FACULDADE DE TECNOLOGIA DE SÃO PAULO

SISTEMA DE BANCO DE DADOS ORIENTADOS A OBJETOS

GUILHERME CANTUÁRIA DE CARVALHO

SÃO PAULO

2011

FACULDADE DE TECNOLOGIA DE SÃO PAULO

SISTEMA DE BANCO DE DADOS ORIENTADOS A OBJETOS

GUILHERME CANTUÁRIA DE CARVALHO

TECNÓLOGO EM PROCESSAMENTO DE DADOS

GABRIEL ISSA JABRA SHAMMAS - ORIENTADOR

SÃO PAULO

2011

AGRADECIMENTOS

Agradeço aos meus pais, que me apoiaram durante toda a minha vida e sem o apoio e incentivos de ambos não teria chegado a este momento.

A Deus, por estar presente em todos os momentos e me fornecendo força nas dificuldades enfrentadas.

Aos meus amigos, que contribuíram para o meu crescimento pessoal e tornaram o período do desenvolvimento do trabalho mais prazeroso.

Por fim agradeço ao meu orientador e a todos que contribuíram de alguma forma para a criação deste trabalho.

SUMÁRIO

LISTA DE FIGURAS.....vi

LISTA DE SIGLAS.....vii

RESUMO.....ix

ABSTRACT.....x

1 INTRODUÇÃO.....1

1.2 OBJETIVOS.....2

2 CONCEITOS DE BANCO DE DADOS.....3

2.1 DADOS.....4

2.2 ABSTRAÇÃO DE DADOS.....4

3 MODELOS DE DADOS.....6

3.1 MODELO HIERÁRQUICO.....6

3.2 MODELO DE REDE.....7

3.3 MODELO RELACIONAL.....8

3.4 MODELO RELACIONAL NÃO-NORMALIZADO.....9

3.5 MODELO ENTIDADE-RELACIONAMENTO.....10

4 PARADIGMA DA ORIENTAÇÃO A OBJETOS.....12

4.1 O QUE É UM OBJETO ? 13

4.2 CLASSE DE OBJETOS.....14

4.3 HERANÇA.....15

4.4 POLIMORFISMO.....16

4.5 ENCAPSULAMENTO.....16

5 BANCO DE DADOS ORIENTADOS A OBJETOS.....18

5.1 O QUE SÃO ? 19

5.3 CONCEITO DE ORIENTAÇÃO A OBJETOS PARA BDOO.....21

5.3.1 PERSISTÊNCIA DOS OBJETOS.....21

5.3.2 OBJETOS PARA BANCO DE DADOS.....22
5.3.2.1 OBJETOS COMPLEXOS.....23
5.3.2.2 IDENTIFICADOR DE OBJETOS.....23
5.3.3 HIERARQUIA DE CLASSES E HERANAÇA PARA BDOO.....23
5.3.4 ENCAPSULAMENTO.....24
5.4 CONCEITOS DE BANCO DE DADOS PARA BDOO.....25
5.4.1 TRANSAÇÕES.....25
5.4.1.1 TRANSAÇÕES DEMORADAS.....26
5.4.1.2 TRANSAÇÕES ANINHADAS.....27
5.4.1.3 TRANSAÇÕES EM COOPERAÇÃO.....27
5.4.2 CONCORRÊNCIA.....27
5.4.3 GERENCIAMENTO DE RECUPERAÇÃO.....29
5.4.4 VERSIONAMENTO.....30
5.4.5 CONSULTAS.....30
6 EXEMPLOS DE BDOO.....32
6.1 O_2.....32
6.2 OBJECTSTORE.....34
6.3 POSTGRES.....36
6.4 JASMINE.....37
7 CONCLUSÃO.....39
REFERÊNCIAS BIBLIOGRÁFICAS.....41

LISTA DE FIGURAS

Figura 2.1 – Conceitos de bancos de dados

Figura 2.2 – Os níveis de abstração

Figura 3.1 – Modelo Hierárquico

Figura 3.2 – Modelo de Rede

Figura 3.3 – Modelo Relacional

Figura 3.4 – Modelo Relacional Não-Normalizado

Figura 3.5 – Relacionamento do MER

Figura 4.1 – Declaração de Classes

Figura 4.2 – Herança entre Classes

Figura 5.1 – Estrutura de BDOO

Figura 6.1 – Declaração de Dados do O_2

Figura 6.2 – Manipulação de Dados do O_2

Figura 6.3 – Declaração de Dados do Objectstore

Figura 6.4 – Manipulação de Dados do Objectstore

Figura 6.5 – Declaração de Dados do Postgres

Figura 6.6 – Manipulação de Dados do Postgres

Figura 6.7 – Declaração de Dados do Jasmine

Figura 6.8 – Manipulação de Dados do Jasmine

LISTA DE SIGLAS

BD	-	Banco de Dados
SGBD	-	Sistema Gerenciador de Banco de Dados
MR	-	Modelo Relacional
1FN	-	Primeira Forma Normal
MRNN	-	Modelo Relacional Não-Normalizado
CAD	-	*Computer-Aided Design*
CAM	-	*Computer-Aided Manufacture*
CASE	-	*Computer-Aided Software Enineering*
GIS	-	*Geographic Information System*
SQL	-	*Structure Query Language*
OQL	-	*Object Query Languade*
ODQL	-	*Object Database Query Language*
MER	-	Modelo de Entidade-Relacionamento
OO	-	Orientação a Objetos
Xerox PARC	-	Xerox Palo Alto Research Center
POO	-	Programação Orientada a Objeto
BDOO	-	Banco de Dados Orientados a Objetos
OID	-	Identificadores de Objetos
ACID	-	Atomicidade, Coerência, Isolamento e Durabilidade
TPS	-	Transações por Segundo

SGBDOO - Sistemas Gerenciador de Banco de Dados Orientados a Objetos

RESUMO

Sistemas de banco de dados representam um papel fundamental em aplicações de processamento de informação não-numérica, onde a utilização de tais sistemas permite facilitar a manutenção de dados consistentes que podem ser compartilhados por diversas aplicações, as quais não precisam conhecer o modo como tais dados são internamente armazenados ou representados. A tecnologia de banco de dados já está madura para diversas aplicações, principalmente nas áreas administrativas e comerciais.

Entretanto, novas categorias de aplicações têm demandado o gerenciamento de dados mais complexos. Para essas aplicações, o paradigma de orientação a objetos surge como alternativa adequada para a representação e manipulação dos dados, mas é preciso ainda integrar de forma adequada esta tecnologia a sistemas de banco de dados.

Sistemas gerenciadores de bancos de dados orientados a objetos constituem a proposta corrente para a solução desse problema. Neste trabalho apresentam-se os aspectos básicos da modelagem e da tecnologia de tais sistemas, com um breve estudo sobre a evolução dos bancos de dados e seus respectivos modelos, analise dos paradigmas da orientação a objetos e dos conceitos de banco de dados.

ABSTRACT

Database systems represent an essential part in the nom-numeric information processing application, where the use of such systems allows to make easier the maintenance of consistent data and can be shared by various applications without needing to know the way where such data are internally stored or represented. The database technology is already mature for various applications, mainly in administrative and commercial areas.

However, new categories of application have demanded more complete data management. For those application, the object-oriented paradigm emerges as a suitable alternative for representation and handling of data, but it’s still needed a properly integration of this technology to database systems.

Object-oriented database management systems form a current proposal for solving this problem. In this paper are presented the basic aspects of the modeling and of the technology of such systems with a brief study about the evolution of database and their respective models, object-oriented paradigms and database concepts.

1 INTRODUÇÃO

A cada dia as empresas dependem mais de grandes e crescentes volumes de informação para suas decisões de negocio, e essa informação deve estar sempre correta e de fácil acesso. Por isso essas empresas exigem sistemas de informação que atendam não só a suas necessidades funcionais, mas também estejam integrados com a necessidade de mudanças no negocio.

Para atender essas necessidades, surgem inúmeros avanços computacionais a fim de possibilitar esse desenvolvimento dos negócios. Entre todos esses avanços computacionais pode-se citar a tecnologia de orientação a objetos.

“As tecnologias de orientação a objetos estão cada vez mais presentes em todas as áreas da computação, desde linguagens de programação até os bancos de dados. Através do modelo orientado a objeto pode-se expressar problemas do mundo real de forma mais fácil e naturalmente usando-se componentes modularizados.” [KHO94]

Está nova tecnologia ao mesmo tempo em que trouxe inúmeras vantagens computacionais, também trouxe alguns problemas, dentre eles o fato que sistemas que utilizam a técnica de orientação a objetos necessitam armazenar seus registros e estes são armazenados em banco de dados relacionais, dos quais tem uma enorme difusão no mercado. Entretanto tal medida não é uma boa prática, pois o mundo real está sendo representado na forma de tabelas e uma possível adaptação do modelo orientado a objetos para o modelo relacional exige um gasto enorme de tempo e produção. Além desses motivos, bancos de dados relacionais apresentam-se inadequados a algumas necessidades dos novos avanços de sistemas computacionais.

Diante disso, este trabalho aborda uma das possíveis soluções para a correção desta divergência entre sistemas baseados no paradigma da orientação aos objetos e banco de dados relacionais. Solução esta denominada como Banco de Dados Orientados a Objetos.

Os bancos de dados orientados a objetos integram os conceitos de orientação a objetos como herança, identidade do objeto e tipo abstrato de dados com aptidões de um banco de dados como transações, segurança, recuperação, etc. Através de construções orientadas a objetos, os usuários podem esconder os detalhes de implementação de seus módulos, compartilhar a referencia a objetos e expandir seus sistemas através de módulos existentes. As aptidões de banco de dados são necessárias para assegurar o compartilhamento simultâneo e a continuidade das informações nas aplicações. [KHO94]

1.1 OBJETIVOS

O objetivo deste trabalho é abordar a área de Bancos de Dados Orientados a Objetos de uma forma unificada, através de uma visão conceitual tendo por base o Modelo de Entidade e Relacionamento.

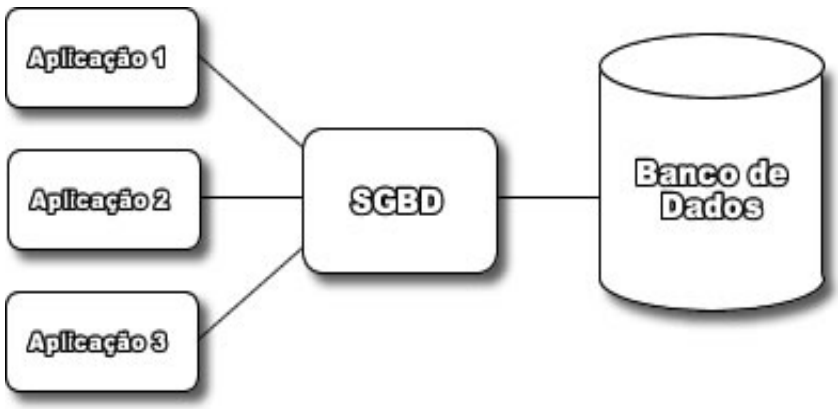
Serão apresentados os conceitos das tecnologias de Banco de Dados e suas respectivas evoluções seguidas pelos conceitos de Orientação a Objetos. Diante disso estabelece-se uma base para o desenvolvimento da tecnologia de Bancos de Dados Orientados a Objetos.

Com isso espera-se preencher todo tipo de lacuna e dúvidas a respeito dessa tecnologia de Banco de Dados que é pouco utilizada pelo mercado, entretanto é uma tecnologia de alto desempenho.

2 CONCEITOS DE BANCOS DE DADOS

Bancos de Dados (BD) é uma expressão que se impôs na língua portuguesa. A expressão vem do inglês, onde se usou por pouco tempo a original Databanks, que foi logo substituída por Database, isto é, Base de Dados. Esse nome é bem mais sugestivo, pois um banco de dados não funciona como um banco, emprestando dados. Funciona muito mais como um repositório de dados, que são usados em diversas aplicações, isto é, uma base sobre a qual atuam essas aplicações, e que esta disponível para o desenvolvimento de outras que usam os mesmos dados [SET05].

Explorando a referencia citada acima, BD são gerenciadores de grandes grupos de informações. Esse gerenciamento consiste em definir a estrutura para o armazenamento de informações e o fornecimento de mecanismo para manipulá-las. Esse gerenciamento é executado pelos Sistemas de Gerenciamento de Banco de Dados (SGBD), softwares que possuem recursos para efetuar esse tipo de manipulação. Além disso, os BD precisam fornecer segurança para que informações não sejam disponibilizadas a usuários não autorizados e possuir integridade uma vez que as informações serão disponibilizadas para diversos usuários.



A Figura 2.1 ilustra os conceitos de bancos de dados.

Para compreender melhor o conceito de BD, é necessário compreender alguns conceitos como:

- Dados
- Abstração de Dados

2.1 DADOS

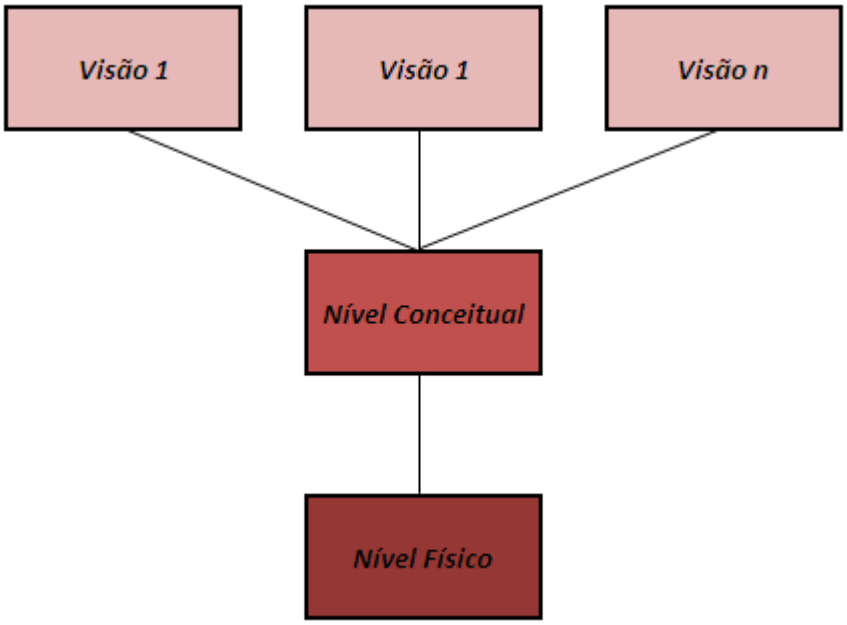
Dado é a estrutura fundamental sobre a qual um sistema de informação é constituído. Símbolo intencionalmente destacado para representar uma característica ou propriedade da realidade a ser tratada. [Pompilho, 1995].

2.2 ABSTAÇÃO DE DADOS

Uma das maiores proposta de BD é fazer com que os usuários acessem e recuperem os dados eficientemente de uma forma abstrata, ou seja, uma vez que os usuários não são treinados em computação, a complexidade dessas operações fica escondida deles através de diversos níveis de abstração que simplifiquem a interação do usuário com os BD.

- Nível Físico é o nível mais baixo de abstração. Descreve como os dados estão realmente armazenados. Num Nível Físico, complexas estruturas de dados de baixo nível são descritas em detalhes.
- Nível Conceitual de abstração descreve quais são de fatos os dados que devem pertencer aos BD e suas respectivas relações. Esse nível é bastante usado pelos administradores de BD. Nesse nível o banco de dados inteiro é descrito em termos de um pequeno número de estruturas relativamente simples.

- Nível Visual é o mais alto nível de abstração e descreve apenas parte do banco de dados representado por uma visão. Este nível é definido para simplificar a interação dos usuários com os BD.



A Figura 2.2 ilustra os níveis de abstração

3 MODELO DE DADOS

Modelo de Dado é uma coleção de ferramentas conceituais que ajudam a formar e descrever a estrutura de dados de um SGBD. Os primeiros modelos de dados utilizados foram o modelo hierárquico e o modelo de rede. Em meados anos 70 um novo modelo se impôs como padrão nos SGDB da época. Esse modelo foi o modelo relacional, que possuía uma estrutura extremamente simples, sendo essa uma das razões pela sua rápida difusão entre os ambientes comercias. Acompanhado por essa rápida difusão estavam suas limitações que não demoraram a aparecer. Decorrente a essas limitações, não demorou a surgir novas extensões do modelo relacional a fim de efetuar correções e melhorias.

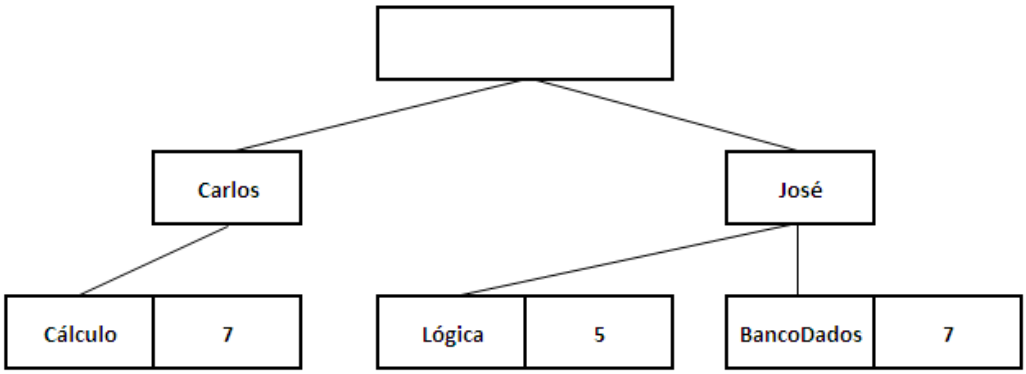
Dentre todas as novas extensões, destaca-se o modelo relacional não-normalizado. Outro destaque desse período é o surgimento de alguns modelos conceituais, com o objetivo de facilitar a modelagem de dados, em um nível mais elevado que no modelo relacional. Nessa categoria será abordado o modelo de entidade e relacionamento.

3.1 MODELO HIERÁRQUICO

O modelo hierárquico provavelmente foi o primeiro modelo a ser comercializado. Seu desenvolvimento só ocorreu devido à consolidação dos discos de armazenamento endereçáveis, pois esses discos possibilitaram o estudo da representação hierárquica da informação nas estruturas de endereçamento físico. Este modelo consiste em um conjunto ordenado de arvores, em que cada nó da arvore representa um tipo de registro do qual cada registro é uma coleção de campo (atributos), cada um contendo somente uma informação. Estes registros são conectados através de ligações entre si.

Cada registro pode ter quantos registros-filho quiser, porém só poderá ter apenas um registro-pai. Diante disso as ligações entre registro-pai para registro-filho é de um para zero ou muitos.

Devido à restrição citada acima, o conteúdo de um registro-pai pode ser repetidos diversas vezes. Este fato pode ocasionar tanto desperdício de espaço devido à quantidade de informações repetidas, como tornar o banco inconsistente caso uma possível atualização não seja efetuada em todos os registros repetidos.



A Figura 3.1 ilustra o Modelo Hierárquico.

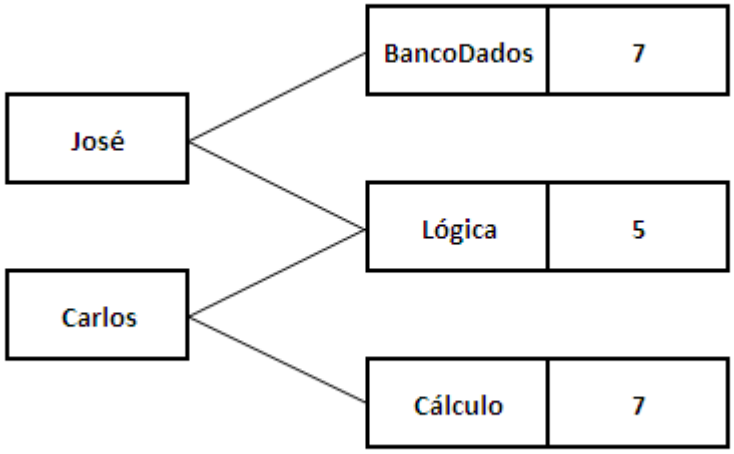
3.2 MODELO DE REDE

O modelo de rede consiste em um conjunto de registros que são conectados um ao outro por meio de ligações. Cada registro, assim como no modelo hierárquico, é uma coleção de campos (atributos), cada um dos quais contendo apenas um valor de dado. Já as ligações são as associações entre dois registros.

O modelo de rede é bastante semelhante ao modelo hierárquico apresentado anteriormente, sendo que a principal diferença entre ambos consiste em que no modelo

hierárquico, cada registro-filho possui exatamente um registro-pai, enquanto no modelo de rede cada registro-filho pode possuir qualquer número de registros-pai.

Esta diferença se deve ao fato da substituição da estrutura de armazenamento em árvore por uma estrutura de armazenamento, citada anteriormente, na forma de uma coleção de registros conectados através de ligações.



A Figura 3.2 ilustra o Modelo de Rede.

3.3 MODELO RELACIONAL

O modelo relacional (MR) assim que apresentado foi logo aderido pelo comercio, no qual se estabeleceu como modelo principal para aplicativos de processamento de dados. Ainda hoje diversos SGBD utilizam seus conceitos básicos.

Tendo como base a teoria dos conjuntos e a álgebra relacional, o MR foi desenvolvido por Codd em meados anos 70.

De acordo com [NAU99], o MR consiste em dados armazenados em tabelas denominadas relações. Cada linha da tabela representa um elemento do conjunto de dados e cada coluna da tabela contém valores de um conjunto definido, denominado domínio. Uma linha da relação é chamada de tupla.

Em sua primeira publicação, Codd definiu que os valores dos domínios deveriam ser atômicos, sendo assim, não se podem armazenar atributos compostos, conjuntos, vetores ou listas em uma célula da tabela. Esta restrição leva o nome de Primeira Forma Normal (1FN). Isto torna o MR bem mais simples, porém deixa engessada a modelagem de dados em algumas situações que a 1FN não é convencional.

Algumas das melhorias evidenciadas com o MR são:

- Aumento da independência dos dados nos SGBD.
- Prover um conjunto de funções apoiadas em álgebra relacional para armazenamento de recuperação de dados.

Nome	Matéria	Média
José	Lógica	5
Carlos	Cálculo	7

Figura 3.3 ilustra o Modelo Relacional.

3.4 MODELO RELACIONAL NÃO-NORMALIZADO

O modelo relacional não-normalizado (MRNN) é bastante semelhante ao MR se diferenciando somente no fato que a 1FN não é uma restrição. Diante disso o MRNN

permite a representação de atributos multivalorados ou compostos ou, de uma forma geral, valores que são relações.

Segundo [NAU99], soluções com o MRNN, além de diminuir a redundância das chaves das relações e o número destas, é mais simples e eficiente. Com esse modelo ainda é possível a representação de listas, textos e outras estruturas que são necessárias para algumas aplicações avançadas, tal como CAD/CAM.

Nome	Matéria	Nota
José	Lógica	7 8
Carlos	Cálculo	7 7

A Tabela 3.4 ilustra o Modelo Relacional Não-Normalizado.

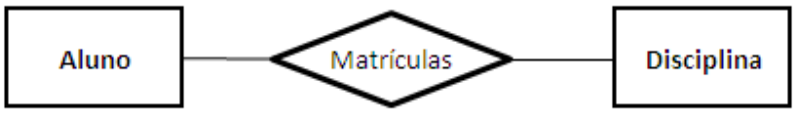
3.5 MODELO ENTIDADE-RELACIONAMENTO

“O modelo entidade-relacionamento (MER) é baseado na percepção do mundo real que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos. Ele foi desenvolvido para facilitar o projeto de banco de dados permitindo a especificação de um esquema de empresa. Tal esquema representa a estrutura lógica geral do banco de dados.” [KOR93]

O MER trata-se de um modelo conceitual, representado em forma de gráfico, que inicialmente foi introduzi por Peter Chen em 1976, porém desde então já sofreu algumas atualizações em sua estrutura.

Como citado anteriormente, a estrutura do MER consiste em dois pilares, assim como propõem seu nome. São eles:

- Entidades: são nada mais que uma representação abstrata de entes (objetos) do mundo real. Um grupo de entidade que possui a mesma natureza no mundo real pode ser denominado como conjunto de entidades. Essas entidades possuem atributos que as distinguem uma das outras, através da representação das informações que cada entidade possui. Os atributos podem receber algumas classificações como compostos, onde o atributo pode ser composto por vários subatributos; multivalorados, onde o atributo leva um elemento do conjunto de entidades a um conjunto de valores; determinante, onde o valor do atributo não se repete em mais de um elemento do conjunto entidade.
- Relacionamentos: são nada mais que representações conceituais que os objetos (representados pelo conjunto de entidade) do mundo real possuem entre si, ou seja, dados dois conjuntos de entidades, elementos de um podem se relacionar com elementos de outro.



A Figura 3.5 ilustra o Relacionamento do MER.

4 PARADIGMA DA ORIENTAÇÃO A OBJETOS

A orientação a objetos (OO) não é um conceito novo. Seu roteiro pode ser evidenciado pela Noruega, no inicio dos anos 60 do qual se encontrava conectada com uma linguagem chamada Simula-67, desenvolvida por Kristen Nygaard e Ole-Johan Dahl no Centro Norueguês de Computação. A Simula-67 apresentou pela primeira vez os conceitos de classes, rotinas correlatas e subclasses muito parecidas com as atuais linguagens orientadas a objetos.

Depois, em meados da década de 70, cientistas do Xerox Palo Alto Research Center (Xerox PARC) criaram a primeira e robusta linguagem orientada a objeto denominada Smalltalk.

Apesar da boa divulgação nos circuitos universitários, tanto a Simula-67 quanto a Smalltalk eram relativamente inacessíveis para a comunidade de desenvolvimento.

Nos ano s 80, a linguagem C tornou-se muito popular entre os desenvolvedores e diante disso proporcionou sua expansão para a linguagem C++, que suporta programação orientada a objeto (POO) e sendo criada por Bjarne Stroustrup. As subseqüentes versões comerciais e os incrementos nas ferramentas da linguagem C++, assim como um maior número de distribuidoras, ajudaram a chamar a atenção da comunidade de desenvolvimento à POO. Com isso os profissionais da comunidade de desenvolvedores estavam aptos a aprender o paradigma da POO.

A OO, portanto não é nova e como será apresentado propõe abstrações do mundo real e traz vantagens como a reutilização não apenas do código, mas também de requisitos, analise, projeto e especificação, permitindo maior produtividade, pois permite capturar processos, procedimentos e regras das empresas.

Modelos orientados a objetos são mais flexíveis, permitem manutenção mais fácil e melhor administração do domínio do problema. Tendo o objeto como ponto central das analise e do desenvolvimento, é possível tratar as funções (operações ou métodos) e os dados (atributos ou propriedades) em uma mesma unidade. Mecanismos como herança e polimorfismo permitem acrescentar novas operações e atributos a classes já existentes, possibilitando a escrita de código reutilizável e extensível.

4.1 O QUE É UM OBJETO?

Segundo do dicionário, objeto é “tudo que é apreendido pelo conhecimento, que não é o sujeito do conhecimento; tudo que é manipulável e/ou manufaturável; tudo que é perceptível por qualquer um dos sentidos; coisa, peça, artigo de compra e venda; matéria, assunto; o que é conhecido, pensado ou representado, em oposição ao ato de conhecer, pensar ou representar”.

Portanto objeto é a representação de elementos físicos do mundo real, que descobrimos estudando suas características (atributos) e seus comportamentos (ações).

Para a OO, após estudar e analisar os objetos do mundo real, separamos mentalmente aqueles que interessam num exercício de abstração.

Recorrendo novamente ao dicionário, abstração é o “ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade; o resultado de abstrações (termo, conceito, idéia, elemento de classe etc.)”.

Assim, a OO trata-se de uma modelagem dos objetos do mundo real, estudando-os e criando classes a partir de suas características e comportamentos. Esse trabalho é

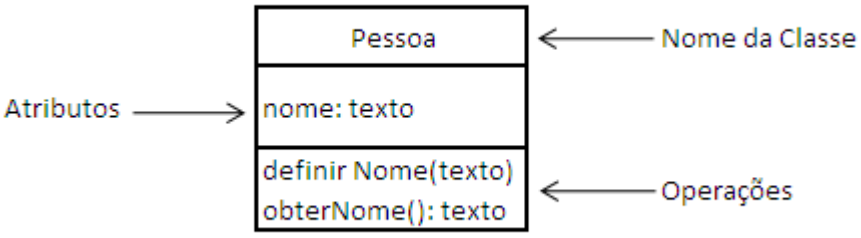
executado pelo analista que usando um processo intelectual é capaz de abstrair esse estudo da realidade e representar os objetos dentro de um contexto.

4.2 CLASSE DE OBJETOS

Ao comparar diferentes objetos, notamos que eles possuem atributos e comportamentos semelhantes, dependendo da finalidade para a qual são analisados. Usando a abstração podemos agrupar alguns objetos de acordo com seus atributos e comportamentos em comum.

Uma classe, portanto, é um modelo onde os objetos são originados (instanciados). Ela possui a definição dos atributos e das ações de um tipo de objeto e é obtida pela classificação de objetos com a mesma estrutura de dados e o mesmo comportamento. Assim por exemplo, em um dado modelo, José, Antonio, João etc. podem ser instancias da classe Pessoa, uma vez que possuem atributos comuns (nome) e comportamentos comuns (andar, deitar, etc.).

A declaração de uma classe é dividida em geral em duas partes: as variáveis de instancias (atributos de um objeto a ser instanciado) e a sua interface, que são operações (métodos) disponíveis para a classe.

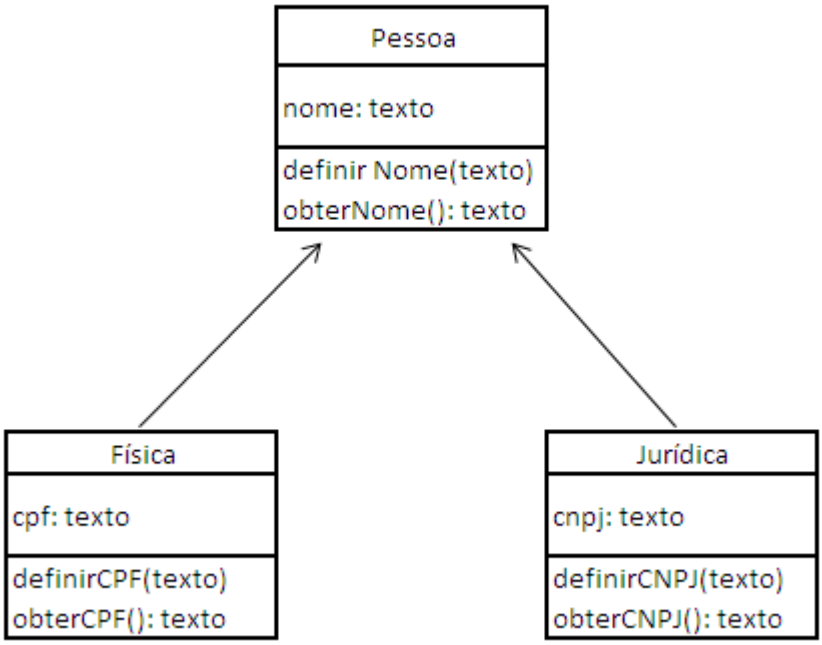


A Figura 4.1 ilustra a declaração de classes.

4.3 HERANÇA

Comparando e analisando objetos do mundo real, podemos evidenciar atributos e comportamentos em comuns entre objetos, fazendo com que esses sejam agrupados em um mesmo tipo de classes. Entretanto, aprofundando um pouco mais na analise e observação, descobre-se que alguns desses objetos pertencentes a essa classe ainda possuem características próprias alem daquelas em comum. Diante disso o mecanismo de herança consegue suprir essa diferença entre as características dos objetos.

De um modo mais especifico podemos definir herança como a capacidade de uma classe herdar os atributos e comportamentos de outra classe. Neste casso a classe herdada é denominada de superclasse e a classe herdeira é denominada de subclasse. Assim a subclasse herda atributos e comportamentos da classe imediatamente superior – a superclasse.



A Figura 4.2 ilustra a herança entre classes.

A herança na OO pode ser única onde uma classe herda características de apenas uma classe ou múltipla onde uma classe pode herdar características de uma ou mais classes. Não são todas as linguagens de POO que tem suporte para heranças múltiplas.

Um fator muito importante na herança é o fato de que se deve tomar um enorme cuidado na manutenção desse recurso, pois qualquer tipo de mudança de uma superclasse acarreta imediatamente a propagação para as subclasses.

4.4 Polimorfismo

Pelo dicionário temos a seguinte definição de polimorfismo: “Particularidade de certas substancias que tomam formas muito diversas; multiforme; que se apresenta sob numerosas formas”.

Para OO, polimorfismo é o principio em que classes derivadas de uma mesma superclasse podem invocar operações que têm a mesma assinatura, mas comportamentos diferentes em cada subclasses e produzindo resultados diferentes dependendo de como cada objeto implementa a operação.

Com outras palavras, é a capacidade de objetos diferentes possuírem operações com o mesmo nome e a mesma lista de argumentos, mas que executam tarefas de forma diferentes.

4.5 ENCAPSULAMENTO

O encapsulamento ou também conhecido como ocultação de informações, é uma técnica que consiste em separar os aspectos internos dos aspectos externos de um objeto, isto é determinados detalhes ficam ocultos, assim como o nome sugere, aos demais objetos tendo como referencia apenas ao próprio objeto. De outra forma pode-se

afirmar que encapsulamento é a proteção da estrutura interna do objeto por traz dos métodos.

Essa característica é muito importante, pois possibilita uma maior independência de dados, uma vez que a implementação das estruturas de dados dos objetos não precisa ser conhecida por quem utiliza os objetos.

Vale lembra que o encapsulamento estabelece uma dependência com a relação de herança, pois objetos da subclasse herdam todas as definições de atributos e operações da superclasse.

5 BANCOS DE DADOS ORIENTADOS A OBJETOS

“O propósito dos sistemas de banco de dados é o gerenciamento de grandes corpos de informação. Os primeiros bancos de dados desenvolveram-se a partir do sistema de gerenciamento de arquivos. Esses sistemas evoluíram primeiro em bancos de dados hierárquicos e de rede, depois em bancos de dados relacionais.” [KOR93]

Inicialmente os primeiros bancos de dados originaram-se a partir dos sistemas de gerenciamentos de arquivos. Com isso foi possível separar pela primeira vez a aplicação dos dados. Acompanhado dos novos bancos de dados, vieram às modelagens de dados que no atual momento se tornaram muito eficiente para o desenvolvimento de aplicações, cujas estruturas se apresentavam de forma uniforme e simplificada e também eram sempre orientadas ao registro. Estes modelos ficaram conhecidos como modelo hierárquico e modelo de rede. Ambos os modelos não possuíam forte fundamentos teóricos e também não suportavam noções de independência de dados físicos e lógicos.

Os bancos de dados relacionais surgiram logo em seqüência aos modelos mencionados acima. Este modelo trouxe muitas vantagens em relação aos modelos hierárquicos e de rede. O formato relacional, tabelado é simples e fácil de entender. A álgebra relacional oferece fundamentos matemáticos capazes de suportar metodologias de análises de dados e a possibilidade de agrupar dados dinamicamente em tabelas virtuais em tempo de execução, possibilitou, aos bancos de dados relacionais, o desenvolvimento de aplicações com um nível mais elevado de independência dos dados do que seus predecessores. Diante disso o modelo relacional caiu nas graças dos desenvolvedores e ainda hoje serve como base para inúmeros SGBD.

“Os sistemas gerenciadores de bancos de dados relacionais existem, na verdade, para registros de dados. Mas a maior parte do mundo real não é feita de registros.” [Whiting, 1989].

O avanço computacional em diversas áreas, a difusão das redes e de novas tecnologias, evidenciou novas necessidades que os bancos de dados relacionais se mostraram inadequados. Dentre os novos usos da capacidade computacional, podemos destacar o CAD/CAM (projeto/manufatura auxiliado por computador), o CASE (desenvolvimento de sistemas de computação auxiliado por computador), o GIS (sistemas de informação geográfica) e etc.. Todas essas aplicações manipulam dados com uma estrutura extremamente complexa.

Na década de 80, visando aumentar a produtividade do desenvolvimento, facilitar a reutilização e manutenção da analise, requisitos, projetos e especificações, a POO se tornou muito popular entre a comunidade de desenvolvimento, sendo considerada por alguns como uma revolução na forma de se fazer sistemas.

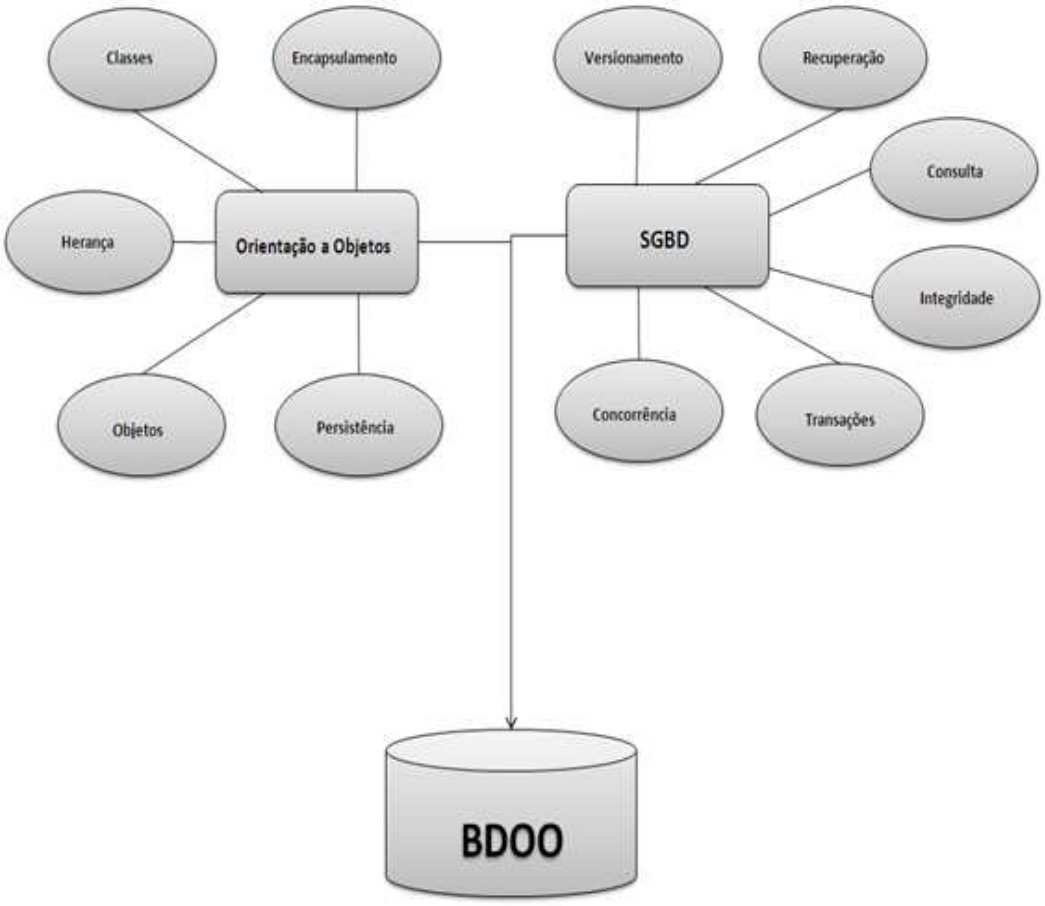
Visando acompanhar a tendência da época e também possibilitar resolver as limitações que os bancos de dados relacionais possuíam, foi proposto um novo sistema de banco de dados, o banco de dados orientados a objetos (BDOO).

5.1 O QUE SÃO?

De uma forma bem simples pode-se dizer que BDOO são nada mais que a junção entre conceitos de OO com conceitos de SGBD, ou seja, ele é todo baseado nos paradigmas da OO unido aos objetivos básicos dos SGBD.

Um BDOO é basicamente um sistema em que a unidade de armazenamento é o objeto, com o mesmo conceito das linguagens de POO. A diferencia fundamental está no fato que em BDOO, os dados não deixam de existir após o encerramento do programa, ou seja, os objetos continuam a existir mesmo se o sistema venha a ser encerrado. Com isso os dados, isto é, os valores dos atributos que fazem referência a seus respectivos objetos não deixam de existir. Este conceito é conhecido como persistência.

Outra característica essencial dos BDOO é que eles oferecem suporte a versões, ou seja, os objetos podem ser vistos de todas ou varias versões.



A Figura 5.1 ilustra a estrutura de BDOO.

5.3 CONCEITO DE ORIENTAÇÃO A OBJETOS PARA BDOO

Em BDOO, os objetos são tratados em um nível lógico e também possuem características que não são encontradas na POO. Essas características serão tratadas ao decorrer do trabalho, porém podemos destacar a persistência de dados, ou melhor, dizendo a persistência de objetos. Desta forma conceitos como “classes” e “herança” sofrem adaptações a fim de possibilitar a aplicação em BDOO.

Para [NAU99], os conceitos de OO básicos e necessários para a estrutura de um BDOO são:

- Persistência de Objetos.
- Objetos complexos.
- Presença de identificadores de objetos.
- Mecanismos de herança.
- Métodos.

5.3.1 PERSISTÊNCIA DE OBJETOS

Sem duvida nenhuma, a persistência de objetos é uma característica primordial para os BDOO, pois alem de ser uma característica que possibilita diferenciar BDOO das linguagens de POO, ela também é fundamental para os BDOO.

Persistência de objetos consiste em não deixar com que objetos deixem de existir, ou seja, ao finalizar a execução de um programa que tenha como base uma linguagem de POO, todos os objetos instanciados deixam de existir. Com isso os valores dos atributos do objeto também desaparecem. Se isso fosse aplicado aos BDOO, o resultado seria uma catástrofe, pois os dados que serão inseridos em um BDOO na

forma de atributos de objetos devem existir mesmo após o encerramento do programa de gerenciamento, tendo seu estado armazenado em um meio físico persistente, a menos é claro que os atributos sofram algum tipo de alteração ou exclusão, sendo estas solicitadas pelo usuário.

Para que os BDOO possam funcionar, é preciso aplicar a persistência de objetos em sua estrutura, a fim de que os dados não desapareçam.

Existem diversas maneira de torna o objeto persistente e todas elas variam conforme o sistema utilizado. Dentre essas formas podem-se considerar as seguintes:

- Por tipo de classe onde os objetos pertencentes às classes assim declaradas serão persistentes.
- Por chamada explícita onde o objeto pode se tornar persistente após a sua criação através de comandos reservados.
- Por referência onde objetos referenciados por objetos persistentes (objetos raízes) também se tornam persistentes.

5.3.2 OBEJOS PARA BANCO DE DADOS

O modelo de OO suporta a representação das abstrações e comportamento dos objetos. Com isso os BDOO incorporam as características dos objetos da linguagem de POO integrada com noções de estrutura de dados e de comportamento. O conjunto de atributos descreve o estado interno dos objetos. Cada “ocorrência” do objeto no banco de dado é denominada de instancia do objeto.

A estrutura de objetos em banco de dados é muito similar ao conceito de entidades, quando aplicado ao MER.

5.3.2.1 OBJETOS COMPLEXOS

Para uma representação direta de objetos complexos, dos quais nos modelos relacionais não possuem representação, proposta a utilização do modelo de dados das linguagens orientadas a objetos. Nota-se que a primeira forma normal não é respeitada nos BDOO, pois podemos representar em um objeto valores não atômicos, como conjuntos, listas, vetores etc..

Assim como nas linguagens orientadas a objetos, a representação se torna extremamente fácil devido a esse fato (não normalizado).

5.3.2.2 IDENTIFICADOR DE OBJETOS

Nos BDOO, os identificadores de objetos (OID) têm uma identidade muito mais forte que na linguagem de POO, pois nos BDOO, os objetos têm a necessidade de continuarem a existir mesmo após a execução do programa. Isto ocorre devido ao fato que os objetos podem voltar a serem usados futuramente.

O OID deve ser único e imutável durante toda a existência dos objetos e também deve ser valido para todo o banco de dados uma vez que ajudam na recuperação do objeto e são utilizados para estabelecer relacionamentos entre os objetos.

5.3.3 HIERARQUIA DE CLASSES E HERANÇA PARA BDOO

Todo o conceito e característica de classe e herança da OO devem estar presentes em BDOO. Diante disso outra importante capacidade em BDOO é o gerenciamento do conceito de herança dentro de uma hierarquia de classes armazenáveis. Da mesma forma que em linguagem de POO, os BDOO podem criar novas classes em função de classes já existentes.

Uma hierarquia de classes oferece muito mais flexibilidade para efetuar alterações na estrutura de um BDOO (incluindo novos atributos ou métodos nos objetos) bem como possibilita a evolução do esquema de banco de dados através da adição de classes novatas na hierarquia.

As “classes do sistema” são um exemplo típico da característica de herança dentro de um SGBD orientado a objeto. Isto porque a maioria dos SGBD orientado a objetos possuem uma coleção de classes e objetos dentro de sua estrutura. Essa coleção esta estruturada de forma hierárquica. Caso necessário, uma determinada aplicação pode herdar atributos e métodos dessas classes de sistema a fim de implementar suas próprias classes. Em ressumo essas classes do sistema de SGBD orientado a objetos possuem métodos que têm como propósito o armazenamento, manipulação, controle de concorrência entre outras funções.

5.3.4 ENCAPSULAMENTO

De acordo com [NAU99], o conceito de encapsulameno para BDOO continua sendo o mesmo, porém, quando se faz uma consulta ao banco de dados, não é possível prever todas as consultas e atualizações que o usuário possa desejar. Assim, não se pode agregar todos os métodos nas classes de antemão.

Sendo assim pode-se considerar que o encapsulamento não é adequado aos BDOO em algumas situações, entretanto o encapsulamento é uma das características fundamentais para POO e descartar o seu uso em BDOO pode descaracterizar parcialmente o conceito de OO.

5.4 CONCEITOS DE BANCO DE DADOS PARA BDOO

Os BDOO sofrem adaptações em sua forma de gerenciamento assim como sofrem adaptações na implantação dos conceitos de OO. Como mencionado anteriormente, essas adaptações possibilitam o uso desses recursos nos BDOO.

Transações, concorrência, consultas são alguns exemplos dos quais necessitam de uma adaptação para que possam ser usados em BDOO. A seguir será apresentado todos esses recursos entre outros.

5.4.1 TRANSAÇÕES

“Tradicionalmente, uma transação é um programa que lê ou grava objetos persistentes”. [KHO94]

Transação é um conceito fundamental de todo sistema de banco de dados. O conceito fundamental de transação é englobar vários passos em uma única operação. Os espaços intermediários entre esses passos não são vistos pelas demais transações simultâneas e caso alguma falha impeça por ventura a finalização da transação, então nenhum dos passos intermediários irá afetar o banco de dados, esteja ela em estado de executado, em execução ou a executar.

Uma transação deve levar o banco de dados de um estado coerente para outro. Para que essa coerência ocorra, cada transação de um banco de dados deve passar pelo teste de ACID (Atomicidade, Coerência, Isolamento e Durabilidade).

- Atomicidade emprega o conceito de que uma transação é executada por completo ou não é executada.

- Coerência emprega o conceito de que as transações mapeiam um banco de dados persistente de um estado coerente para o outro.
- Isolamento emprega o conceito de que transações não lêem resultados intermediários de outras transações não efetivadas.
- Durabilidade emprega o conceito de que uma vez que uma transação é efetivada, seus efeitos ficam permanentemente no banco de dados, mesmo que ocorram falhas.

As transações em BDOO podem ser definidas em três vertentes que serão apresentadas a seguir.

5.4.1.1 TRANSAÇÕES DEMORADAS

Algumas transações em BDOO como CAD, CAM e CASE, exigem um modelo diferenciado do modelo de banco de dados mais tradicionais.

Em SGBD mais tradicionais, as transações interagem (atualizam ou referenciam) somente alguns registro, diante isto, tornam-se mais curtas. Alem disso esses tipos de SGBD dão muita ênfase em medidas de performance (em tempos de TPS – transações por segundo) e robustez (capacidade de recuperação). Já em SGBDOO, algumas transações tendem a levar um tempo relativamente longo, isto porque SGBDOO interagem com muitos objetos complexos, tornando as transações mais complexas e ao mesmo tempo mais demoradas.

5.4.1.2 TRANSAÇÕES ANINHADAS

Transações aninhadas são utilizadas para melhorar as transações mais demoradas. Esse modelo consiste em decompor a transação principal (transação de alto

nível) em subtransações, a fim de executá-las. Todas as subtransações devem ser executadas com êxito para que se possa obter o resultado da transação principal.

Caso uma subtransações venha por ventura falhar, fica a critério da transação de alto nível tentar executá-la novamente ou ate mesmo cancelá-la. Seguindo o conceito de ACID, quando uma transação de alto nível é cancelada, por conseqüência todas as suas subtransações também serão canceladas, independente do fato dessas terem sido executadas com sucesso.

5.4.1.3 TRANSAÇÕES EM COOPERAÇÃO

Este modelo de transação é utilizado para resolver os problemas de transações que necessitam trabalhar em conjunto para serem concluídas, sendo que essas transações podem ser executadas por diversos usuários. Em um contraste das transações mais tradicionais, as transações em cooperação podem ver os resultados imediatos umas das outras.

5.4.2 CONCORRÊNCIA

Concorrência para SGBD permite o acesso simultâneo aos dados para diferentes usuários. O mais notável algoritmo de controle de concorrência existente para SGBD são os bloqueios, que se baseiam em uma estratégia de não deixar transações serem executadas através de bloqueios, caso haja conflito no acesso e/ou atualização de um objeto. Em SGBDOO cada objeto persistente pode ser bloqueado. Isto normalmente é feito por meio de uma tabela de bloqueio que armazena os OID dos objetos bloqueados.

Existem vários tipos (modos) de bloqueios, porém os dois modos mais simples são:

- Modo de Leitura
- Modo de Gravação

O modo de leitura (modo compartilhado) permite que diversas transações de operações de leituras sejam executadas no mesmo objeto concorrentemente. O modo de gravação (modo exclusivo) reserva o acesso (operações de leitura e gravação) de um objeto à transação que contem esse bloqueio no momento. Quando uma transação mantém um bloqueio exclusivo, sobre um objeto, nenhum outro bloqueio pode ser disponibilizado. Esse controle do uso de bloqueio é feito por um recurso conhecido como “matriz de compatibilidade”. Essa matriz gerencia se a solicitação de bloqueio pode ser concedida ou não.

Segundo [KHO94] existem dois aspectos relevantes no que diz respeito ao controle de concorrência em SGBDOO.

- Bloqueio de hierarquia de classe: as classes em BDOO são organizadas em hierarquias de herança. Caso uma superclasse venha a ser bloqueada, o mesmo ocorrerá de forma implícita para todas as suas subclasses, que são descendentes diretas da superclasse.
- Bloqueio de objeto complexo: os BDOO podem conter objetos que referenciam ou incorporam outros objetos, isto pode causar problemas, pois algum objeto complexe pode possuir subobjetos bloqueados, ocasionando problemas de atualizações e/ou possíveis problemas de acesso. Tal problema pode ser resolvido pelo bloqueio de intenção. Neste bloqueio as subclasses de uma classe não são bloqueadas implicitamente.

Somente as instancia dependentes do objeto-pai são bloqueadas. Diante disso as outras instancias ficam livres para serem acessadas.

5.4.3 GERENCIAMENTO DE RECUPERAÇÃO

Como já visto anteriormente, as transações em BDOO, assim como em banco de dados tradicionais, utilizam o conceito de atomicidade, isto quer dizer, ou são executadas por completo ou não executam nada.

A confiabilidade e a grata recuperação de falhas que possam ocorrer é um fator muito importante e que se encontra presente tanto em banco de dados tradicionais como em BDOO.

Os BDOO possuem muitas formas de contornar algum possível problema relacionado à falha em seu sistema. Mecanismo de duplicação ou espelhamentos de dados são exemplos de recursos usados no gerenciamento de recuperação nos BDOO, porém o recurso mais utilizado é sem duvida nenhuma é a estrutura de *logs*, que consiste e armazenar imagens anteriores e posteriores dos objetos atualizados. A imagem anterior é o estado do objeto antes de sua atualização e a imagem posterior é o estado do objeto após sua atualização.

Ambas as imagens são registradas no *log* durante a execução normal das transações. Diante isto, assim que houver uma falha, o sistema gerenciador de recuperação é acionado com o intuito de efetuar a correção a fim de tornar o banco de dados coerente novamente, através dos *logs* e dos dados da memória secundaria (disco rígido).

5.4.4 VERSIONAMENTO

O controle de versão é uma característica importante e presente em alguns sistemas como, por exemplo, CAD ou CASE. A necessidade de armazenar versões anteriores devido a uma possível situação que seja imprescindível o uso desta versão é muito comum hoje em dia em determinadas aplicações. Com isto, assim que um objeto sofre uma alteração no banco de dados, é essencial o armazenamento da versão recém atualizada.

De acordo com [KHO94], o gerenciamento de versões de um BDOO consiste em um conjunto de ferramentas e construções que automatizam ou simplificam a construção e a organização de versões ou configurações. Sem estas ferramentas, caberia ao usuário criar e manter suas próprias versões.

Assim que um objeto é versionado, uma espécie de raiz que aponta para todas as versões do objeto é criada. Durante o versionamento, um objeto pode sofrer desde alterações em seu estado até modificações estruturais. A OID é a propriedade comum entre todas as versões de um objeto.

5.4.5 CONSULTAS

O acesso a dados armazenados é feito basicamente de duas formas. Uma é pela linguagem de programação, da qual utiliza os OID dos objetos e a outra é por linguagem de consulta, em geral derivadas as SQL.

O acesso aos dados por meio de linguagem de programação vem tentar resolver umas das principais criticas que os SGBD relacionais sofrem o chamado “não casamento de impedâncias”. Quando se utiliza uma linguagem de programação

qualquer para o desenvolvimento de uma aplicação, é necessário reestruturá-la para que se possa interagir com as linguagens de consultas de bancos de dados relacionais, já que a única estrutura do MR é a relação. Isto também é valido para o contrario, ou seja, assim que se obtém o resultado de uma consulta através da linguagem de consulta, é necessária uma adaptação dos dados para a estrutura da linguagem de programação.

Quando se usa uma linguagem e um banco de dados de uma única estrutura, este trabalho é poupado, implicando em diminuição de código e de tempo de execução, além de possibilitar o compartilhamento de estruturas entre aplicações.

6 EXEMPLOS DE BDOO

Nos últimos anos foram produzidas inúmeras implementações de BDOO, entre protótipos de empresa, pesquisas de universidades e produtos comerciais. Pode-se citar o O2,ObjectStore, Orion, Postgres entre outros.

A seguir serão apresentadas as principais características de alguns sistemas existentes. Também serão abordadas as estruturas de linguagens de definição e de manipulação de dados dos respectivos sistemas.

6.1 O_2

O sistema O_2 inicialmente foi criado na França em 1988 através de um projeto experimental conveniado. Com o fim do convênio em 1991 o sistema passou de projeto experimental para um produto sendo comercializado pela empresa recém criada O_2 Technology.

O sistema O_2 fornece um ambiente gráfico para criação de telas (O_2Look), um browser para navegar entre os objetos do banco de dados. Para facilitar a migração a partir de SGBD relacionais, há uma ferramenta de conversão e migração (O_2DBAccess). O sistema O_2 fornece integração com as linguagens de programação C, C++ e recentemente Java.

A declaração de dados ocorre através de uma linguagem (O_2C) que se originou a partir da linguagem C. Esta linguagem permite herança múltipla e não tem distinção na declaração de objetos persistentes dos não-persistentes.

Os tipos de dados do O_2 são boolean, character, integer, real, string e bit. Existem também construtores de tipos complexos de dados, que podem ser aplicados recursivamente.

```
Class Pessoa
   Type tuple(nome: string
              RG: string
              Endereço: string)

   Method Muda_Endereco(novoEndereco: string)
end
```

A Figura 6.1 ilustra a declaração de dados do O_2 .

O sistema O_2 garante automaticamente a integridade referencial em seus relacionamentos, ou seja, se um dos objetos for excluído do banco de dados, o sistema automaticamente torna as referências a este objeto nulas.

A manipulação de dados no sistema O_2 pode ser efetuada de duas formas: uma é pela própria linguagem de programação nativa do sistema e a outra é pela OQL que é uma linguagem de consulta do tipo SQL.

Já a persistência dos objetos é concedida ou através do comando “add name” que faz com que o OID de um objeto seja associado a um nome de uma variável, para ser recuperado posteriormente, ou quando um objeto é referenciado por um outro objetos que já é persistente.

```
/* Cria objeto aluno persistente */
add name Eugenio: Aluno;
/* Cria conjunto de alunos persistente */
add name AlunosDoBCC: set(Aluno);
/* Cria um aluno, a princípio, não persistente */
Francisco: Aluno;

run body
{
Eugenio.Nome = "Eugênio Akihiro Nassu";
/* modificando/iniciando o objeto Aluno */
...
AlunosDoBCC += set(Eugenio);
/* colocando o Aluno em um conjunto */
AlunosDoBCC += set(Francisco);
/* o aluno Francisco também se torna persistente
pois é referenciado por um objeto persistente */
...
/* modifica o endereço do Aluno Eugenio */
Eugenio.Muda_Endereco("Alameda Barros 380");
...
}

select tuple(Aluno: m.Al.Nome, Nota: m.nota)
from m in Matriculas
where m.Disc.nome = "MAC-110" and m.SemAno = "1/95"
```

A Figura 6.2 ilustra a manipulação de dados do O_2.

6.2 OBJECTSTORE

O sistema Objectstore é um banco de dados produzido pela empresa Object Design Inc. Este sistema foi criado com o intuito de tornar a linguagem C++ uma linguagem para banco de dados.

A principal característica do sistema Objectstore é a uniformidade no tratamento de objetos comuns e persistentes, sem grande perda de desempenho, graças a um esquema de uso de memória virtual.

Os objetos persistentes fazem parte da memória virtual, portanto ficam gravados no disco.

Quando se faz um acesso as objeto, é gerada uma falha de página de memória no sistema, que faz com que o objeto seja carregado na memória principal.

O sistema Objectstore usa tipos e a linguagem C++ como base. Diante disso as declarações são quase idênticas ao C++, permitindo que programas escritos em C++ sejam convertidos para Objectstore.

```
class Pessoa {
  char *nome;
  char *RG;
  char *endereço;

  void Muda_Endereco( char *NovoEndereco );
}
```

A Figura 6.3 ilustra a declaração de dados do Objectstore.

O armazenamento de objetos no banco de dados ocorre ao declarar uma variável do tipo database. Já a manipulação de dados ocorre através do uso de métodos.

```
#include <objectstore/objectstore.H>
#include <records.H>

main(){

  //declara uma variável bando de dados
  database *db;

  //abre um banco de dados
  db = database::open("dados/IME");

  //inicia transação
  transaction::begin();
  //cria conjunto persistente
  os_set <Aluno *> Alunos = os_set(Aluno *)::create(db);
  .
  .
  .
  //cria novo objeto. Já é criado como persistente, pela extensão do
  //operador new
  aluno a = new(db) aluno ("Eugênio");
  //insere objeto no conjunto
  Alunos->insert(a);
  //grava as alterações em definitivo
  transaction::commit();
}
```

A Figura 6.4 ilustra a manipulação de dados do Objectstore.

6.3 POSTGRES

O sistema Postgres é um protótipo desenvolvido na universidade da California, Berkeley. Foi construído em linguagem C e teve como referência o Ingress (banco de dados relacional).

O Postgres é baseado em um modelo relacional estendido, oferecendo objetos, OID, objetos compostos, herança múltipla, versões, dados históricos (sendo esta uma das principais características do banco de dados em questão) e uma linguagem de consulta denominada Postquel.

A declaração de dados no Postgres tem como base o modelo relacional. O sistema oferece um tipo abstrato de dados, para que se possa definir um novo tipo de base de dados.

```
create pessoa( nome = char[25],
               Endereco = char[30],
               RG = char[15]
key(RG))
```

A Figura 6.5 ilustra a declaração de dados do Postgres.

Para a manipulação de dados, o sistema Postgres possui uma linguagem de consulta derivada do Quel, linguagem de consulta do banco de dados relacional antecessor (Ingres).

```
reatrieve (A.nome, M.nota)
where A.nome = "Eugenio Akihiro Nassu"
```

A Figura 6.6 ilustra a manipulação de dados do Postgres.

6.4 JASMINE

O sistema Jasmine foi criado no Japão pela empresa Fujitsu. Inicialmente o gerenciador de objetos foi implementado como um tipo de “casca” de um banco de dados relacional não-normalizado. Atualmente o Jasmine é comercializado pela Computer Associates, sendo um dos primeiros BDOO a ser comercializado por uma empresa de software de grande porte.

Para definição e manipulação de dados o Jasmine utiliza a linguagem ODQL (Object Database Query Language). A ODQL pode ser usada tanto como linguagem de consulta, como pode ser embutida em aplicativos escritos em C e C++.

O Jasmine possui recursos de herança múltipla. Sempre que uma classe é criada, existe a necessidade de informar a qual família a mesma pertence. A unicidade do nome da classe dentro de uma família é um fator fundamental.

```
DefineClass demoCF::Pessoas
  description: "informação sobre uma pessoa"
{
  maxInstanceSize: 4;
  class:
    Integer   proxNumPessoa default:0;
  Instance:
    Integer     NumPessoa   unique;
    String      Nome        mandatory;
    Date        DataNasc;
    Bag<string> Telefones   default:Bag{};
    Integer     Idade();
}
```

A Figura 6.7 ilustra a declaração de dados do Jasmine.

Consultas no sistema Jasmine retornam normalmente retornam valores que são atribuídos a uma variável, o que lembra o cursor de variável da linguagem SQL.

```
//Cria um novo objeto
Pessoas p;
p = Pessoas.new( Nome := "Romário" );

//Modifica um objeto existente
Pessoas pp;
pp = Pessoas from Pessoas where Pessoas.NumPessoa = 111
pp.Nome = "Novo Nome!"

//Exclui um objeto existente
Pessoas pp;
pp = Pessoas from Pessoas where Pessoas.NumPessoa = 222
pp.delete();
```

A Figura 6.8 ilustra a manipulação de dados do Jasmine.

7 CONCLUSÃO

Com o crescimento da utilização dos conceitos OO para desenvolvimento de aplicações de alto nível, torna-se cada vez mais necessário que se utilize uma forma de armazenamento em que os objetos possam ser modelados e armazenados sem que haja a necessidade de transformação dos mesmos.

As necessidades de funcionalidade e desempenho da nova dimensão da computação exigem bancos de dados que transcendam os limites do modelo relacional. Com a utilização dos conceitos de OO é possível uma representação realista do mundo dos dados complexos.

Ao estudar BDOO percebe-se que sua conceituação traz novos recursos antes não existentes em bancos de dados puramente relacionais. Estes recursos surgiram pelo largo uso de linguagens de programação orientada a objetos. Um dos desafios, em face a este contexto de evolução da modelagem de dados sugerido pelos desenvolvedores de bancos orientados a objetos, é a grande quantidade de aplicações estáveis que usam bancos relacionais. Por isso, grande é o esforço em prol de padronizar as linguagens de acesso aos BDOO de forma a difundir seu uso e aplicabilidade.

É errado pensar puramente que o BDOO são os substitutos da tecnologia atual de banco de dados relacionais. Eles estão disponíveis para situações distintas, que devem ser bem medidas para evitar sub aproveitamento de seus detalhes de funcionamento.

Talvez motivado por esta situação que grandes projetos de bancos de dados relacionais já adotaram alguns conceitos da orientação a objetos como a herança, tipos abstratos de dados e funções personalizadas. Estes bancos, conhecidamente, como

objeto-relacional são cada vez mais usados em aplicações diárias como alternativas mais estáveis, uma vez que boa parte dos projetos de BDOO não estão largamente difundidos.

Diante disso pode-se afirmar que o trabalho contribui muito para o entendimento do funcionamento dos BDOO, esclarecendo dúvidas e fixando novos conceitos sobre esta tecnologia.

REFERÊNCIAS BIBLIOGRÁFICAS

[SET05] Setzer, Valdemar W. - Banco de Dados: aprenda o que são, melhore seu conhecimento, construa os seus / Valdemar W. Setzer e Flavio Soares Corrêa da Silva – 1º edição – São Paulo: Edgard Blucher, 2005.

[LIM09] Lima, Adilson da Silva, 1959 – UML 2.2: do requisito à solução / Adilson da Silva Lima. – 4. Ed ver. e atual – São Paulo: Érica, 2009.

[NAU99] Nassu, Euginio A. - Banco de dados orientados a objetos / Eugênio A. Nassu, Valdemar W. Setzer – São Paulo: Blucher, 1999

[TON03] Tonsig Sérgio Luiz – Engenharia de software / Sérgio Luiz Tonsig – São Paulo: Futura 2003

[KOR93] Korth, Henry F. – Sistemas de bancos de dados / Henry F. Korth, Abrahem Silberschatz; tradução Mauricio Heihachiro Galvan Abe: revisão técnica Sílvio Carmo Palmieri, - 2° Ed. – São Paulo: MAKRON Books, 1993.

[KHO94] Khoshafian, Setrag – Banco de Dados Orientado a Objeto / Setrag Khoshafian; tradução de Tryte Informática. – Rio de Janeiro: Infobook, 1994