relacional também tem algumas restrições: restrições inerentes ao modelo de dados (em uma relação não pode ter tuplas repetidas), restrições baseadas em esquema – são especificações em DDL (data definition language), que são restrições de domínio, de chave, restrições em null, restrições de integridade de entidade e restrições de integridade referencial, e restrições baseadas em aplicação.

4.4 – Modelo Relacional-OO

O modelo relacional OO é a junção do modelo relacional com o modelo OO. Segue o padrão SQL 1999 e estendem a SQL para incorporar o suporte para o modelo de dados relacional-objeto, gerencia transações, processamento e otimização de consultas. Como por exemplo, ele passou a ter construtores de tipos para especificar objetos complexos, passou a ter tuplas e array. Os construtores set, list, bag ainda não foram adicionados ao modelo. Nesse Modelo passou a ter identidade de objeto (reference type), encapsulamento de operações e foram adicionados mecanismo de herança e polimorfismo.

Mesmo com todas essas características a implementação fisicamente continua sendo feita através de tabelas, ou seja, como um modelo relacional. A semântica da aplicação é modelada e representada através de objetos, enquanto sua implementação física é feita na forma relacional.

As principais extensões ao modelo relacional que caracterizam os modelos

relacionais-objeto são: definição de novos sistemas de tipos de dados, mais ricos, incluindo tipos de dados complexos; incorporação de novas funcionalidades ao SGBD para manipular este novos tipos complexos de dados, suporte a herança, possibilidade de manipulação de objetos diretamente por parte do usuário, extensões feitas na linguagem SQL, para possibilitar manipular e consultar objetos.

5 – Banco de Dados Orientado a Objetos

Um banco de dados orientado a objeto é um banco em que cada informação é armazenada na forma de objetos, e só pode ser manipuladas através de métodos definidos pela classe que esteja o objeto. O conceito de banco de dados OO e o mesmo da LOO, havendo uma pequena diferença: a persistência de dados. Existem pelo menos dois fatores que levam a adoção desse modelo, a primeira é que banco de dados relacional se torna difícil trabalhar com dados complexos. A segunda é que aplicações são construídas em linguagens orientadas a objetos (java, C++, C#) e o código precisa ser traduzido para uma linguagem que o modelo de banco de dados relacional entenda, o que torna essa tarefa muito tediosa. Essa tarefa também é conhecida como “perda por resistência”. (ELMASRI, 2005)

O modelo OO ganhou espaço nas áreas como banco de dados espaciais, telecomunicações, e nas áreas científicas como física de alta energia e biologia molecular. Isso porque essa tecnologia oferece aumento de produtividade, segurança e facilidade de manutenção. Como objetos são modulares, mudanças podem ser feitas internamente, sem afetar outras partes do programa. O modelo OO não teve grandes impactos nas áreas comerciais embora tenha sido aplicado em algumas.

Em 2004 os bancos de dados orientados a objeto tiveram um crescimento devido ao surgimento de banco de dados OO livres. A Object Data Management Group (ODMG) com a Object Query Language (OQL) padronizou uma linguagem de consulta para objetos. Uma característica que vale a pena ser ressaltada, é que o acesso a dados pode ser bem mais rápido, porque não é necessário junções. Já que o acesso é feito diretamente ao objeto seguindo os ponteiros. Outra característica importante é que o BDOO oferece suporte a versões, isto é, um objeto pode ser visto de todas e várias versões.

Os bancos de dados OO e relacionais apresentam uma série de características, e cada um tem a sua vantagem e desvantagem. Como por exemplo, os modelos OO utilizam interfaces navegacionais ao invés das relacionais, e o acesso navegacional é bem eficiente implementada por ponteiros. Um problema seria a inconsistência desse modelo em trabalhar com outras ferramentas como OLAP, backup e padrões de recuperação. E os críticos afirmam que o modelo relacional é fortemente baseado em fundamentos matemáticos o que facilita a consulta, já os modelos OO não, o que prejudicaria e muito as consultas. A dificuldade de implementar encapsulamento seria um outro problema, porque como serão feitas as consultas se não é possível ver os atributos.

5.1 - Estrutura e Caracterísiticas

O objeto é formado como se fosse uma tripla (i, c, v), onde o i é o OID do objeto, o c é um construtor, ou seja, que tipo de valor ele vai receber ex.: atom, tuple,

set, list, bag, array e v é o valor corrente. Então o objeto passa a suportar aquilo que foi definido para ele. Se ele vai receber um valor atômico ele só aceitará valores atômicos.

Os construtores de tipos sets, bags, lists e arrays são caracterizados como tipos de coleções e a diferença entre eles é a seguinte: sets e bags, o primeiro só aceita valores distintos enquanto o segundo aceita valores duplicados. Lists e arrays, o primeiro só aceita números arbitrários, enquanto o segundo, o tamanho deve ser pré-estabelecido.

Umas das características dos sistemas OO é ocultar informação e tipos abstratos de dados, sendo que é muito complicado aplicar esse modelo na prática. Por exemplo, nos sistemas atuais para uma consulta em uma determinada tabela é necessário saber todos os atributos da tabela, para formar a consulta. Em um sistema OO, que preza pelo encapsulamento nem toda tabela pode enxergar a outra, o que dificultaria muito as consultas.

A idéia do encapsulamento em um BD OO, já que não dá para ser aplicado a rigor, é pelo menos tratar o comportamento do objeto com funções pré-definidas. Por exemplo, insert, delete, update etc. Ou seja, a estrutura interna do objeto é escondida, e os usuários externos só conhecem a interface do tipo de objeto como os argumentos (parâmetros), de cada operação. Então a implementação é oculta para usuários externos que está incluído a definição da estrutura interna, de dados do objeto e a implementação das operações que acessam essas estruturas. Enfim o BD OO propõe o seguinte, dividir estrutura do objeto em partes visíveis e ocultas então para operações que exigem atualização da base de dados torna se oculta e para operações que exige consultas, torna-se visível. (ELMASRI, 2005)

Em grande parte, os bancos de dados OO tem suas restrições com relação às extensões, isto é, as extensões possuem o mesmo tipo ou classe. Na Linguagem OO nem sempre é assim. SMALLTALK, por exemplo, permite ter uma coleção de objetos de diferentes tipos.

É comum em aplicações de banco de dados que cada tipo ou subtipo de dado possua uma extensão associada, que mantenha a coleção de todos os objetos persistentes daquele tipo ou subtipo. Nesse caso, a restrição é de que todo objeto numa extensão que corresponda a um subtipo também deva ser um membro de extensão que corresponda a seu supertipo. Alguns sistemas de banco de dados OO possuem um tipo de sistema predefinido (chamado de classe raiz (root) ou classe OBJETO, cuja extensão contém todos os objetos do sistema. A classificação então segue, designando objetos para supertipos adicionais que são significativos para a aplicação, criando uma hierarquia de tipo ou hierarquia de classe para o sistema).

Grande parte do modelo OO separa claramente o que é objeto persistente, e objeto transiente. Por exemplo, quando é realizada uma consulta, é carregada uma lista de objetos numa classe transiente (temporária), o sistema pode manipular os dados nessa classe e assim que forem feitas as manipulações necessárias elas deixam de existirem.

Uma das grandes vantagens de um SGBDOO é que ele permite salvar objetos grandes e depois obter a recuperação facilmente desses grandes objetos como texto longos, imagens etc. Eles são considerados não estruturados porque o SGBD não conhece a sua estrutura. A aplicação pode utilizar várias funções para manipular esses objetos. E o mais importante é que o SGBD não conhece essas funções, mas através de técnicas oferecidas por ele é capaz de reconhecer esses objetos e buscá-los no banco de dados. Caso o objeto seja muito grande ele pode utilizar técnicas como buffering e caching.

É importante frisar que SGBDOO não é capaz de processar diretamente condições de seleções e outras operações desses objetos. É necessário que esses dados sejam passados para o BD para que ele possa saber tratar os objetos corretamente. Por exemplo, considere objetos que são imagens bitmap bidimensional. Suponha que a aplicação precise selecionar a partir de uma coleção de tais objetos somente aqueles que incluem certo padrão. Nesse caso, o usuário deve fornecer o programa de reconhecimento do padrão, como um método em objetos do tipo bitmap. O SGBDOO recupera, então, um objeto do banco de dados e aplica nele o método para o reconhecimento do padrão para determinar se o objeto adere ao padrão desejado.

Objetos complexos estruturados são os objetos que contém vários tipos de objetos dentro deles. Por exemplo, um objeto é composto de um list, de tupla, de um set, isto é, o SGBDOO conhece todas essas estruturas, porém o objeto se torna complexo por composto de tipos de objetos diferentes.

5.2 Banco de Dados OO no mercado

Existem vários bancos de dados orientados a objeto, discutir cada um deles é essencial para a tomada de decisão. É importante saber qual modelo é mais apropriado para o uso da sua aplicação. A seguir estão alguns exemplos:

- **CACHÉ**: trabalha com as seguintes linguagens: Java, .Net, C++, XML e outras. É um banco de dados comercial.
- **VERSANT:** trabalha com as seguintes linguagens: Java e C++. É bastante utilizado nos sistemas telecomunicações, redes de transporte, áreas médicas e financeiras. É um banco de dados comercial.
- **DB4Objects**: Trabalha com as seguintes linguagens: Java e .Net. Sua linguagem de Consulta é a Object Query Language (OQL) e é um banco de dados distribuído em duas licenças, a GPL (licença pública Geral) e uma licença comercial.
- **O2**: Trabalha com as seguintes linguagens: C, C++ e o ambiente O2. Sua linguagem de Consulta: O2Query, OQL. Seu gerenciador do Banco de Dados é o O2Engine, e é um banco de dados comercial.
- **GEMSTONE**: trabalha com as seguintes linguagens: Java, C++, C#, XML e outras. Sua linguagem de Consulta é o DML. É um banco de dados comercial.
- **JASMINE**: Possui alta conectividade com Web, suporte à linguagem Java. Pode-se ainda desenvolver aplicações em Visual Basic usando Active/X, em HTML (HyperText Markup Language) usando as ferramentas de conectividade para Web disponíveis no Jasmine, em C e C++ usando APIs e em Java usando interfaces de middleware embutidas no Jasmine. É um banco de dados comercial.
- **MATISSE**: Trabalha com as seguintes linguagens: Java, C#, C++, VB, Delphi, Perl, PHP, Eiffel, SmallTalk. É um banco de dados comercial.
- **Objectivity/DB**: trabalha com as seguintes linguagens: C#; C++; Java; Python, Smalltalk; SQL++ (SQL com objeto - extensões orientadas) e XML (para a importação e a exportação somente). É um banco de dados comercial.
- **Ozone**: trabalha com as seguintes linguagens: Java e XML. É um banco de dados opensource.

5.3 - Object Definition Language (ODL) – Linguagem de Definição de Dados

A ODL foi feita para dar suporte aos construtores semânticos do modelo de objetos ODMG e é independente de qualquer linguagem de programação em particular. O objetivo da ODL é criar especificações de objetos, isto é, classes e interfaces. A ODL não é considerada uma linguagem de programação completa. Ela permite que o usuário especifique um banco de dados independente da linguagem de programação, e utilizando o binding específico com a linguagem para especificar como os componentes ODL podem ser mapeados para componentes em linguagens de programação específica, como C++, SmallTalk e Java.

5.4 – Object Query Language (OQL) – Linguagem de Consulta de Objetos

É a linguagem de consulta declarativa definida pela ODMG (1995). Prevê suporte ao tratamento de objetos complexos, invocação de métodos, herança e polimorfismo. É projetada para trabalhar acoplada com as linguagens de programação com as quais o modelo ODMG define um biding, como C++, SMALLTALK e JAVA. Isso faz com que qualquer consulta OQL embutida em uma dessas linguagens de programação pode retornar objetos compatíveis com os sistemas de tipos dessa linguagem.

Fornece suporte para lidar com set, structure, list e array, tratando estas construções com a mesma eficiência. Permite expressões aninhadas. Pode chamar métodos dos tipos envolvidos na consulta. Não fornece operadores para atualização, mas pode chamar operações definidas nos objetos para realizar esta tarefa, não violando assim a semântica do modelo de objetos, o qual, por definição, é gerenciado pelos métodos especificados no objeto. É possível definir um nome para uma determinada consulta, que é armazenada no BD. Para uso posterior, a consulta é referenciada através do nome definido. Apresenta construtores de objetos, structure, set, list, bag e array. Uma consulta em OQL parte dos pontos de entrada do banco de dados e constrói como resposta, um objeto que é tipicamente uma coleção. Suporta as cláusulas SELECT, FROM, WHERE, GROUP BY, HAVING e ORDER BY.

6 – Object Query Language

A sintaxe básica da OQL é uma estrutura select ... from ... where, igual a da SQL. A seguir será mostrado uma consulta, o nome de todos os departamentos na faculdade 'Engenharia’:

```
SELECT d.dnome From d in departamentos WHERE d.faculdade ='Engenharia';
```

Na maioria das vezes, é necessário um ponto de entrada para o banco de dados para cada consulta, que pode ser qualquer objeto persistente nomeado. Em muitas consultas, ponto de entrada é o nome da extensão de uma classe, ou seja, uma extensão é considerada como sendo o nome de um objeto persistente cujo tipo é uma coleção. Onde na maioria das vezes é um conjunto, exemplos: set<Pessoa>, set<Professor>, set <Disciplinas> e assim por diante.

A utilização de um nome de extensão, por exemplo, departamentos em OQL, como um ponto de entrada refere-se a uma coleção persistente de objetos. Toda vez

que uma coleção for referenciada em uma consulta OQL, devemos definir uma variável de interação (é parecido com as variáveis de tupla que percorrem as tuplas nas buscas da SQL), d em OQL que percorre cada objeto na coleção.

A consulta exemplificada irá selecionar certos objetos de acordo com a cláusula where d.faculdade = 'Engenharia' e serão selecionados para o resultado da consulta. Sendo assim o resultado da consulta é do tipo Bag<string> porque o tipo de cada valor dnome é recuperado no resultado da consulta (embora o resultado seja um set porque dnome é um atributo chave). É importante ressaltar que em geral, uma consulta seria do tipo bag para select ... from ... e do tipo set para select distinct.

Existem três tipos de opções para especificar variáveis de interação: d in departamentos, departamentos d, departamentos as d.

É importante ressaltar que qualquer objeto persistente nomeado que se refira a um objeto atômico (simples) ou a um objeto coleção pode ser utilizado como um ponto de entrada para o banco de dados.

Uma consulta não precisa seguir necessariamente a estrutura select ... from ... where ... . Por exemplo, qualquer nome persistente por si mesmo é uma consulta, e neste cado o resultado é uma referência a esse objeto persistente.

Por exemplo, Q1: departamentos;

Neste caso, retorna uma referência à coleção de todos os objetos persistentes em departamentos, cujo tipo seja set <departamento>.

Caso um nome persistente departamentocc estivesse atribuído a um departamento específico, tipo 'ciencia da computação'. Q1a: departamento; O resultado seria ‘ciencia da computação’.

Uma vez que o ponto de entrada é especificado, o resultado, o conceito de expressão de caminho (path expression) pode ser utilizado para indicar o caminho de atributos e objetos relacionados. Normalmente, uma expressão de caminho inicia-se pelo nome de um objeto persistente ou pela variável de interação que percorre os objetos individuais de uma coleção. Esse nome será acoplado por zero ou mais nomes de relacionamentos ou nomes de atributos conectados utilizando-se a notação de ponto.

Veja no exemplo a seguir, considerando um banco de dados UNIVERSIDADE:

```
Q2 :  departamentocc.coordenador;
Q2a: departamentocc.coordenador.classificacao;
Q2b: departamentocc.possui_professores;
```

Fórmula 1 – Consulta de um banco de dados UNIVERSIDADE - exemplo

A primeira retorna um objeto do tipo Professor, porque é o tipo do atributo coordenador da classe Departamento. O objeto professor que está relacionado com o objeto Departamento cujo nome persistente é departamentocc pelo atributo coordenador. A segunda é bem parecida, só que retorna classificação do objeto Professor. A terceira expressão retorna um objeto do tipo set <Professor> mesmo quando aplicada a um objeto simples, pois é o tipo do relacionamento possui_professor da classe Departamento. O resultado vão ser todos os professores que trabalham no Departamento ciencia da computação.

Consultas como a Q3, não são permitidas porque o resultado poderia ser do tipo set<string> ou bag<string>, uma vez vários professores podem possuir a mesma classificação. Q3: departamentocc.possui_professores.classificaçao;

```
Q3a: select  f.classificacao
        form f in departamentocc.possui_professores;
Q3b: select distinct f.classificacao
        from f in departamentocc.possui_professores;
```

Fórmula 2: Q3a: retornar os professores com suas respectivas classificações
Q3b: retornar o mesmo resultado só que não retorna os resultados repetitivos.
Fonte: ELMASRI, 2005

A OQL Q3a retorna a classificação de todos os professores que trabalham no Departamento de ciencia da computacao, o resultado é do tipo bag, ou seja, trará valores de classificação duplicados. Já a segunda Q3b, trará o mesmo resultado da primeira, com uma exceção, os valores não serão duplicados.

Na OQL também é possível especificar Visões como Consultas Nomeadas. Na OQL visão (view da SQL), da-se o nome de consulta nomeada. Para realizar uma visão, usa-se a palavra-chave define é utilizada para especificar um identificador para a consulta nomeada, e deve ser único entre todos os nomes de objetos, classes, métodos e funções do esquema. Uma vez definida a consulta torna se persistente até que seja redefinida ou removida. Veja o exemplo:

```
V1:     define possui_alunos(nomedepto) as
                select s
                from s in alunos
                where s.estuda_em.dnome = nomedepto;
```

Fórmula 3: V1: retorna os alunos com seus respectivos departamamentos
Fonte: ELMASRI, 2005

Da expressão estuda_em para Aluno, pode utilizar a visão V1, para representar seu inverso sem ter que definir um relacionamento explicitamente. Também pode ser utilizado para representar os relacionamentos inversos que não possuem expectativa de uso freqüente. É possível então fazer: possui_alunos ('ciencia da computação'). Essa OQL retornará um bag de alunos que estudam no departamento de ciência da computação.

Na maioria das consultas é retornada uma lista de objetos. Caso seja necessário retornar um único elemento pode ser feito o seguinte. Existe um operador chamado element na OQL que garante o retorno desse único elemento e de uma coleção unitária c que contém somente um elemento. Se c contiver mais que um elemento ou se c estiver vazia, o operador element causa uma exceção. Veja o caso Q6:

```
Q6 : element (select d
        from d in departamentos
        where d.dnome = "ciencia da computacao);
```

Fórmula 4: Extrai elementos únicos de coleções unitárias
Fonte: ELMASRI, 200

Considerando que o departamento é único para todos os departamentos, o resultado será um único departamento. O resultado é do tipo d:Departamento.

Os operadores de agregação (min, max, count, sum, e avg) operam sobre uma coleção. Esses operadores têm a mesma função, que em SQL. O operador count retorna um tipo inteiro. Os outros (min, max, sum e avg) retornam o mesmo tipo da coleção. Para compreender melhor esses operadores será mostrado duas pesquisas a Q7 e a Q8. A primeira retorna o número de alunos que estudam 'ciencia da computação', enquanto a segunda retorna a média mdc de todos os aluno de nível superior que se especializam em ciencia da computacao.

```
Q7 : count (s in estuda_em ('ciencia da computacao'));
Q8 : avg (select s.mdc
        from s in alunos
        where s.especializa_em.dnome = 'ciencia da computação') and s.turma = 'superior');
```

Fórmula 5: q7 : A primeira retorna o número de alunos que estudam 'ciencia da computação',
q8 : retorna a média mdc de todos os alunos de nível superior que se especializam em ciencia da computacao.
Fonte: ELMASRI, 2005

As expressões de pertinência e quantificação retornam um tipo booleano. Seja v um variável, c uma expressão de coleção, b uma expressão do tipo booleano e e um elemento do tipo dos elementos da coleção c. Então : (e in c) - retorna verdadeiro se o elemento e for um membro da coleção c. (for all v in c:b) – retorna verdadeiro se todos os elementos da coleção c satisfizerem b. (exists v in c:b) – retorna verdadeiro se existir pelo menos um elemento em c que satisfaça b.

A situação a seguir ilustrará a condição de pertinência, por exemplo, é necessário recuperar os nomes de todos os alunos que completaram o curso chamado 'sistema de banco de dados I'. Veja a Q10:

```
Q10 : select s.nome.unome, s.nome.p.nome
        from s in alunos
        where 'sistema de banco de dados I' in
(select c.nome from c in s.disciplinas_cursadas.disciplinas.do_curso);
```

Fórmula 6: q10 : retornar os nomes de todos os alunos que completaram o curso chamado 'sistema de banco de dados I'
Fonte: ELMASRI, 2005

A OQL Q10 mostra um modo mais simples de se especificar a clausula select de consultas que retornam uma coleção de estruturas.

Consultas que retornam valores booleanos, true ou false. Assume-se que exista um objeto jeremy do tipo Aluno. Então a consulta Q11, responde à seguinte pergunta: “jeremy se especializa em ciencias da computaçao”. E a Q12, responde à pergunta “Todos os alunos de ciência da computação são orientados por professores do departamento de ciencia da computação”. Tando Q11 quanto Q12 retornam true ou false, que são interpretados como resposta sim ou não para perguntas anteriores.

```
Q11: jeremy in especializa_em (ciencia da computação);
Q12: for all g in (select s    from s in grad_aluno
        where s.especializa_em.dnome = 'ciencia da computação')
        : g.orientado_por in departamentocc.trabalha_em;
```

Fórmula 7: Q11: responde à seguinte pergunta: “jeremy se especializa em ciencias da computação?”
Q12: responde à pergunta “Todos os alunos de ciência da computação são orientados por professores do departamento de ciencia da computação?”
Fonte: ELMASRI, 2005

Pode ser observado na consulta Q12 a herança de atributos, relacionamentos e operações é aplicada em consultas. Seja s um interator que percorre grad_aluno, pode se escrever s.especializa_em porque o relacionamento especializa_em é herdado por Grad_Aluno de Aluno via Extends. Para finalizar, será mostrado o quantificador exists, representado na consulta Q13, respondendo a seguinte pergunta: “Algum aluno de ciencia da computação possui mdc (média das disciplinas cursadas) igual a 4.0?”. Também é possível notar a operação mdc herdada da classe Grad_Aluno de Aluno via Extends.

```
Q13 : exists g in
(select s from s in grad_aluno
        where s.especializa_em.dnome = 'ciencia da computacao';
                : g.mdc = 4;
```

Fórmula 8: Q13: respondendo a seguinte pergunta: “Algum aluno de ciencia da computação possui mdc (média das disciplinas cursadas) igual a 4.0?”
Fonte: ELMASRI, 2005

Podem existir casos de recuperar o i-ésimo, do primeiro e do último elemento, ou extrair uma subcoleção e concatenar duas listas. Observe a consulta Q14:

```
Q14: first (select struct (professor : f.nome.unome, salario : f.salario)
                from f in professor order by f.salario desc)
```

Fórmula 9: Q14: operador first em uma coleção list que contém os salários dos professores em ordem decrescente de salário
Fonte: ELMASRI, 2005

A OQL 14 mostra o operador first em uma coleção list que contém os salários dos professores em ordem decrescente de salário. Então, o primeiro elemento dessa lista ordenada contém o professor com o maior salário. Neste caso, o resultado mostra um membro da faculdade que ganha o maior salário.

Esta consulta Q15 recupera os três melhores alunos que se especializam em ciência da computação, considerando o mdc;

```
Q15: (select struct (sobrenome : s.nome.unome, primeiro_nome:
                        s.nome.pnome, mdc: s.mdc)
        from s in departamentocc.forma_especialistas order by mdc desc) [0:2];
```

Formula 10: Q15: recupera os três melhores alunos que se especializam em ciência

da computação, considerando o mdc
Fonte: ELMASRI, 2005

Essa consulta select-from-order-by retorna uma lista de alunos de ciencia da computação, ordenados por mdc em ordem descendente. Sendo que o primeiro elemento de uma coleção possui índice 0, de maneira que a expressão [0:2] retorna uma lista contendo o primeiro, segundo, e o terceiro elemento.

A cláusula group by da OQL, é bem parecida com da SQL, só que ela fornece uma referência explícita à coleção de objetos em cada grupo ou partição. Observe a Q16:

```
Q16 : select struct  (nomedepto, numero_de_especialistas :
        count (partition)) from s in alunos
        group by nomedepto :  s.especializa_em.dnome;
```

Fórmula 11: Q16: Operador de agrupamento. Retorna o número de alunos em cada departamento
Fonte: ELMASRI, 2005

O resultado é do tipo set<struct> (nomedepto: string, partition: bag <struct> (s:Aluno)>)>, que contém uma estrutura para cada grupo partition com dois componentes: o valor do atributo de agrupamento (nomedepto) e a bag de objetos de aluno no grupo (partition). A cláusula select retorna o atributo de agrupamento (nome) e a bag de objetos de aluno no grupo (partition). A cláusula de select retorna o atributo de agrupamento (nome do departamento) e uma contagem do número de elementos em cada partição (ou seja, o número de alunos em cada departamento), na qual partition é a palavra-chave utilizada para se referir a cada partição.

7 – Conclusão

É possível implementar um sistema de complexo usando um SGBD Orientado a Objetos. A manipulação de objetos nativamente, aumenta a performance e os ganhos de desempenho de linguagens orientadas a objeto como o Java e as linguagens da plataforma .NET. Permite maior desempenho e redução no tempo de desenvolvimento do software, já que não é necessário traduzir o modelo orientado a objeto para um modelo relacional, eliminando a complexidade extra e a perda de performance com a conversão para outros formatos como SQL. Uma outra vantagem desse banco é de não precisar de um DBA, pois sua administração é de responsabilidade do próprio analista de sistemas.

Por ser uma tecnologia nova, muitas empresas preferem não arriscar, pois o modelo relacional ainda é muito empregado nos dias atuais. Migrar de uma tecnologia bem difundida no mercado para uma que está apenas começando seria muito arriscado. Essa é uma das desvantagens do BD OO. O que ocasiona poucas aplicações nessa nova tecnologia. Muitos ainda não confiam na sua integridade. À medida que a complexidade for aumentando, as empresas vão cada vez mais buscar por alternativas que consigam adequar às suas necessidades.

Referências bibliograficas

BOOCH, Grady. RUMBAUGH, James. JACOBSON, Ivar. UML – Guia do usuário. Leitura: Elsevier. 13º edição. 2000.

DEITEL, H. M. Java, como programar/ H.M e P.J Deitel; trad. Carlos Arthur
Lang Lisbôa - 4.ed - Porto Alegre : Bookman, 2003

ELMASRI, Ramez. Sistemas de banco de dados. Ramez Elmasri e Shamkant B. Navathe; revisor técnico Luis Ricardo de Figueiredo. São Paulo: Addison Wesley, 2005.

FOWLER, Martin. UML essencial: um breve guia para a linguagem padrão de modelagem de objetos. Martin Fowler; tradução João Tortello. 3ª ed. Porto Alegre: Bookman, 2005.

FURLAN, José Davi. Modelagem de Objetos através da UML - the Unified Modeling Language. São Paulo: Makron Books, 1998.

HAGE, Armando. Análise e Projeto de Sistemas. Disponível em < sig.ufpa.br:8080/hage/disciplinas/uml/Aula1UML.pdf>. 2005. Acesso em 20/10/2005.

NOGUEIRA, Admilson. UML. 2005. Disponível em < http://www.imasters.com.br/artigo/3636>. Acesso em 25/07/2007.

ROCHA, A. R. R. et al. Unified Modeling Language. 2001. Disponível em <http://www.dei.unicap.br/~almir/seminarios/2001.1/5mno/uml/>. Acesso em 22/06/2007.