RICARDO MARIANO GALO BORGES

BANCO DE DADOS ORIENTADOS A OBJETOS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo, Educação Continuada em Engenharia para obtenção do título de especialista em Engenharia de Software.

Área de Concentração :
Engenharia de Software

Orientador :
Jorge Rady de Almeida Júnior

São Paulo
2002

AGRADECIMENTOS

A paciência e dedicação ao orientador Jorge Rady, que revisou e fez os comentários necessários para execução deste trabalho.

A minha esposa Andréia, que deu o apoio necessário para que eu tivesse tempo para me dedicar a pesquisa e redação.

RESUMO

O trabalho apresenta os principais pontos que caracterizam os SGBD Orientados a Objetos, tendo como base o Manifesto do Sistema de Gerenciamento de Dados. São explicadas todas as características consideradas obrigatórias. Também é feita uma comparação entre os SGBD Relacionais, Objetos-Relacionais e Orientados a objetos, mostrando a evolução gradativa do SGBD até o momento. Uma análise do mercado para os SGBD Orientados a Objetos é apresentada de forma a indicar os pontos que favorecem a adoção destes sistemas. A linguagem OQL também é apresentada e também é discutido os problemas em definir um padrão definitivo sobre este tipo de linguagem, os grupos ODMG, OMG e SQL3 tem trabalhado em definir a padronização da linguagem criada para manipular objetos.

ABSTRACT

This dissertation introduces the main points of the OODBMS technology, based on DBMS Manifesto, it explores all the issue that are required. The RDBMS, the ORDBMS and the OOBDMS are compared, showing the gradual evolution of the DBMS until this time. An analysis of database market for OODBMS has been showing, and the do's and don'ts of this technologic is presented itself. The language OQL and compared with SQL language, and how the organization ODMG, OMG and SQL3 have been working to define the standard language.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Os bancos de dados orientados a objetos surgem como uma evolução natural dos bancos de dados relacionais para complementar as necessidades das linguagens de programação orientadas a objeto. A complexidade dos sistemas desenvolvidos atualmente, as necessidades de novos tipos de dados e os requisitos do comércio eletrônico e da Internet fez surgir a necessidade de um banco de dados de terceira geração. O banco de dados relacional, que surgiu na década de 70, revolucionou os sistemas da época, mantém até o momento a preferência devido à flexibilidade e capacidade de processamento apresentadas, mas encontra barreiras que limitam as características dos sistemas e tomam tempo de programação devido a adaptações necessárias entre a linguagem orientada a objeto e o banco de dados relacional.

As características e as promessas anunciadas pelas comunidades e fornecedores dos Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos – SGBDOO são as mesmas pelos fornecedores das linguagens orientadas a objetos: menos abstração de dados, maior rapidez no desenvolvimento e implementação de aplicações, reutilização, etc. Da mesma forma, as dificuldades encontradas pelas empresas que adotam os SGBDOO são praticamente as mesmas dos sistemas desenvolvidos em linguagens orientadas a objeto: dificuldade em encontrar pessoal com conhecimento necessário, dificuldades de implementação e desenvolvimento, desempenho insatisfatório dos sistemas.

O objetivo deste trabalho é apresentar as características dos bancos de dados orientados a objeto, destacando suas principais vantagens e aplicabilidade dos mesmos. Através da comparação com o banco de dados relacional e com o banco de dados objeto relacional, exemplificar suas principais características e modos de utilização. Os Sistemas Gerenciadores de Bancos de Dados Objeto-Relacionais – SGDBOR apresentam características comuns aos Sistemas Gerenciadores de Bancos de Dados Relacionais – SGBDR e aos SGBDOO e tiveram aceitação mais rápida do mercado, apesar de não atenderem a todas as exigências do mundo da orientação a objetos.

1.1 ESTRUTURA DO TRABALHO

O capítulo 2 apresenta as características que compõe um sistema de gerenciamento de dados. Descreve a cronologia dos sistemas de gerenciamento, situando e descrevendo a evolução dos sistemas, inicialmente adotado o conceito de banco de dados relacionais, evoluindo para objeto relacional e como orientado a objetos.

No capítulo 3 são apresentados dados estatísticos e previsões sobre o crescimento da participação dos Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos no mercado de software. Também são descritas as tendências de mercado e de desenvolvimento que devem favorecer o mercado de Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos e facilitar a participação deste no mercado.

O capítulo 4 descreve os principais pontos das definições apresentadas pelo Object Database Management Group - ODMG para os Sistemas de Gerenciamento de Dados. O ODMG é um consórcio de fornecedores de Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos cuja missão é a de definir padrões e tecnologia dos sistemas de bases de dados orientadas a objetos.

No capítulo 5 apresentam-se características dos diferentes sistemas de gerenciamento de dados, bem como são feitas comparações entre eles, incluindo-se comentários sobre qual tecnologia se adapta às diferentes necessidades.

O capítulo 6 baseia-se no Manifesto dos Sistemas de Gerenciamento de Banco de Dados, apresentando as características obrigatórias a um sistema para que ele seja considerado um Sistema Gerenciador de Bancos de Dados Orientados a Objetos. Estas características são: Objetos Complexos, Identidade de Objetos, Encapsulamento, Tipos e Classes, Hierarquia de tipos e classes, overriding, overloading e late binding, perfeição computacional, extensibilidade, persistência, gerenciamento de alocação secundária, concorrência e facilidade de consultas Ad-Hoc.

O capitulo 7 descreve a capacidade que mais caracteriza os Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos, que é a capacidade de tornar um objeto persistente e ser compartilhado entre várias sessões.

O capítulo 8 descreve como os Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos devem tratar importantes operações de gerenciadores de bases de dados, que garantem usabilidade e determinam a maturidade do sistema. Estas operações constam transações, concorrência, recuperação e controle de versão.

O capítulo 9 apresenta como as consultas que manipulam objetos estão sendo desenvolvidas e padronizadas segundo os consórcios ODMG, OMG e SQL3

No capítulo 10 são descritos alguns dos principais comandos OQL (Object Query Language) e a sintaxe correspondente em SQL

O capítulo 11 apresenta uma tabela com alguns sistemas de gerenciamento orientado a objeto e quais as principais características destes sistemas.

Finalmente, o capítulo 12 apresenta as conclusões finais deste trabalho.

2. TECNOLOGIA DE BANCOS DE DADOS

O conceito de banco de dados surgiu com a definição de que é o componente da Tecnologia da Informação voltado para o armazenamento da informação, cuja *estrutura* e *comportamento* deve propiciar esse armazenamento de forma *persistente* e *consistente*.

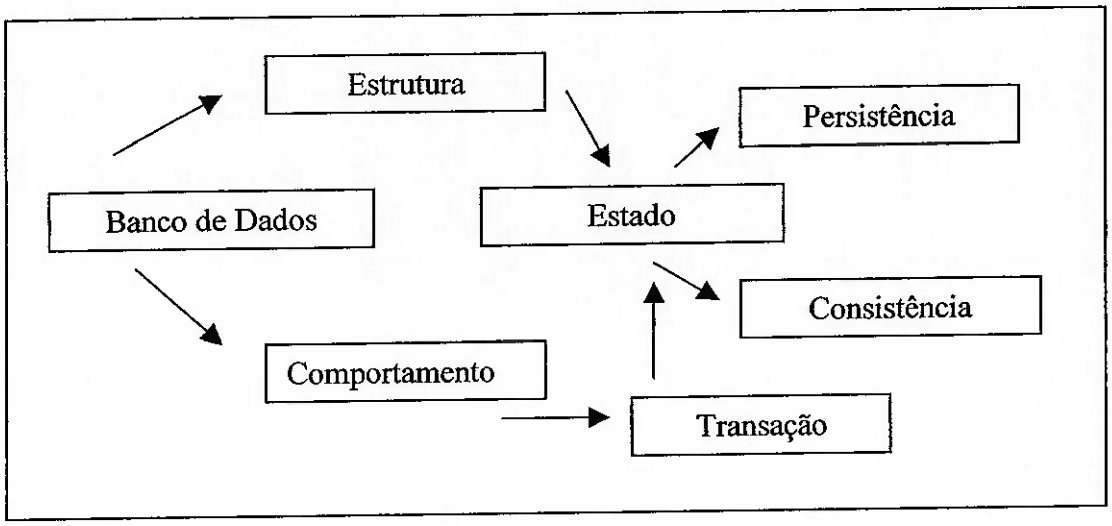

Figura 1 – Componentes de um banco de dados

As definições dos componentes são as seguintes :

1) **Estrutura:** Reflete a estrutura dos objetos ao qual o modelo representa
2) **Comportamento:** Reflete o comportamento dos objetos do modelo correspondente
3) **Transação:** Conjunto de operações que levam o banco de dados de um estado consistente a outro estado consistente.
4) **Estado:** Conjunto de dados armazenado do banco de dado num determinado momento do tempo.
5) **Consistência:** Cada estado do banco de dados deve corresponder a um estado do modelo
6) **Persistência:** Capacidade dos dados continuarem a existir após o término da execução das transações

Segundo Neto e Pereira Neto, entende-se como modelo a representação de algo a ser representando a partir do mundo real.

A Primeira Geração de banco de dados surgiu na década de 70 e foi classificado como banco de dados hierárquico e de rede. Tais bases de dados eram utilizadas para oferecer funções de gerenciamento de dados em um sistema com definição de dados e linguagem de manipulação de dados para coleções de registros. Na década de 80, a primeira geração de banco de dados foi substituída pelos sistemas de gerenciamento de dados relacionais, o qual foi considerado a Segunda Geração dos sistemas de bancos de dados. Os bancos de dados relacionais foram considerados uma evolução em relação aos bancos de dados hierárquicos, pois permitiam acesso aos dados através de linguagens de manipulação que não eram orientadas a procedimentos e propiciam um significativo grau de independência de dados.

Para a segunda geração de Sistema Gerenciadores de Banco de Dados - SGBD, o principal objetivo era atender às necessidades das aplicações classificadas como "business data processing", e atualmente muitos pesquisadores indicam que eles são inadequados para um grande número de novas aplicações. Aplicações do tipo CAD, CASE e hipertexto são exemplos de aplicações que necessitam utilizar um SGBD com capacidades específicas. Vários esforços têm sido feitos na construção de protótipos com funções avançadas, e também os principais fornecedores de bancos de dados estão trabalhando para melhorar as funções da chamada segunda geração de banco de dados. Com todos estes esforços e necessidades do mercado, surgiu o que está sendo considerada a terceira geração de sistemas de bancos de dados. Apesar de vários distribuidores e empresas já estarem utilizando as linguagens orientadas a objetos, os bancos de dados orientados a objetos ainda não conquistaram a unanimidade no mercado e ainda apresentam vários pontos que necessitam de e definição e padronização.

A necessidade do bancos de dados orientados a objetos já é um fato. Considere um exemplo descrito no Manifesto dos Bancos de Dados de Terceira Geração, uma aplicação para auxiliar na publicação de jornais, na qual o usuário deseja definir o lay-out do jornal e então imprimi-lo. Esta aplicação necessita manipular e armazenar em um banco de dados textos, gráficos, ícones e um incontável número de elementos encontrados na maioria dos hipertextos, comuns na Internet. Já existem sistemas hoje que atendem estas necessidades utilizando linguagens orientadas a objetos e bancos de dados relacionais, mas gasta-se tempo em convertendo o modelo de objetos para o modelo relacional e a utilização de artimanhas de programação para atender às necessidades dos sistemas e a falta de opções do banco de dados podem provocar um tempo maior de programação e problemas de desempenho do sistema.

Outro exemplo envolve aplicações de apólices de seguros. Estas aplicações envolvem dados tradicionais como nome e os dados de cada pessoa assegurada por uma apólice. Sistemas assim seriam mais completos se pudessem armazenar fotos relacionadas à vistoria resultante de alguma alegação, ou guardar os originais de fax enviados. É sempre complicado guardar este tipo de informação em banco de dados relacionais.

Os primeiros desenvolvimentos e pesquisas relacionadas com banco de dados orientados a objetos datam da década de 1970 e tiveram um significativo aumento durante a década de 1980, sendo que os primeiros produtos comerciais datam do início da década de 1990. Atualmente existem várias empresas com diversos produtos disponíveis, trabalhando para colocar no mercado produtos cada vez mais robustos e que atendam às necessidades do usuário.

Os bancos de dados orientados a objeto têm adeptos fiéis, como aplicativos que envolvam comércio eletrônico, gerenciamento de dados para engenharia de produtos, e com propósitos especiais para áreas como segurança e medicina.

O SGBDOO está seguindo uma linha de maturação muito similar aos SGBDR. A Figura 2 descreve a evolução da tecnologia do SGBDOO. À esquerda, há as linguagens orientadas a objetos que evoluíram do ponto de propiciar persistência de objetos na aplicação para persistência de objetos entre sessões de usuários. Deve haver as características mínimas para banco de dados, como controle de concorrência, transações, recuperação, etc. Na parte central da figura estão representados os produtos suporte para a maioria das características de um banco de dados tradicional. Finalmente, os produtos de banco de dados passam a possuir semânticas para declaração de objetos e a possibilidade de reduzir os trabalhos de desenvolvimento, conforme representado na arte direita da figura.

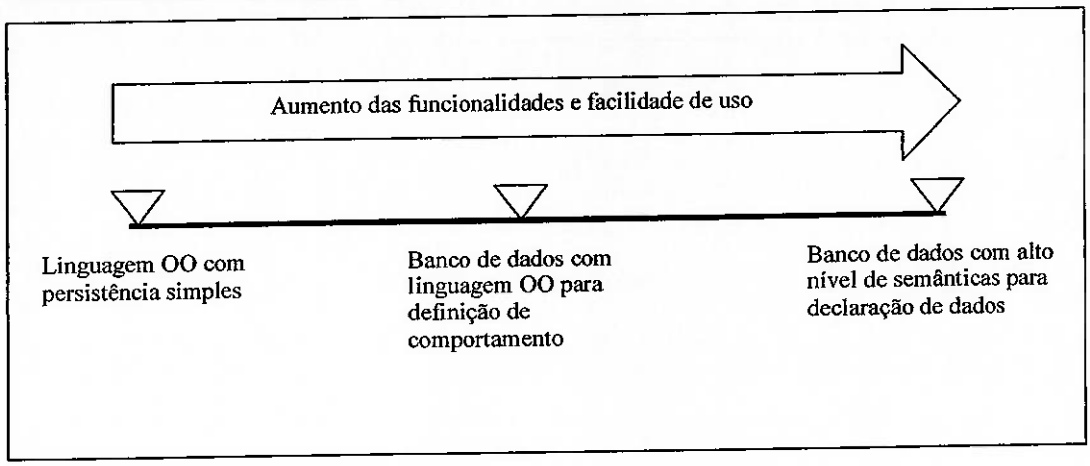

Figura 2 – A Evolução dos SGBDOO

O próximo estágio de evolução é mais difícil. Quanto mais recursos o banco de dados propícia para o usuário, menor o esforço requerido para sua programação. Atualmente o SGBDOO fornece um grande número de funções de baixo nível com o propósito de otimizar o acesso aos dados. O ônus é totalmente do desenvolvimento, pois tem que determinar como utilizar estas funções para otimizar sua aplicação.
Um guia genérico para medir a maturidade de um banco de dados é o grau de quais funções, tais como otimização de acesso ao banco, regras de integridade, esquema e migração de banco de dados, arquivo, cópia de segurança e recuperação de operação podem ser moldadas pelos usuários usando comandos declarativos de alto nível. Atualmente, a maioria dos bancos de dados orientados a objeto requer que os desenvolvedores escrevem códigos para ter acesso a estas funções.

Outro sinal de maturidade desta nova tecnologia é o estabelecimento de grupos para padronizar os diferentes aspectos da tecnologia. As empresas estão preocupadas e interessadas que sejam desenvolvidos padrões para banco de dados orientados a objetos. Por exemplo, o **Object Management Group** (OMG) é um associação sem fins lucrativos onde o objetivo é promover uma série de padrões para viabilizar a interoperabilidade entre os componentes de softwares. Interfaces são definidas em área de comunicações (Object Request Broker), bancos de dados orientados a objetos, interfaces de usuário orientados a objetos, etc. Uma interface de programação de aplicativos (API) vem sendo desenvolvida (por ODMG, Object Database Management Group, um grupo de fornecedores de SGBDOO) deste modo permitindo portabilidade de aplicativos através do SGDBOO. Outros padrões, como X3H7, um comitê técnico sobre X3, formado para definir SGBDOO padrões em áreas como modelo de objetos e extensões de objetos para SQL. Como a OMG está trabalhando na padronização de bancos de dados orientados a objetos e sobre a definição do OQL (Object Query Language), uma extensão do SQL para manipulação de objetos, serão mais detalhados mais adiante neste trabalho.

Continuamente, os fornecedores de SGBDOO estão adicionando cada vez mais características aos seus produtos para propiciar características que o mercado espera de sistemas maduros de gerenciamento de bancos de dados. Esta evolução nos leva para o meio da escala evolucionária mostrada na Figura 2.

3. MERCADO PARA OS SGBDOO

O IDC, uma empresa especializada em realizar pesquisas de mercado, reportou no ano de 1999 uma receita de US$ 11,1 bilhões para bancos de dados relacionais e orientados a objetos, mas somente US$ 211 milhões estavam relacionados com SGBDOO. Até 2004, segundo um relatório de 2000, o instituto IDC previa um crescimento anual a taxas de 18,2 % para bancos de dados relacionais e somente 12,2 % para SGBDOO

Os SGBDOO não se tornarão os maiores participantes do mercado de banco de dados, mas existem nichos de mercado específicos que utilizam tal tecnologia SGBDOO. Isto porque a tecnologia de orientação a objetos propicia manipular objetos complexos muito bem, podendo-se então também manipular relacionamentos complexos mais eficientemente. Bancos de dados orientados a objetos são populares para o uso em inteligência artificial e em aplicações CAD/CAM, as quais envolvem relacionamentos complexos de dados. Aplicações CAD/CAM também utilizam tipos de dados multimídia, os quais os bancos de dados orientados a objetos manipulam com maior eficiência. A tecnologia de bancos de dados orientados a objetos também está sendo utilizada na modelagem de aplicativos financeiros, tais com títulos e derivados. A orientação a objeto também provê características como herança, permitindo a modelagem de novos instrumentos mais rápida e facilmente, o que ajuda a estas empresas a lançar produtos no mercado mais rapidamente.

A mais significante característica da tecnologia dos bancos de dados orientados a objetos é que ele combina a programação orientada a objetos com tecnologia de bancos de dados tradicionais para propiciar um sistema integrado de desenvolvimento de aplicações. Há muitas vantagens em incluir as definições de propriedades com a definição de dados. As definições de operações facilitam o fato de não se criar dependências a um particular banco, e as definições de dados podem ser estendidas para suportar dados complexos como multimídias, pela definição de novas classes que tenham operações para suportar os novos tipos de informação.

Outros pontos fortes da modelagem orientada a objetos são bem conhecidos. Por exemplo, a herança permite desenvolver soluções para problemas complexos e pela definição de novos objetos e pela utilização de objetos pré-definidos. Polimorfismo e associação dinâmica permitem definir uma operação para um objeto e então compartilhar as especificações das operações com outros objetos. Estes objetos podem também estender suas operações para propiciar comportamentos que são únicos para aqueles objetos. A associação dinâmica determina em tempo de execução, qual destas operações está sendo executada, dependendo da classe do objeto que solicitou e execução desta operação. O polimorfismo e associação dinâmica são características poderosas da orientação a objetos que permitem compor objetos para promover soluções sem ter escrito código que é específico para cada objeto. Todas estas capacidades reunidas propiciam vantagens significativas para o desenvolvimento de aplicações de bancos de dados.

Apesar de algumas dificuldades e a complexidade de um sistema de banco de dados orientado a objetos, há algumas tendências indicando que esta tecnologia terá bastante destaque no futuro.

3.1 TENDÊNCIAS DOS PRODUTOS SGBDOO

As principais tendências que se observam para os Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos são :

- **Maturidade dos Produtos :** Para muitas aplicações, a barreira para um SGDB é definida muito alta. Os clientes esperam características como Recuperação Automática, Failover Automático, e replicação de dados. Mais que isso, eles querem comprovação de desempenho e confiabilidade. No início dos SGBDOO, os produtos simplesmente não alcançavam esta barreira. Clientes tinham a percepção de que escolhendo um SGBDOO, ocorria uma troca de ganhos de desempenho com a perda de confiança no produto. Entretanto, a maioria dos produtos disponíveis no mercados já se encontra na sua quinta geração é oferecem características SGDB e bom tratamento de registros. Os SGBDOO estão muito próximos dos SGBDR em relação às características que um SGDB oferece, eliminando a maiorias das objeções em usar SGBDOO.
- **Definições de Padrões :** Umas das objeções em adquirir produtos SGBDOO é que os mesmos possuem interfaces proprietárias, sujeitando os clientes a um produto em particular. Muitos fornecedores então tem se sujeitando aos padrões do Object Database Management Group (ODMG). Estes padrões cobrem interfaces C++, Java, e SmalTalk, como também suportam as ODL (Object Definition Language), OQL (Object Query Language) e OIF (Object Interchange Format). Desta forma aceitando estes padrões, muitos clientes sentem-se mais seguros em realizar investimentos no aprendizado do SGBDOO, pois poderá ser aplicado para diferentes produtos. O OIF ainda permite que organizações possam mover objetos entre diferentes produtos SGDBOO, e entre SGBDOO e SGBDR.

3.2 TENDÊNCIA DO DESENVOLVIMENTO DE SOFTWARES

Uma das barreiras para adoção de SGBDOO era que o desenvolvimento de aplicações em SGBDOO era diferente de um desenvolvimento de uma aplicação típica. Entretanto, tendências de desenvolvimento de programas têm tornado o modelo de desenvolvimento para SGBDOO mais atrativo.

- **Aplicações Clientes Thin** : Muitas aplicações hoje utilizam um Cliente Thin, ou cliente leve, um servidor de aplicações e um servidor de banco de dados para propiciar eficiência, aumentar a reutilização e aumentar a escalabilidade. Um dos problemas históricos com aplicações envolvendo SGBDOO é que era necessário a instalação e manutenção de pesados runtimes nos clientes. Também era necessária a execução de quase toda a lógica das aplicações nos clientes. Com aplicações que utilizam Thin Client, só o servidor de aplicativos contém os runtimes do SGBDOO. Um runtime muito pesado, neste caso, não é problema, pois um servidor de aplicativos normalmente roda em máquinas bem poderosas e compartilhados por diversos clientes. A execução da lógicas da aplicação não é um problema porque este é o propósito dos servidores de aplicativos.
- **Aplicações via Web Interativas** : Alguns modelos de dados favorecem ainda mais os SGBDOO. Eles tipicamente substituem um intermediário com interação direta entre o navegador do usuário e os sistemas de informações da empresa. Tais aplicações incluem self-service de empregados/cliente, publicações de informações, e gerenciamento de cadeia de suprimentos. Tais serviços tipicamente requerem o gerenciamento de uma grande quantidade de informações sobre os usuários do sistema e o contexto dos processos de negócios envolvidos. Esta informação compõe uma complexa rede de relacionamentos ideal para um SGBDOO.

- **Aplicações em Java :** Também um caso especial de aplicações Thin Client, a natureza da linguagem de programação Java favorece ainda mais os SGBDOO. As características da linguagem Java enfatizam conceitos da orientação a objetos, como encapsulamento, interfaces, e polimorfismo, bem como propiciam a minimização de tarefas de baixo nível como gerenciamento de memória. Uma associação transparente de linguagens de programação com o SGBDOO atende esta necessidade melhor que a utilização de APIs de baixo nível como JDBC (Java Data Base Connectivity). O endosso da Sun com a associação da linguagem feita pelo ODMG e o suporte para seus padrões em todos os produtos SGBDOO, o torna atração atrativa para armazenar objetos Java. Também, o suporte ao Java tem encorajado as organizações a considerar uma maior utilização de aplicações Java, incluindo aplicativos para Intranet, Internet, etc ... Como as organizações planejam e desenvolvem cada vez mais aplicações que utilizam objetos distribuídos, o papel dos SGBDOO para permitir persistência em objetos distribuídos aumenta.
- **Aplicações Baseadas em Componentes** : Enquanto desenvolvedores utilizam componentes nas interfaces para usuário, há novas tendências em direção à utilização de componentes lógicos de negócios que rodam em servidores de aplicativos. Com estes componentes, desenvolvedores podem reutilizar unidades lógicas de negócios em múltiplas aplicações e distribuir a lógica de negócios entre servidores de aplicativos. Como estes componentes codificam lógicas de negócios, eles requerem complexas redes de objetos para capturar o relacionamento entre processos de negócios e recursos da empresa. Além do mais, eles requerem manutenção dos estados destas redes através de componentes rodando e através do tempo. A necessidade de uma manutenção de uma complexa rede de objetos e de estados distribuídos tornam o SGBDOO uma solução atrativa.

3.3 TENDÊNCIAS DOS NEGÓCIOS

Além das tendências dos produtos SGDBOO e das tendências do desenvolvimento de softwares, há também tendências de negócios que favorecem a adoção dos SGBDOO.

- **Complexidade e Competição** : Companhias utilizam cada vez mais as informações tecnológicas como vantagens competitivas. Elas têm que utilizar os avanços tecnológicos para gerenciar complexidade e sobreviver à competitividade. Uma crescente tendência de negócios indica que complexas redes de objetos têm um papel em permitir negócios de tecnologia. Esta tendência é a integração de processos de negócios entre departamentos funcionais e até mesmo entre companhias. Por exemplo, o processo de negócios de "Customer Care" necessita incluir o desenvolvimento de produtos, montagem, suporte técnico, compatibilidade, e vendas. O processo de planejamento de recursos precisa ser incluído em todos os níveis da cadeia de suprimento. A mais significativa extensão disto é em relação ao "único ponto de contato" de um cliente em uma empresa. Deste modo, é oferecido para o cliente um único ponto de contato para todas as operações e serviços que o cliente utilize. Serviços de telecomunicações são um ótimo exemplo disto. Um cliente pode ser beneficiado por único ponto de contato para ativação de serviços, gerenciamento de contas, e suporte para ligações locais, de longas distâncias, celulares, correio de voz, e serviços de Internet. A chave para alcançar este tipo de integração é rastreando o relacionamento entre os vários clientes, produtos e serviços, organizações, fornecedores e regulamentações governamentais. Somente bancos de dados orientados a objetos podem prover suporte para a integração em grande escala.

- **Aumento do Uso da Internet :** Negócios têm tornado a Internet um meio para gerenciar complexidade e aumentar a competitividade porque permite que sejam feitas novas conexões. A Internet conecta pessoas com recursos das informações. Também conecta as companhias de modo que possam cooperar com mais eficiência. A barreira para estabilizar as conexões para gerenciar complexidade e aumentar competitividade é monitorar os relacionamentos resultantes. Estes relacionamentos ocorrem nas aplicações e na infraestrutura da Internet. Os switches e roteadores que fazem com que a rede necessite gerenciar um grande número de relacionamento de informações como configurações de dados para produzir conexões de rede. Além da infraestrutura de hardware, há um crescimento dos tamanhos dos softwares corporativos que necessitam gerenciar relacionamentos, incluindo serviços de diretórios, proxies, e firewalls. Além do mais, as razões das organizações para construir aplicações para Internet é vincular pessoas, processos, e recursos de forma conjunta, resultando na necessidade de gerenciar relacionamentos cada vez em um nível mais rápido. O foco no relacionamento significa que o SGBDOO irá se tornar cada vez mais presente na infraestrutura da Internet e nas aplicações.

- **Pacotes de Aplicações :** A complexidade para desenvolver e gerenciar aplicações personalizadas tem se tornando intolerável para muitas organizações. Para gerenciar isto, muitas destas organizações têm utilizado pacotes de aplicações de fornecedores como Baan, Oracle, PeopleSoft e SAP para funções como Prestação de Contas, Recursos Humanos, e Planejamento de Recursos. Um grande número de fornecedores de segunda camada de softwares tem fornecido soluções e atacado problemas especiais como automação de força de vendas e planejamento de produtos. Há também um grande número de ISVs (Independent Software Vendor)[1] que constróem pacotes de aplicativos específicos para determinadas aplicações como a área financeira, de pesquisas biomédicas, e fábricas de semicondutores. Como estes pacotes de aplicativos devem atender cadeias de suprimentos e suportar cada vez mais complexos processos de negócios., eles irão precisar de SGBDOO. Um importante requisito dos SGDBOO para muitos pacotes de aplicativos seria a conversão de dados relacionais existentes ou integração com SGBDR.

Segundo o relatório sobre o mercado para bancos de dados orientados a objetos, estas tendências mostram como o mercado está aberto e irá favorecer a entrada e a participação dos bancos SGBDOO.

---

[1] Um ISV escreve e vende programas para rodar em mais de um tipo de hardware e em diferentes sistemas operacionais.

4. DEFINIÇÕES DO SGBDOO SEGUNDO A ODMG

O ODMG define os padrões e as associações entre o SGBDOO e certas linguagens de programação orientadas a objetos. Quando se desenvolve um sistema com linguagem orientada a objetos e utiliza-se bancos de dados relacionais, sempre ocorre o que os programadores chamam de problema de impedância ("impedence mistmatch"), que se trata de uma forma educada de nomear os problemas encontrados quando é necessário mapear os complexos objetos e seus relacionados definidos no modelo de objetos para representações tabulares do banco de dados.

Segundo as especificações do ODMG, existem dois tipos de objetos: o persistente e o temporário. Os objetos temporários são gerenciados pelos sistemas em tempo de execução, seja em SmalTalk, Java ou C++. Objetos temporários já são utilizados por programadores, e os mesmos deixam de existir quando o programa termina. Objetos permanentes são gerenciados pelo SGBDOO, mas o processo é totalmente transparente para o programador. O objeto permanente não deixa de existir quando o programa termina e pode ser utilizado por qualquer outro programa que esteja em execução. Um próximo acesso ao mesmo objeto será mais rápido porque o objeto já se encontra em memória.

O ODMG padronizou a linguagem para definição de objetos (ODL – Object Definition Laguange). Foi padronizada também a linguagem de manipulação dos objetos (OML – Object Manipulation Language). Estas linguagens não fazem distinção entre objetos permanentes ou temporários. A padronização do ODMG não seria completa sem uma linguagem de acesso aos objetos, por isso existe a Linguagem de Consulta a Objetos (OQL – Object Query Language), a qual se baseia na padronização SQL-92 mas oferece suporte a orientação a objetos e integração com linguagens de programação que tenham associações com os padrões definidos pela ODMG. Segundo Jepson, uma consulta OQL pode ser executada a partir da linguagem específica, e incorporar métodos da mesma. Um paralelo aqui pode ser estabelecido com bancos de dados relacionais. Os SGBDR também foram padronizados, bem como foi criada uma linguagem de definição de dados, a DDL (Data Definition Language). Definiu-se também uma linguagem para manipulação de dados, o DML (Data Manipulation Language), e por último foi criado o padrão mundialmente conhecido e aceito pelos desenvolvedores, a linguagem de consulta estruturada, o SQL (Structured Query Language).

O ODMG também define e descreve como os objetos devem se relacionar entre si, e da mesma forma que o banco de dados relacional, os objetos podem se relacionar com cardinalidades um-para-um (1-1), um-para-muitos (1-N), muitos-para-muitos (N-N).

Atualmente a norma do ODMG está na versão 3.0, publicada em 2002. Informações sobre esta publicação podem ser encontradas no website do grupo (*http://www.odmg.org/*).

5. BANCOS RELACIONAIS, OBJETOS-RELACIONAIS E ORIENTADOS A OBJETO

Um sistema de gerenciamento de dados relacional (SGBDR) trata-se de uma coleção de dados organizados como tabelas, com as colunas representando as categorias dos dados e as linhas representando os próprios dados. Por exemplo, uma tabela pode conter informações sobre produtos, com as colunas representando tipos de produtos, clientes, datas de venda e preços.

Usuários podem acessar os dados na ordem em quem foram gravados na tabela ou consultá-los em diferentes modos para examiná-los sob diferentes perspectivas.

Os bancos de dados relacionais são bons para processar grandes quantidades de dados estruturados e alfanuméricos. Por exemplo, empresas usam os dados para manter os registros das transações ou arquivos pessoais.

Entretanto, bancos de dados relacionais são rígidos porque seus dados podem ser estruturados somente em tabelas. E os mesmos têm que trabalhar com um número limitado e simplificado de dados, como os inteiros, e portanto poderá haver problemas manipulando tipos de dados complexos definidos pelos usuários, incluindo também dados multimídia.

Um banco de dados orientado a objetos (SGBDOO) suportam a modelagem e criação de dados como objetos. Usuários podem utilizar novos tipos de dados com SGBDOO simplesmente criando novos objetos.

Com bancos de dados orientados a objetos, a aplicação e o banco de dados utilizam exatamente o mesmo modelo de objeto. Isto não ocorre com bancos de dados relacionais, onde usuários devem utilizar um modelo de objetos para a aplicação e um modelo relacional de dados para o banco de dados. Desenvolvedores têm então que desenvolver procedimentos de mapeamento entre objetos e o modelo relacional.

Programadores para bancos de dados relacionais gastam aproximadamente 25 % do seu tempo de programação realizando o mapeamento entre objetos do programa e do banco de dados, segundo o IDC.

SGBDOO apresenta grandes vantagens para aplicações que necessitam processar relacionamentos complexos entre objetos de dados. Seja um exemplo relativo à modelagem de um Boeing 747 com SGBDOO. O relacionamento entre as partes da aeronave é gerenciando automaticamente pelo banco de dados. Com um banco de dados relacional, seria necessário decompor a aeronave em tabelas e então estabelecer relacionamentos entre as tabelas que são necessárias para construção de uma aeronave.

Os bancos de dados objetos relacionais (SGBDR) são um passo intermediário entre os bancos de dados relacionais e os bancos de dados orientados a objetos. Os SGBDOR atendem partes dos critérios de um banco orientado a objeto. Os SGBDOR representam uma evolução em relação ao SGBDR pois permitem a criação de tipos de dados definidos pelos usuários e implementam características de identidade de objetos. Os SGBDOR apresentam algumas vantagens em relação aos SGBDOO, pois são fornecidos pelos principais distribuidores de bancos de dados relacionais, e os usuários de SGBDR podem preservar seus investimentos e relacionamento com os distribuidores e atualizar seus bancos de dados relacionais para objetos–relacionais. A curva de aprendizado dos bancos SGBDOR é menor porque se aproveita o conhecimento da programação relacional e práticas de desenvolvimento.

Apesar destas facilidades, os SGBDOR não apresentam os mesmos benefícios do SGBDOO. O objetivo principal é suportar tipos de dados complexos mas falha em suportar relacionamento entre objetos complexos. Os bancos de dados objeto–relacionais suportam o SQL3, que é baseado no SQL2, adicionando os conceitos de orientação a objetos. Entretanto, os métodos de utilização de objetos não são os mesmos entre as linguagens de programação como C++, Java e SmalTalk. Isto acaba provocando um desenvolvimento de código extra para resolver as diferenças do modelo de objeto e o modelo relacional. Os bancos de dados objeto–relacionais são uma opção onde existe somente a necessidade de processar tipos de dados complexos. Mas caso a necessidade seja construir aplicações que usam relacionamentos complexos entre objetos, deve ser considerado o uso do banco de dados orientado a objeto.

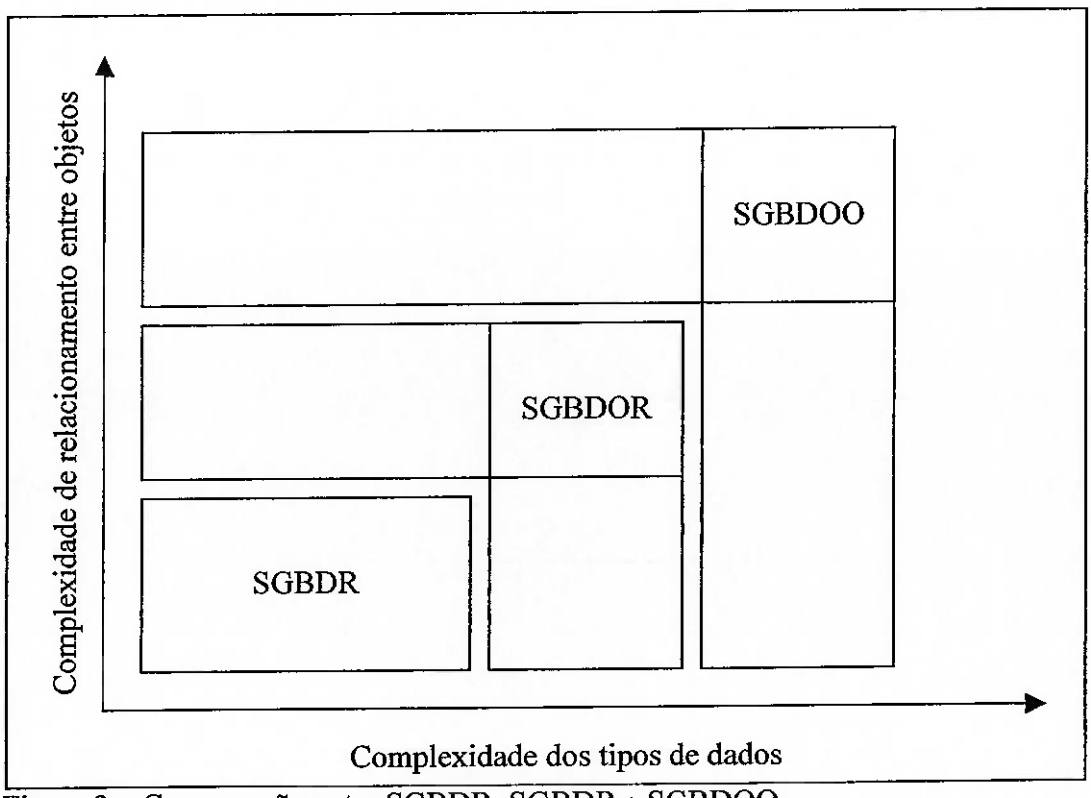

Figura 3 – Comparação entre SGBDR, SGBDR e SGBDOO

Uma tabela em um banco de dados relacional pode ser comparada com uma classe em um banco de dados orientado a objetos, uma linha é similar a uma instância de uma classe com atributos, mas não com comportamentos. Uma coluna de uma tabela é correspondente a um atributo da classe, com a diferença que a coluna pode somente armazenar tipos de dados simples, enquanto um atributo de uma classe pode armazenar dados de qualquer tipo. As classes possuem métodos que são completos em termos computacionais, enquanto bancos de dados relacionais não possuem estas características. Pode-se dizer que algumas características de programação conhecidas como Stored Procedures atendem parcialmente estes requisitos.

Algumas vantagens em se utilizar SGBDOO em comparação a SGBDR utilizando linguagens de programação orientadas a objetos seriam :

- **Composição de Objetos e relacionamentos :** Um objeto em um SGBDOO pode armazenar diversos tipos de dados como também outros objetos. Estes objetos podem ser formados por diversas classes, de tamanhos e definições diversas. Em um banco relacional, uma tabela também pode ter relacionamento com um incontável número de tabelas, com relacionamentos normalizados e ligados através de chaves estrangeiras. Ter um grande número de relacionamentos com tabelas pequenas é sempre um problema devido às consultas que são desenvolvidas e as junções existentes. Os objetos também representam melhor alguns modelos de dados reais e objetos complexos do que o mapeamento em tabelas e linhas. Como este é o objetivo dos SGBDOO, poder manipular objetos complexos, eles apresentam clara superioridade em relação aos SGBDR e até SGBDOR.
- **Hierárquica de classes :** Os dados do mundo real possuem uma hierarquia de características. Utilizando o exemplo de Empregados é mais facilmente descrito em SGBDOO do que em SGBDOR. Um empregado pode ser um gerente ou não, isto pode ser implementando em um SGBDR tendo um campo de identificação ou criando uma nova tabela com uma chave estrangeira para indicar o relacionamento entre gerentes e empregados. Em um banco SGBDOO, a classe Empregado é simplesmente a classe pai da classe de Gerentes.
- **Linguagens de consulta :** O acesso aos dados de um SGBDOO é executado de forma mais transparente do que com um SGBDR, devido a interação entre a linguagem de programação e o banco de dados. Mesmo com esta integração, pode ser feito o acesso aos dados através da linguagem de consulta a objetos.
- **Evitar o "*Impedence Mismatch*" :** é o que ocorre quando uma linguagem de programação orientada a objetos faz uso de um banco de dados relacional. Um tempo é gasto mapeando os objetos em tabelas. Ocorrem também problemas quando tipos de dados da linguagem não possuem correspondentes no banco de dados, pois é necessário implementar a conversão entre eles.

- **Menos modelos de dados :** Um modelo de dados tipicamente deveria modelar entidades e seus relacionamentos, as restrições e as operações que alteram o estado dos dados no sistema. Com o SGBDR não é possível mapear as operações dinâmicas e as regras que mudam o estado de um dado porque isto não está no escopo do banco de dados. As aplicações que utilizam SGBDR possuem o modelo entidade relacionamento para modelar as partes estatísticas do sistema e um modelo separado para operações e comportamentos das entidades na aplicação. Com o SGBDOO não há esta necessidade porque no modelo do banco de dados e no modelo da aplicação, as entidades são objetos no sistema. Uma aplicação inteira pode ser totalmente modelada em um diagrama de UML.

Apesar destas atrativas vantagens, existem pontos que devem ser levantados para que os desenvolvedores estejam atentos :

- **Alterações no modelo :** Em um SGBDR, modificar a estrutura dos objetos no banco de dados é, em sua grande maioria, independente da aplicação. Em uma aplicação baseada em SGBDOO a criação, atualização ou modificação de uma classe persistente significa que as outras classes que interagem com este também devem sofrer alteração. Isto tipicamente significa que as mudanças em um esquema envolvem uma recompilação em todo o sistema. E também a atualização de todas as instâncias de um objeto dentro do banco de dados pode demorar bastante tempo dependendo do tamanho do banco.
- **Dependência de linguagem :** Um banco orientado a objeto tem relação direta com uma linguagem específica de programação via API. Isto significa que os dados de um SGBDOO são somente acessíveis a partir de uma linguagem de programação usando uma API específica. Isto não ocorre com os bancos relacionais, onde existem aplicações desenvolvidas que podem trabalhar com diferentes distribuições de banco de dados relacionais
- **Falta de queries Ad-Hoc :** em um banco de dados relacional, pode-se criar facilmente novas tabelas que em determinada consulta têm relacionamentos com tabelas já existentes para atender necessidades de consultas Ad-Hoc. As consultas que podem ser realizadas em um SGBDOO têm dependência direta com o desenho do sistema.

6. CARACTERÍSTICAS DOS SISTEMAS GERENCIADORES DE BASES DE DADOS ORIENTADAS A OBJETOS

A figura 4 representa as principais características de bancos de dados orientados a objetos.

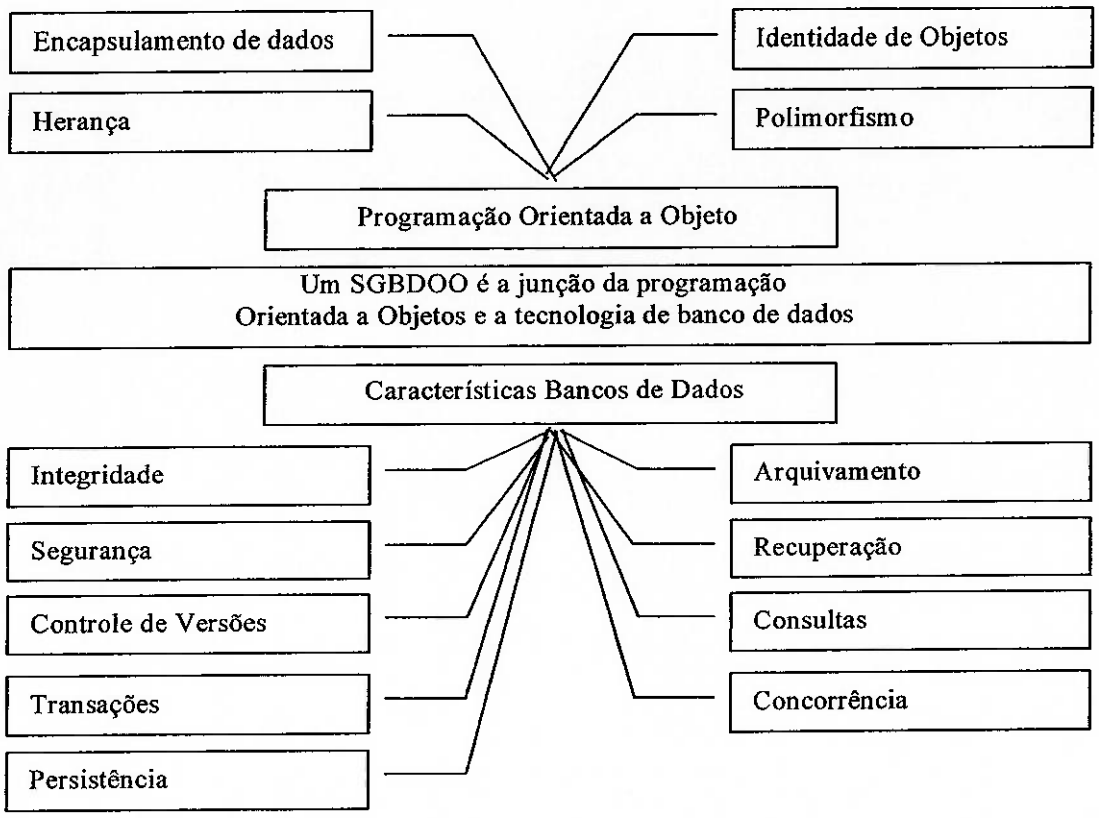

Figura 4 – Rascunho de um SGBDOO

Alguns conceitos relacionados com orientação a objeto necessitam ser claramente definidos para que o banco de dados orientado a objetos seja também melhor utilizado. Podemos definir que orientação a objeto corresponde à organização de sistemas como uma coleção de objetos que integram estruturas de dados e comportamentos. Alguns conceitos básicos e importantes são :

- **Abstração:** é a consideração apenas das propriedades comuns de um conjunto de objetos, omitindo detalhes, utilizada com freqüência na definição de valores similares e na formação de um tipo a partir de outro, em diferentes níveis de abstração. O uso de abstração permite a geração de tipos baseados em hierarquias de tipos e de relacionamentos. Os principais conceitos de abstração utilizados no banco de dados são generalização e agregação. A generalização corresponde à associação "é um" onde, a partir de propriedades comuns de diferentes entidades, é criada uma outra entidade. O processo inverso é a especialização. A agregação corresponde a associação "parte de".
- **Objeto:** Os objetos são abstrações de dados do mundo real, com uma interface de nomes de operações e um estado local que permanece oculto. As abstrações de representação e das operações são ambas suportadas no modelo de dados orientados a objetos, ou seja, são incorporadas as noções de estruturas de dados e de comportamento. Um objeto tem um estado interno descrito por atributos que podem apenas ser acessados ou modificados através de operações definidas pelo criador do objeto. Um objeto individual é chamado instância ou ocorrência de objeto. A parte estrutural de um objeto é similar à noção de entidade no modelo Entidade-Relacionamento.

Um banco de dados orientado a objetos é o resultado da combinação dos princípios da programação orientada a objetos com os princípios do gerenciamento de dados. Os conceitos da programação orientada a objetos, tais como encapsulamento, polimorfismo e herança são reforçados com os conceitos de gerenciamento de dados, com a capacidade de controlar transações, concorrência, recuperação e controle de versões.

O Manifesto sobre Bancos de dados orientados a objetos, lista as características obrigatórias de um sistema de gerenciamento de dados antes que ele seja classificado como SGBDOO: Objetos Complexos, Identidade de Objetos, Encapsulamento, Tipos e Classes, Hierarquia de tipos e classes, Overriding, overloading e late binding, perfeição computacional, extensibilidade, persistência, gerenciamento de alocação secundária, concorrência e facilidade de consultas Ad-Hoc.

Cada uma destas características é abaixo detalhada.

1) **Objetos Complexos :** São construídos a partir de objetos simples e da aplicação de construtores sobre eles. Os objetos mais simples são os objetos como integers, chars, cadeias de bytes de qualquer tamanho, tipos boolean e floats. Há vários construtores de objetos complexos, como tuplas, sets, bags, listas e matrizes. Sets são importantes porque é a forma mais natural de representar coleções do mundo real. Tuplas são críticas porque são a maneira natural de representar propriedades de uma entidade. Ambos têm sua importância porque apresentam grande aceitação como construtores de objetos no modelo relacional. Listas e matrizes são importantes porque podem determinar a ordem na qual o evento realmente ocorreu, e também tem importância em muitas aplicações científicas, onde existe a necessidade de matrizes e a ordenação de dados por tempo.

   Os construtores de objetos devem ser ortogonais, ou seja, qualquer construtor pode ser aplicado para qualquer objeto. Os construtores de modelos relacionais não são ortogonais, porque os construtores de sets podem ser somente aplicados em tuplas e as tuplas podem ser somente aplicadas para valores atômicos.

   O suporte a objetos complexos também requer que as operações apropriadas devam existir para manipulá-los, independente de sua composição. Isto é, uma operação em um objeto complexo deve ser propagada para todos seus componentes. Operações adicionais em objetos complexos podem ser implementadas, pelos usuários do sistema.

2) **Identidade de Objetos:** A identidade de objetos sempre existiu em linguagens de programação, mas o conceito é novo em banco de dados. A idéia é a seguinte : em um modelo com identidade de objetos, um objeto tem uma existência a qual é independente de seu valor. Deste modo surgem dois cenários ao se manipular objetos: dois objetos podem ser idênticos (são o mesmo objeto) ou eles podem ser iguais (possuem o mesmo valor). Isto provoca duas implicações que são o de **compartilhamento de objeto** e o de **atualização de objetos**.

- **Compartilhamento de objetos :** em um modelo baseado em identidade, dois objetos podem compartilhar um componente. Considere-se o seguinte exemplo : uma pessoa tem um nome, idade, e um conjunto de crianças. Peter e Susan, ambos têm um filho de 15 anos chamado John. Duas situações isto poderiam provocar: Susan e Peter são pais da mesma criança ou há duas crianças diferentes envolvidas. Em um sistema sem identificação, Peter seria representado como `(PETER, 40, {(JOHN, 15, {})})`, e Susan seria representada como `(SUSAN, 41, {(JOHN, 15, {})})`.

Deste modo, não há como expressar que Susan e Peter são pais da mesma criança. Em um modelo com identificação de objetos, as estruturas podem compartilhar o mesmo objeto `(JOHN, 15, {} )` ou não, identificando assim as duas possibilidades

- **Atualização de objetos :** Assumindo que Susan e Peter são realmente os pais de John. Neste caso, qualquer atualização no filho de Susan será aplicada ao objeto John e, consequentemente, também ao filho de Peter. Em um sistema baseado em valores, os sub-objetos devem ser atualizados separadamente. A identidade de um objeto também é uma primitiva poderosa para manipulação de dados que pode ser o básico de set, tuplas e manipulação recursiva de objetos complexos.

Suportar identificação de objetos implica em oferecer operações tais como alinhamento de objetos, cópia de objetos (cópia superficial ou profunda) e testes para identificações de objetos e igualdade de objetos (igualdade superficial ou profunda). Dois objetos são superficialmente iguais se os atributos dos objetos são idênticos, mas esta igualdade não é recursiva. Dois objetos são profundamente iguais se seus valores são recursivamente iguais. Esta definição examina os valores de um objeto recursivamente. O mesmo conceito vale para a cópia, quando é realizada uma cópia superficial, somente os atributos são copiados, mas não recursivamente. Quando é realizada uma cópia profunda, os atributos recursivos também são copiados.

É claro que se pode simular identificação de objetos em sistemas baseados em valores, introduzindo explicitamente identificadores de objetos. Entretanto, este método transfere a responsabilidade para o usuário do sistema de garantir unicidade dos identificadores de objetos e manter integridade referência.

Vale lembrar que modelos baseados em identificadores são comuns em linguagens de programação: cada objeto manipulado em um programa possui uma identificação e pode ser atualizado. Esta identificação pode ser definida através de uma variável ou de um endereçamento de memória. Mas o conceito é relativamente novo em sistemas relacionais, onde os relacionamentos são baseados em valor.

3) **Encapsulamento :** O conceito de encapsulamento vem (i) da necessidade de distinguir claramente entre a especificação e a implementação de uma operação e (II.) da necessidade de modularização. A modularização é necessária para desenvolvimento de aplicações complexas e implementadas por um grupos de programadores. E também é necessária como uma ferramenta para proteção e autorização.

As duas formas de visualizar o encapsulamento são a visão da linguagem de programação e a adaptação para banco de dados desta visão. A visão da linguagem de programação é a visão original pois o conceito foi originário nesta camada.

O encapsulamento em linguagens de programação tem destaque na definição de tipos de dados abstratos. Um objeto possui uma interface e possui uma implementação. A parte da interface consiste na especificação de um conjunto de operações que podem ser executadas no objeto. Apenas parte do objeto é visível. A parte de implementação possui os dados e os procedimentos. A parte de dados é a representação ou estado de um objeto e a parte de procedimento descreve, em alguma linguagem de programação, a implementação de cada operação.

A implementação do encapsulamento no banco de dados representa que o objeto encapsula os dados e o programa. Em um banco de dados, não está claro se a parte estrutural de um tipo é ou não parte da interface, enquanto que em linguagens de programação, a estrutura de dados é parte da implementação e não da interface.

Considere, por exemplo, um Empregado. Em um sistema relacional, um empregado é representando por algumas tuplas. A consulta é feita utilizando uma linguagem relacional, e posteriormente, um programador escreve um programa para atualizar este empregado quando o mesmo tiver aumento de salário ou for demitido. Isto é geralmente escrito utilizando-se uma linguagem de programação com comandos DML, ou em uma linguagem de programação de quarta geração e são armazenados em programas e não no banco de dados. Deste modo, há uma clara distinção entre programa e dados, e entre a linguagem de consulta (consultas Ad-Hoc) e a linguagem de programação (os programas da aplicação).

Em um sistema orientado a objetos, o empregado pode ser definido como um objeto que possui uma parte de dados (provavelmente muito similar com aquela estrutura definida em um sistema relacional) e uma parte de operações, a qual consiste das operação de aumento de salário e demissão e qualquer outra operação para acessar dados do Empregado. Quando se armazena a estrutura de um empregado, dados e operações são armazenadas no banco de dados.

Assim, há somente um modelo simples para dados e operações, e informações podem ser escondidas. Nenhuma operação, além daquelas especificadas pela interface, pode ser realizada. Esta é a restrição que se impõe sobre operações de atualização e recuperação de dados.

O encapsulamento fornece uma forma de "independência lógica de dados", podendo ser alterada a implementação de um tipo sem mudar qualquer programa que utiliza aquele tipo. Assim, os programas são protegidos de mudanças de implementação em baixas camadas do sistema.

O encapsulamento adequado é obtido quando somente as operações são visíveis e os dados e a implementação das operações são escondidas no objeto.

Entretanto, há alguns casos onde o encapsulamento não é necessário, e o uso do sistema pode ser significativamente simplificado se o sistema permitir que o encapsulamento possa ser violado sobre algumas circunstâncias. Por exemplo, com consultas Ad-Hoc a necessidade de encapsulamento é reduzida desde que questões como sustentabilidade não sejam importantes. Assim mesmo, um mecanismo de encapsulamento deve ser fornecido pelo SGBDOO, mas aparecer somente em casos onde sua execução não é apropriada.

4) **Tipos e Classes :** Está característica é sensível : há duas características principais dos sistemas orientados a objetos, aquelas que suportam a noção de classes e aquelas que suportam noções de tipos. Nesta primeira característica, pode-se citar sistemas como SmalTalk, Gemstone, Vision, e mais genericamente todos os sistemas da família SmalTalk, Orion, Flavors, G-Base, Lore e mais genericamente ainda todos os sistemas derivados do Lisp. Em uma segunda categoria, nós encontramos sistemas como C++, Simula, Trellis/Owl, Vbase e O2.

Um tipo, em um sistema orientado a objetos, resume as características comuns de um conjunto de objetos com as mesmas características. Isto corresponde a noção de um tipo de dados abstrato. Há duas partes: a interface e a implementação (ou implementações). Somente a parte da interface é visível para os usuários do tipo, as implementações de um objeto são vistas somente pelo projetista daquele tipo. A interface consiste de uma lista de operações reunidas com suas marcas (por exemplo, o tipo dos parâmetros de entrada e o tipo de resultados).

A implementação de um tipo consiste de uma parte de dados e uma parte de operações. Na parte de dados, descreve-se a estrutura interna dos dados do objeto. Dependendo do poder do sistema a estrutura da parte de dados pode ser mais ou menos complexas. As partes da operação consistem de procedimentos os quais implementam as operações da parte de interface.

Em linguagens de programação, tipos são ferramentas para aumentar a produtividade dos programadores, garantindo precisão nos programas, forçando o usuário a declarar os tipos das variáveis a expressões manipuladas, as razões do sistema sobre precisão dos programas baseado nas informações digitadas. Se os tipos do sistema forem cuidadosamente projetados, o sistema pode executar verificação de tipos em tempo de compilação, caso contrário isto deve ser postergado do tempo de compilação. Assim, tipos são principalmente usados em momento de compilação para verificar a precisão dos programas. Em geral, em sistemas baseados em tipos, um tipo não é um "cidadão de primeira classe" e tem um estado especial e não pode ser modificado em tempo de execução.

A definição de classe é diferente do tipo. Sua especificação é o mesmo que o tipo, mas é mais no sentido de tempo de execução. A classe contém dois aspectos: um fabricador de objeto e um depósito de objeto. Um fabricador de objeto pode ser usado para criar novos objetos, realizando a operação *new* na classe, ou clonando algumas representações de protótipos de objetos da classe. O depósito de objeto significa que junto a classe está sua extensão. Por exemplo, o conjunto de objetos são as instância de uma classe. O usuário pode manipular o depósito aplicando operações em todos os elementos da classe. Classes não são utilizadas para verificar a precisão de um programa, mas certamente para criar e manipular objetos. Em muitos dos casos em que se emprega o mecanismo da classe, classes são o "cidadão de primeira classe" e, tais como, podem ser manipuladas em tempo de execução, como por exemplo, atualização ou passado como parâmetros. Em muitos dos casos, enquanto se propicia sistemas com aumento de flexibilidade e uniformidade, isto provoca verificação de tipos de tempo de compilação impossível.

É claro, há grandes similaridades entre classes e tipos, nomes têm sido usando com mesmo significado e as diferenças pode ser sutis em alguns sistemas.

5) **Hierarquia de Classes ou Tipos :** a herança tem duas vantagens : é uma ferramenta poderosa para modelagem, por que propicia uma concisa e precisa descrição do mundo real e ajuda na divisão de especificações compartilhadas e implementações na aplicação.

Um exemplo pode ajudar a ilustrar o interesse em ter um sistema que propicie o mecanismo de herança. Supondo que existam Empregados e Estudantes. Cada Empregado tem um nome, uma idade acima de 18 anos e um salário, o mesmo pode morrer, casar e receber salários. Cada Estudante tem uma idade, um nome e um conjunto de notas, o mesmo também pode morrer, casar ou conseguir seu diploma.

Em um sistema relacional, o Administrador de Dados define um relacionamento para o Empregado, um relacionamento para o Estudante, programa as operações de Morte, Casamento e Pagamento para o Empregado, e também programa as operações de Morte, Casamento e Graduação do Estudante. Deste modo, o desenvolvedor teve que escrever 6 programas.

Em um sistema orientando a objetos, usando a propriedade de herança, é possível definir que Estudantes e Empregados são Pessoas, pois os mesmos possuem atributos em comum e outros que são específicos. Pode ser criado um tipo nomeado Pessoa, que possui os atributos Nome e Idade e serem escritas as operações de Morte e Casamento para este tipo. Assim, pode-se declarar que os Empregados são um tipo especial de Pessoa, que herdou os atributos e as operações, e ainda possui atributos adicionais como Salário e uma operação Pagamento. Da mesma forma, é declarado que o Estudante é um tipo especial de Pessoa, com um atributo Conjunto-de-Notas e uma operação especial nomeada Graduação. Desta forma, temos uma estrutura melhor e uma descrição mais concisa de um esquema (redução das especificações) tendo sido necessário escrever somente 4 programas (redução da implementação). A herança também auxilia a reutilização do código, porque todo programa está em um nível que o maior número de objetos podem compartilhar.

6) **Overrinding, Overloading e Late Binding :** Em contraste com o exemplo anterior, há casos onde se faz necessário utilizar o mesmo nome para operações diferentes. Considere, por exemplo, a operação Display: ela recebe um objeto como argumento e mostra o objeto na tela. Dependendo do tipo do objeto, podemos querer utilizar diferentes mecanismos para mostrá-lo na tela. Se o objeto é uma figura, pode-se querer que a mesma apareça na tela. Se o objeto é uma Pessoa, pode-se querer que as informações da linha sejam mostradas. Considere o problema de mostrar um conjunto, onde o tipo de seus membros são desconhecidos em tempo de compilação.

Em uma aplicação que use o sistema convencional, há três operações: Display-Pessoal, Display-Figura e Display-Gráfico. O programador irá testar o tipo de cada objeto do conjunto e usar a correspondente operação de DISPLAY. Isto faz com o que o programador tenha a preocupação de verificar todos os tipos de objetos possíveis em um conjunto, programar a operação de Display daquele tipo e usa-lo corretamente.

```
FOR X IN X DO
  BEGIN
    CASE OF TYPE(X)
     PERSON : DISPLAY(X);
     BITMAP : DISPLAY-BITMAP(X);
     GRAPH : DISPLAY-GRAPH(X);
  END;
END
```

Em um sistema orientado a objetos, é definida a operação de Display no nível do tipo de objeto. Deste modo, a operação Display tem um único nome e pode ser utilizada indiferentemente em Gráficos, Pessoal e Figuras. Entretanto, é redefinida a implementação da operação de cada um dos tipos de acordo com cada tipo, isto é chamado de Overriding. Isto resulta em um único nome (Display) reportando 3 diferentes programas, isto é chamado de Overloading. Para mostrar um conjunto de elementos, simplemente é aplicada a operação Display para cada um deles, e deixar que o sistema execute a implementação apropriada que será determinada em tempo de execução.

```
FOR X IN X DO DISPLAY(X)
```

Aqui temos uma vantagem diferenciada: o implementador de tipos irá escrever o mesmo número de programas, mas o programador da aplicação não terá que se preocupar com 3 tipos diferentes de programas. Complementando, o código é mais simples e não há necessidade programar operações do tipo CASE. Finalmente, o código é de manutenção mais fácil pois quando um novo tipo e nova instâncias de um tipo são adicionadas, o programa Display continuará a trabalhar sem modificações (devido ao fato que uma operação de Override mantém o método display para aquele tipo).

Para prover estas novas funcionalidades, o sistema não consegue associar nomes de operações com programas em tempo de compilação. Por esta razão, os nomes das operações devem ser resolvidas em tempo de execução, traduzido em endereçamento de programas. Esta tradução atrasada é chamada de LateBinding.

7) **Perfeição Computacional :** A partir do ponto de vista da linguagem de programacão, esta propriedade é óbvia: simplesmente significa que pode ser expressa qualquer função computacional, usando as funções DML do sistema de banco de dados. Apartir do ponto de vista do banco de dados, isto é uma inovação, uma vez que a linguagem SQL não é completa.

Isto não significa que desenvolvedores de bancos de dados orientados a objetos desenvolvam uma nova linguagem de programação: a perfeição computacional pode ser implementada através de uma conexão com linguagens de programação já existentes.

Note que isto é diferente de ser "completo de recursos", por exemplo, sendo possivel utilizar todos os recursos do sistema (tela e comunicação remota) a partir de dentro da linguagem de programação. Portanto, o sistema, mesmo tendo perfeição computacional pode não ser capaz de executar uma aplicação por completo. Isto, entretanto é mais que um sistema de banco de dados que somente armazena e recupera dados e realiza operações simples em valores atômicos.

8) **Extensão :** O sistema de banco de dados já possui uma série de tipos pré definidos. Estes tipos podem ser utilizados pelos programandores para escrever suas aplicações. Este conjunto de tipos deve ser extensível no seguinte sentido: há uma maneira de definir novos tipos e não há distinção de utilização entre os tipos definidos pelo sistema e os tipos definidos pelo usuário. É claro, haverá uma diferença na maneira que os tipos do sistema e os tipos definidos pelo usuário são suportados. Esta definição de tipos também inclui a definição de operações nos tipos. Repare que os requisitos de encapsulamento implicam que há mecanismos para definição de novos tipos. Este requisito fortacele esta capacidade de que tipos novos devem ter.

Entretanto, não se é requerido que a coleção de construtores de tipos (tuples, sets, lists, etc.) sejam extensíveis.

9) **Persistência :** Este requisito é evidente do ponto de vista do banco de dados, mas recente do ponto de vista da linguagem de programação. Persistência é a habilidade de que o dado continuará existindo após a execução de um processo, para ser reutilizado por outro processo. A persistência deve ser ortogonal, isto quer dizer, para cada objeto, independente de seu tipo, pode ser permitido se tornar persistente. Isto também deve ser implícito: o usuário não deve ter que explicitamente mover ou copiar dados para torná-lo persistente.
10) **Gerenciamento Secundário de Armazenamento :** Esta é uma característica clássica de um sistema de gerencimento de dados. É normalmente suportada através de um conjunto de mecanismos. Isto inclue gerencimento de índices, cluster de dados, buffer de dados, seleção de caminhos de acesso e otimização de consultas.

Nenhuma destas características está vísivel ao usuário: elas são características simplificadas de desempenho. Entretanto, elas são tão críticas em termos de desempenho que sua ausência tornaria o sistema incapaz de executar algumas tarefas (simplemente porque elas iriam tomar muito tempo). O ponto mais importante é que estas características são invisíveis. O programandor da aplicação não deve ter que escrever códigos para manter índices, alocar espaço em disco, ou mover dados do disco para memória principal. Assim, haveria uma clara independência entre o nível lógico e físico do sistema

11) **Concorrência :** Com relação ao gerenciamento da interação de múltiplos usuários concorrentes com o sistema, deve-se oferecer o mesmo nível de serviço que os atuais sistemas de banco de dados propiciam. Isto deverá garantir um harmoniosa coexistência entre os usuários trabalhando simultaneamente com o banco de dados. O sistema deve suportar a noção padrão de atomicidade da seqüência de operações e de controle de compartilhamento. Serialização de operações deveria ser oferecida, embora alternativas menos rigorosas possam ser oferecidas.

12) **Recuperação :** Novamente, o sistema deve prover o mesmo nível de serviço que os atuais sistemas de banco de dados. Portanto, em caso de falha de hardware ou software, o sistema deve realizar a recuperação, como por exemplo, voltar o sistema para algum estado coerente dos dados. Falhas de hardware incluem falhas de processador ou falhas de disco.

13) **Facilidades de Consultas Ad-Hoc :** Um dos principais problemas é propiciar a funcionalidade de linguagens para consultar Ad-Hoc. Não é requirido que isto seja feito na forma de uma linguagem de consulta mas um serviço deve ser estar disponível. O serviço consiste em permitir que o usuário formule simples consultas para um simples banco de dados. A comparação óbvia é com o sistema relacional, e assim o teste é usar um número representativo de consultas relacionais e verificar se eles podem ser determinados com a mesma quantidade de trabalho. Esta facilidade deve ser suportada pela linguagem de manipulação ou um sub-grupo dela.

As consultar devem satisfazer os seguintes critérios : (i) Devem ser de alto nível, ou seja, serem capazes de expressar consultar não triviais consistentemente. Isto implica que deve ser razoavelmente declarativa, ou seja, com enfaze no "*o que*" e não no "*como*". (ii) Deve ser eficiente, ou seja, ao se formular uma query deve haver alguma forma de otimizar tal consulta. (iii) Deve ser independente da aplicação, ou seja, deve funcionar com a mesma sintaxe em diferentes bancos. Este último requisito elimina facilidades específicas de consultas que são dependentes da aplicação, ou a necessidade de escrever operações adicionais para cada tipo definido pelo usuário.

7. PERSISTÊNCIA DE OBJETOS

O termo persistência como foi comentado, é raramente utilizado no contexto de bancos de dados, mas é a características que define a evolução do SGBDOO em relação às linguagens de programação orientada a objetos.

Preferencialmente, o termo usado é banco de dados, que conota o espaço de objeto resiliente, concorrentemente compartilhado. A função de um sistema de gerenciamento de banco de dados é permitir o acesso e a atualização simultâneos de bancos de dados persistentes. A fim de garantir a persistência dos dados a longo prazo, os sistemas de gerenciamento de banco de dados utilizam várias estratégias de recuperação em caso de falhas na transação, no sistema ou no meio.

Há uma relação fundamental entre o compartilhamento e a persistência simultâneos de banco de dados. As atualizações de transações devem persistir, mas como o banco de dados persistente é ao mesmo tempo acessado e atualizado, o sistema de gerenciamento de banco de dados deve preocupar-se com a coerência dos objetos de dados persistentes. Isso normalmente é obtido por meio de estratégias de controle e recuperação concorrentes.

7.1 NÍVEIS DE PERSISTÊNCIA

Os dados manipulados por um banco de dados orientado a objeto podem ser ***transientes*** ou ***persistentes***. Os dados transientes só são válidos dentro de um programa ou transação, eles se perdem quando o programa ou a transação termina. Os dados persistentes, por outro lado, são armazenados fora do contexto de um programa e assim sobrevivem a várias invocações de programas.

Dados persistentes existem nos bancos de dados compartilhados, acessados e atualizados através de transações. Por exemplo, banco de dados pessoais, banco de dados de inventário e banco de dados de vendedores, contas ou itens, todos contêm dados persistentes. No entanto, há vários níveis de persistência. Os objetos menos persistentes são aqueles criados e destruídos em procedimentos. Depois, há os objetos que persistem dentro do espaço de trabalho de uma transação, mas que são invalidados quando a transação termina. As transações são normalmente executadas dentro de uma sessão. O usuário estabelece seu login e define diferentes parâmetros dentro de uma sessão, como caminhos, opções de exibição, janelas, etc. Se o sistema suportar o multiprocessamento, várias transações poderão estar ativas dentro da mesma sessão de usuário ao mesmo tempo. Todas estas transações compartilharão os objetos da sessão. No entanto, quando o usuário terminar a sessão, os objetos da sessão serão invalidados. O único tipo de objeto que persiste através das transações são objetos permanentes normalmente compartilhados por vários usuários. Esses objetos persistem através de transações, instabilizações de sistema e até de meio. Tecnicamente, esses são os objetos recuperáveis do banco de dados.

8. TRANSAÇÕES, CONCORRÊNCIA, RECUPERAÇÃO E CONTROLE DE VERSÃO

8.1 TRANSAÇÃO

Uma transação consiste em um trecho de programa que deve ser executado inteiramente ou então não ser executado em nenhum de seus subtrechos. As transações devem mapear um objeto de um banco de dados de um estado coerente para outro. Para manter a coerência, as transações devem passar pelo teste ACID : Atomicidade, coerência, isolamento, e durabilidade.

- **Atomicidade**: Como uma transação é executada inteiramente ou então não é executada, ou a seqüência completa de operações é aplicada ao banco de dados ou então nenhuma. Este recurso chama-se Atomicidade; as transações devem ser atômicas.
- **Coerência**: Diz-se que o banco de dados é coerente se todas as suas restrições de integridade são satisfeitas. Pressupõe-se que na execução de uma transação, na ausência de interferência de outras transações concorrentes, o banco de dados seja levado de um estado coerente para outro.
- **Isolamento**: Como as transações são executadas concorrentemente no mesmo banco dados, elas devem ser isoladas das outras operações. Do contrário, a operação intercalada de transações concorrentes pode levar a anomalias. Assim, os sistemas de gerenciamento de dados suportam isolamento, que fornece segurança contra interferências entre transações concorrentes.
- **Durabilidade**: A durabilidade está relacionada à capacidade do SGDB de se recuperar de falhas no sistema e no meio. As atualizações de uma transação efetivada devem ser preservadas e registradas em algum meio durável. Deve-se manter redundância suficiente para que se reconstrua um banco de dados coerente.

Transações aninhadas : As transações de aplicações de bancos de dados orientados a objetos são normalmente mais demoradas que as de aplicação comerciais convencionais. A longa duração das transações em aplicações avançadas é uma característica das aplicações de bancos de dados da próxima geração. Várias estratégias relacionadas à longa duração foram propostas na pesquisa de bancos de dados. Algumas estratégias influenciaram as implementações de bancos de dados orientados a objetos. As transações aninhadas são utilizadas para resolver alguns problemas associados às transações de longa duração. Um modelo de transação aninhada pode conter sub-transações, também chamadas transações-filhas. Em uma transação aninhada, todas as transações-filhas devem ser efetivadas para que a transação de nível mais alto se efetive. Cada sub-transação deve ser concluída ou abortada. Também, em aplicações avançadas, as tarefas normalmente envolvem vários usuários. As transações em cooperação são utilizadas para suportar essas tarefas em conjunto.

8.2 CONCORRÊNCIA

Vários algoritmos de controle podem ser usados para garantir a capacidade de serialização das transações e a coerência do banco de dados. O mais notável deles é o bloqueio. Nos bancos de dados orientados a objetos, o bloqueio pode ser associado a vários grânulos que são manipulados pelos usuários, incluindo classes, instâncias e objetos complexos.

Nos bancos de dados orientados a objetos, há dois aspectos de bloqueio que são relevantes para o compartilhamento concorrentes de objetos :

- Bloqueio de hierarquia de classe : As classes nos bancos de dados orientados a objeto são organizadas em hierarquias de herança, de modo que cada classe da hierarquia tenha uma extensão ou instância pré existente. Por isso é importante fornecer bloqueio de granularidade a essas estruturas. Por exemplo, uma superclasse poderia bloquear implicitamente todas as subclasses no mesmo modo de bloqueio. As subclasses incluem os descendentes diretos da superclasse e os descendentes de suas subclasses.
- Bloqueio de Objeto complexo : Os bancos de dados orientados a objetos contêm objetos que podem referenciar ou incorporar outros objetos. Além disso, alguns objetos são "valores", enquanto outros possuem identidade. Para otimizar a concorrência na presença de modelos que envolvam objetos complexos, foram analisados vários esquemas de bloqueio de "objetos compostos" ou de "objetos dependentes" para objetos complexos.

8.3 RECUPERAÇÃO

A confiabilidade e a pronta recuperação de falhas são importantes recursos de um sistema de gerenciamento de banco de dados. O gerenciador de recuperação é o módulo que administra as técnicas de recuperação dessas falhas. Os três importantes tipos de falhas que são responsabilidade do gerenciador de recuperação são: as falhas de transação, as falhas no sistema, as falhas no meio.

Uma das estruturas mais utilizadas para o gerenciamento de recuperação é o histórico de falhas. O histórico de falhas é utilizado para registrar e armazenar as imagens anteriores e posteriores dos objetos atualizadas. A imagem anterior é o estado do objeto antes da atualização da transação, e a imagem posterior é o estado do objeto após a atualização da transação. Quase todos os SGDBOO utilizam o registro de operações para a recuperação do banco de dados a um estado coerente. Alguns utilizam a duplicação ou espelhamento de dados.

8.4 CONTROLE DE VERSÃO

O acesso a estados anteriores ou a estados alterados de objetos é parte inerente de muitas aplicações. Ele é obtido por meio de várias versões do mesmo objeto. O gerenciamento de versão em um banco de dados orientado a objetos consiste em ferramentas e construções que automatizam ou simplificam a construção e a organização de versões ou configurações. Sem essas ferramentas, caberia ao usuário organizar e manter as versões.

Podemos considerar a configuração como um grupo de objetos tratados como uma unidade para bloqueio e controle de versões. Os objetos individuais dentro da configuração podem sofrer modificações, de modo que cada objeto pode ter um histórico das versões. Vários objetos dentro da configuração são atualizados em momentos diferentes e não necessariamente na mesma freqüência.

9. PADRÕES DE CONSULTAS DE OBJETOS

Wade faz uma descrição sobre o esforços realizados para definição de um padrão das consultas OQL. Vários grupos têm trabalhado no sentido de criar e definir os padrões para consultas SQL acessando os objetos, através da linguagem chamada OQL (Object Query Language). Entre estes grupos destacam-se o ODMG, OMG, ANSI X3H2 (SQL3).

9.1 O QUE É DIFERENTE AO CONSULTAR OBJETOS ?

A introdução de objetos dentro de bancos de dados trouxe uma série de novas questões. De fato, isto trouxe a necessidade de novos tipos de interfaces. Como complemento para a já familiar declaração da linguagem de consulta, o tratamento de objetos apresenta a possibilidade de integrar a interface com a linguagem de objetos, assim os objetos usados nas linguagens podem ser manipuladas automaticamente pelo banco de dados. Isto permite ao programador um integração com uma linguagem de objetos e evitar tradução entre estruturas do banco de dados e as estruturas da linguagem. Além disso, a linguagem para definição de objetos é incrementada permitindo muitas diferentes tipos de estruturas. Finalmente, a linguagem de consulta também permite acesso a diferentes tipos de estruturas. Assim há possibilidades de interfaces para **definição de tipos, de consultas**, e para **integração com linguagens de programação orientado a objeto**, tudo com aumento significativo da variedades de tipos.

Mais que a diferença na estrutura, a natureza fundamental dos objetos apresenta novas capacidades que devem ser endereçadas pode estes 3 tipos de interface. Dentre estas capacidades, inclui :

- Encapsulamento (operações)
- Identidade
- Referências
- Coleções

Uma diferença básica entre um objeto e os dados tradicionais é que o objeto também inclui operações. O usuário do objeto pode invocar estas operações e pode acessar externamente informações visíveis, os chamados atributos. Ambas operações que podem ser implementadas usando informações internas que são invisíveis externamente. A definição do tipo para o usuário externo, então, deve incluir a especificação do atributo e da operação. Para implementar objetos deve existir uma maneira para especificar a implementação de operações e da mesma forma qualquer informação interna.

Outra diferença é a identidade. Bancos de dados tradicionais armazenam somente dados. Os dados podem ser consultados e modificados por associação (por valor), e usuário pode criar chaves baseado-se nestes valores para localizar registros desejados. Com objetos, o sistema mantém o conceito de identidade. Mesmo se todas os valores dos atributos do objeto mudarem, o usuário pode ainda acessar o mesmo objeto pela geração interna e gerenciamento de identificadores de objetos (OIDs), e o usuário pode solicitar ao sistema verificar se dois objetos são de fato o mesmo (operação de identidade). As interfaces dos objetos do banco de dados, devem prover alguns mecanismos para acessar estas identidades.

Além do usuário ter acesso a atributos de objetos visíveis externamente e a operações, há outra característica características que são visíveis e acessadas externamente do objeto, as relações entre objetos. Para relacionamentos binários, um relacionamento simples, um-para-um pode ser visto como um atributo de objetos com valores de dois objetos relacionados. Relacionamentos muito-para-muitos podem ser vistos como coleção de objetos de valores. A interface para definição de tipos deve suportar uma maneira para declarar tais relacionamentos, e as interfaces de consultas e linguagem deve suportar meios de criar, remover e associar relacionamentos.

Coleções podem ser vistas como atributos de múltiplos valores, onde valores individuais, ou membros da coleção, consistem de vários, possivelmente complexos, tipos de dados, tais como objetos. Vários sub-tipos de coleções são possíveis, incluindo um bag, um set, uma list, um array, todos podendo variar de tamanho dinamicamente. Novamente, a linguagem de definição de tipos deve prover uma maneira de definir tais coleções, e as interfaces das linguagens de consulta e dos objetos deve prover uma forma de criar, remover, consultar e atualizar tais coleções.

9.2 ODMG

Object Database Management Group (ODMG), um consórcio de fornecedores de banco de dados orientados a objetos, criou todas as 3 interfaces descritas acima para propor-las ao OMG (Object Management Group), ANSI (American National Standards Institute), ISO (International Standards Organization) e outras organização relevantes. Em particular, o ODMG-93 contém 4 interfaces :

- Object Definition Language (ODL)
- Object Query Language (OQL)
- Compatibilidade com C++
- Compatibilidade com SmalTalk

Para a interface para uma linguagem de objetos, o ODMG escolheu usar linguagens pré-existentes, como a C++ e a SmalTalk, e associa-las com o modelo de objetos do banco de dados, desta forma os programadores destas linguagens podem definir objetos normalmente, e o banco de dados irá suporta-los automaticamente.

Para definição de objetos, a ODMG iniciou com a Linguagem de Definição de Objetos (IDL) da OMG, e adicionou capacidades extras necessárias ao sistema de gerenciamento de dados, incluindo declarações de objetos, relacionamentos bi-direcionais n-para-n, chaves e extents. Combinando com as habilidades do IDL para definir atributos e operações, isto permite definir qualquer tipo de objeto. Como todos os adicionais a linguagem de definição de interfaces são definidas semanticamente equivalente para os métodos gerados, todos os objetos definidos pela ODL produzem interfaces IDLs, sendo assim acessíveis para todos os padrões do mundo OMG, incluindo o CORBA, os Serviços de Objetos e as Facilidades dos Objetos.

A Figura 5 ilustra como as interfaces ODMG interagem com o sistema de gerenciamento de banco de dados.

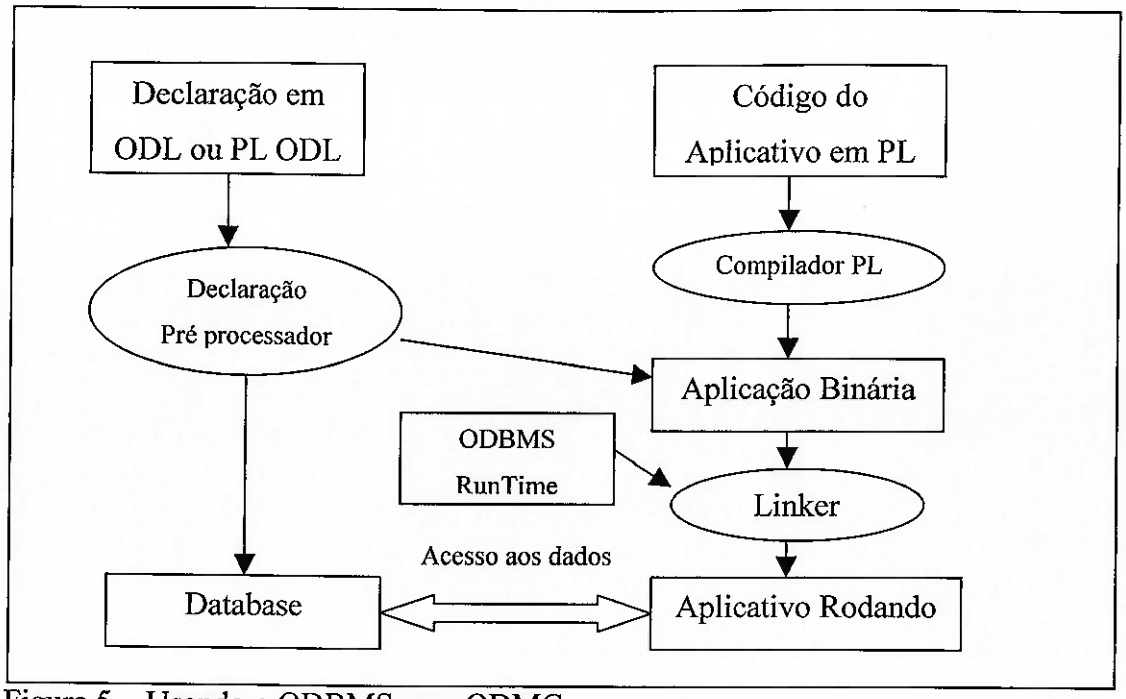

Figura 5 – Usando o ODBMS com ODMG

Para consultas, ODMG adotou as capacidades de consulta básicas do SQL-92 (algumas vezes conhecidas com o apelido SQL2), o já conhecido e tradicional comando SELECT, e adicionou a habilidade em aplicar consultas para qualquer objeto ou coleção de objetos, e a habilidade de invocar métodos a partir das consultas. Isto é definido por uma simples e limitada, linguagem funcional de tipos. Os tipos resultante de uma consulta pode ser um tipo escalar (incluindo tuplas), um objeto, ou uma coleção de objetos, com regras de combinação de tipos especificando quais operações cada tipo produz com outro. A invocação de operações é sintaticamente a mesma da referência a atributos, com a opção de incluir parâmetros a invocação. Relacionamentos horizontais são implementados através do ponto (a..b). Também foi adicionado a sintaxe para definição de coleções, incluindo lists, sets, bags e arrays.

Na seqüência encontra-se uma série de exemplos da implementação da linguagem SQL2 baseando nas definições do ODMG aplicado para objetos.

- `SELECT X FROM X IN FACULTY WHERE X.SALARY < X.DEPT.CHAIR.SALARY`
- ```
  SORT S IN
  (SELECT STRUCT (NAME:X.NAME, S:S.SSA)
  FROM X IN FACULTY
  ```
  ```

```
WHERE FOR ALL Y ON X.ADVISSES:Y.AGE < 25) BY S.NAME
```

- `CHAIR.SALARY`
- `STUDENTS EXCEPT TAS`
- `LIST (1,2) + LIST ( COUNT (JSE.ADVISEES), 1+2 )`
- `EXISTS X IN FACULT[1:N]: X.SPOUSE.AGE < 25`

Exceto pelas implementações para se tornar compatível com os objetos, como relacionamento horizontal, parâmetros para operações e coleções, OQL é muito similar com as consultas READ-ONLY do SQL2, o comando SELECT. Na versão V1.2, ODMG procurou tornar o OQL completamente compatível com o SQL2 de modo que qualquer consulta válida em SQL2 deve ser uma consulta OQL válida, com a mesma sintaxe, semântica, e resultados. Infelizmente, as linguagens não se tornaram compatível para todos os casos. Há alguns casos em que uma consulta OQL e uma consulta sintaticamente idêntica em SQL2 irão produzir resultados diferentes. SQL2 sempre irá trabalhar em tabelas, mas OQL pode produzir coleções. Por exemplo, a consulta abaixo

```
SELECT F(X) FROM R X WHERE P(X)
```

retorna uma tabela do tipo

```
MULTISET(ROW(TYPE(F(X))))
```

enquanto o OQL retornará

```
MULTISET(TYPE(F(X)))
```

Por causas destas exceções, ODMG fez uma pausa em tentar tornar o OQL totalmente compatível com SQL.

9.3 OMG

O Object Management Group (OMG) é um consórcio de empresas que defini as especificações de interfaces para que se permita a interoperabilidade enter objetos e ferramentas de objetos. O grupo começou com a definição do CORBA, uma interface que permite que operações de objetos remotos possam ser invocadas. Cada objeto registra nos dispatchers dos CORBA as operações que eles suportam por meio de IDL. Então o OMG adicional definições de interfaces para diversos serviços, cada de em conformidade com a sintaxe do IDL. Entre estas interfaces, temos:

- Nomeação
- Persistência
- Relacionamento
- Transações
- Consultas

O OMG tem trabalhando com a ISO Open Distributed Processing (ODP) para colocar sua interface CORBA com o processo formal de padronização.

Diversos serviços do OMG sobrepõem o domínio do sistema de gerenciamento de dados. Na verdade, o sistema de gerenciamento de dados pode ser visto como o fornecedor destes diversos serviços do OMG, todos juntos em um pacote, de modo que o pacote também inclui recuperação, buffer e índices para desempenho. Outra forma de ver a relação entre o OMG e o sistema de gerenciamento de dados é vê-lo como uma camada. O serviço de transações do OMG, por exemplo, é compatível com o protocolo X/Open X/A para realizar two-phase commit, que é usado por gerenciadores de transações para coordenar transações atômicas entre múltiplos sistemas de gerenciamento de dados e outros gerenciadores.

O serviço de persistência propicia uma interface que permite a um cliente requisitar que um objeto seja persistente, incluindo armazenar o estado atual do objeto ou recuperar um estado armazenado previamente. O serviço de persistência inclui outros serviços, como :

- **Persistent Object Manager (POM):** é a interface entre o cliente e o objeto que está sendo armazenado.

- **Persistent Identity Service (PID):** serviço que gera os identificadores para ser utilizado pelos clientes para futuras recuperações
- **Persistent Data Store (PDS):** serviço que interagem com o banco de dados e é responsável por manter as informações do objeto

O protocolo de comunicação entre o serviço POM e PDS permite que as informações possam residir em diferentes meios, como em arquivos seqüências ou em um banco de dados relacional, enquanto o PDS utiliza a interface ODMG-93 para que o objeto que está sendo manipulado seja armazenado como objeto, em um SGBD que suporte a interface de objetos.

A figura 6 ilustra o relacionamento entre estes serviços.

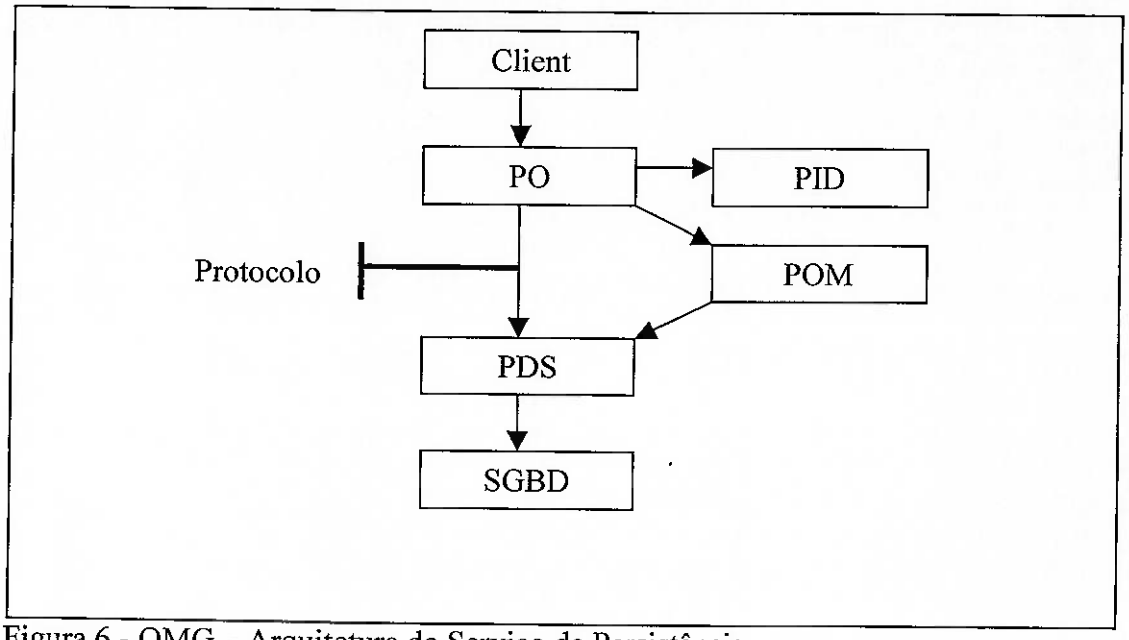

Figura 6 - OMG – Arquitetura do Serviço de Persistência

A especificação do OMG Query Service define servidores de consultas aninhadas e ligadas. O usuário envia uma consulta para um servidor de consultas, o qual passa as sub consultas para outros servidores de consultas. Assim, se uma sub consulta se aplica a objetos armazenados no banco, o SGBD pode executá-la diretamente usando qualquer mecanismo de otimização que ele tenha. A consulta de nível mais alto monta os resultados de todas as sub consultas, faz a avaliação dos predicados das consultas e retorna o resultado, que pode ser um tipo escalar OMG, um objeto, uma coleção de objetos, para o usuário. A figura 7 ilustra como estes servidores de consulta se interagem.

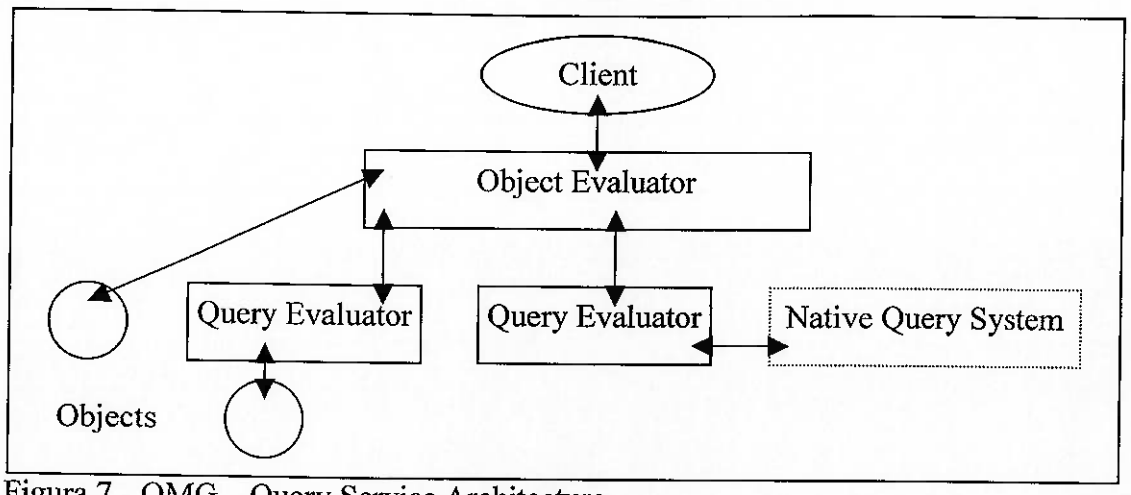

Figura 7 - OMG – Query Service Architecture

A linguagem utiliza para expressar a consulta pode ser qualquer linguagem combinada entre o servidor e o cliente. A linguagem deve utilizar a sintaxe dos objetos segundo a OMG. A especificação requer entretanto que ela seja compatível com os seguintes itens :

- Que seja uma derivação do SQL2 (similar com SELECT mais INSERT, UPDATE e DELETE): para que seja compatível com maioria das ferramentas de consulta que já existem no mercado.
- OQL: para que a linguagem seja compatível com a funcionalidade de objetos
- Uma variação do OQL, restrita para consultas simples, mas que inclua operações: para permitir o uso de outros serviços do OMG que precisam de uma linguagem simples para avaliação de predicados, que inclua a capacidade de invocar operações.

O OMG continua a trabalhar em outras áreas relevantes com o SGBD.

9.4 SQL3

ANSI X3H2, em cooperação com ISO/IECJTCI/SC21/WG3 DBL, trabalhou no que foi considerado um novo padrão para consultas, que foi nomeado de SQL3. Esta versão inclui novas capacidades, e as mais significantes são: funcionalidade de objetos, e uma Linguagem Computacional Completa (PSM).

A adoção do PSM propicia uma linguagem completa, com controle de fluxo, operações procedurais, e resolução de funções. Isto é diferente do escopo do OMG e ODMG. Ao invés de criar sua própria linguagem, mapeou linguagens que já existem no mercado, como C++ e SmalTalk. Mas não há conflitos, uma vez que os grupos OMG e ODMG esperam adicionar outras linguagens no futuro. Assim o OMG e o ODMG poderá criar associações com e só tornarem-se compatível com o PSM o SQL3.

A adoção da capacidade de trabalhar com objetos inclui a habilidade para definir e acessar Abstract Data Types (ADTs), os quais tem praticamente as mesmas funcionalidades dos objetos OMG e ODMG, incluindo acesso aos atributos externas visíveis e operações. A linguagem própria do SQL3 (PSM) é usado para implementar as operações e atributos internos dos objetos. O objetivo é permitir que existam objetos em linhas das tabelas. Estas linhas poderiam conter ADTs. O mecanismo para identificar ADTs em outras linhas existe, e assim é possível identificar um valor único como uma linha, ao invés de fazer procura baseado em valor. Este é o conceito básico de identidade, semelhante aos PID do OMG e a referência de objetos do ODMG.

Consultas em SQL3 incluem as habilidades das consultas SQL2, acompanhadas com habilidade adicional para invocar métodos e relacionamentos horizontais. Isto é mais que as consultas ODMG ou OMG.

9.5 ESFORÇOS PARA UNIFICAÇÃO

Embora diferentes SGBD e arquiteturas são desenvolvidas, não é desejável que haja duas ou mais linguagens de consultas diferentes. Certamente, seria melhor que pelo menos um núcleo comum declarativo de consulta de linguagem pudesse ser combinada, ainda permitindo a diversidade de implementações por baixo dela.

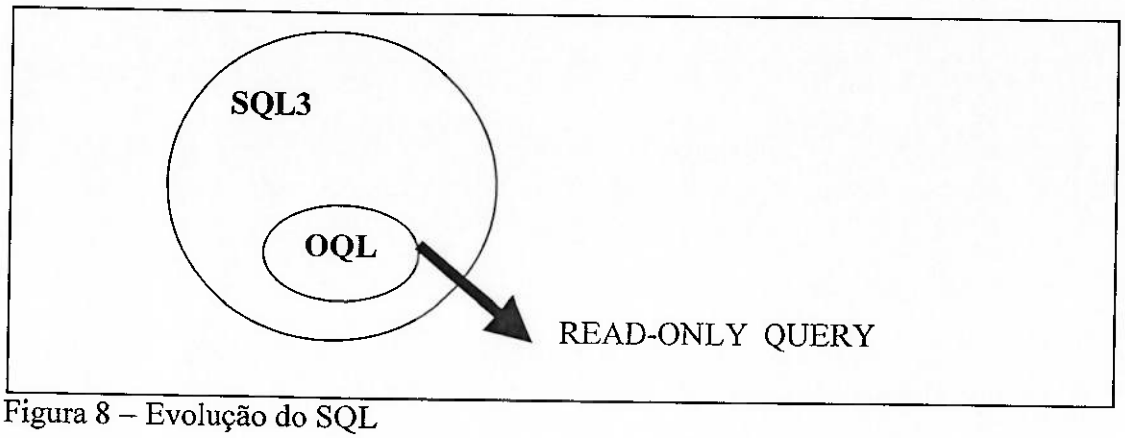

Figura 8 – Evolução do SQL

Isto é exatamente o objetivo de uma união de um grupo de trabalho formando entre X3H2 e ODMG. Um pequeno grupo tem trabalhado juntos procurando um padrão. O objetivo, como indicado na figura 8, é não forçar um modelo ou um padrão qualquer ser o mesmo que o anterior, mas preferindo procurar uma linguagem comum de consultas read-only. Um dos fatores que dificulta a padronização e a compatibilidade total com os comandos SQL, é que as consultas OQL manipulam diferentes tipos de dados, como objetos, coleções e não somente tuplas como é no caso dos comandos SQL. É importante lembrar que mesmo as consultas SQL há alguns casos especiais que não estão completamente padronizados e dependentes do fornecedor do Sistema de Gerenciador de Dados. Outra questão que merece atenção é como é feita a manipulação dos OID, que inicialmente foi armazenado junto com os dados da tabela mas que sofria alterações caso o objeto move-se de lugar. Isto foi endereçado armazenando apenas referências a ADTs nas linhas.

10. EXEMPLOS DA LINGUAGEM OQL

O objetivo aqui não é descreve todas as características da linguagem OQL nem como cobrir todos extensões propostas pelos padrões ODMG. Serão abordados alguns temas e demostrado como a linguagem OQL atende as consultas. O primeiro produto comercial a fazer uso da linguagem OQL foi o O2, baseado-se nos padrões definidos pela ODMG. O ODMG lançou versões finais dos padrões nos anos 1993 (ODMG 1.0), 1995 (ODMG 1995) e 1997 (ODMG 2.0).

Na linguagem SQL, os resultados de uma consulta são sempre tuplas de uma tabela. Já no OQL, o resultado pode ser dados atômicos, estruturas ou coleções.

O conceito tradicional de tabela do SQL é obtido em OQL através da definição de struct..

Ex. :

```
STRUCT ENDERECO { STRING RUA; LONG NUMERO; DATE DATA_ATUAL; }
```

O resultado de uma consulta OQL pode ser um Bag, um Set ou um List.

Um Bag é o resultado de um consulta sem ordenação e com linhas duplicadas. Esta consulta pode retornar um Bag de objetos ou literais.

Ex. :

```
SELECT P.IDADE FROM PESSOAS AS P
```

Neste exemplo é retornado um `BAG<INTERGER>`.

Já o Set trata-se de um retorno de uma consulta ainda sem ordenação, mas sem linhas duplicadas. E o List trata-se de um retorno ordenado de uma consulta e sem linhas duplicadas.

O resultado é caracterizado como set<> quando for utilizada a cláusula distinct. O list<> é caracterizado quando é utilizada a cláusula order by.

Ex.:

```
SELECT P FROM PESSOAS P ORDER BY P.IDADE[0..4]
```

Neste exemplo é retornada uma list<pessoa> com 5 objetos.

Para facilitar o entendimento dos próximos exemplos de consultas OQL, iremos utilizar o seguinte modelo de objetos :

O exemplo será utilizado para demonstrar como obter relacionamento e junções em OQL. Baseando-se no modelo de objeto, o comando OQL abaixo retorna o conjunto de nomes dos vendedores da filial de nome "Canoas".

Ex.:

```
SELECT DISTINCT VEND.NOME
FROM VENDEDORES VEND
WHERE VEND.AFILIAL.NOME='CANOAS'
     //RELACIONAMENTO IMPLÍCITO POR EXPRESSÕES DE CAMINHO
OU
SELECT DISTINCT V.NOME
FROM FILIAIS F, F.OSVEND V
WHERE F.NOME = 'CANOAS'
```

Abaixo está o exemplo utilizando expressões SQL tradicionais para mostrar a diferença entre OQL e SQL.

Ex. :

```
SELECT DISTINCT V.NOME
FROM FILIAIS F, VENDEDORES V
WHERE (F.NOME='CANOAS') AND (F.CODFILIAL=V.CODFILIAL)
      //RELACIONAMENTO EXPLÍCITO POR JUNÇÕES DE CHAVES
```

Abaixo está o exemplo utilizando expressões SQL tradicionais para mostrar a diferença entre OQL e SQL.

O exemplo abaixo exemplifica uma busca com os nomes da filiais nas quais um vendedor realizou pelo menos uma venda em "13/05/98".

Ex. :

```
SQL -
SELECT F.NOME
FROM FILIAIS F, VENDEDORES VEND, VENDAS VD
WHERE  (VD.DATA='13/08/2000')  AND    (F.CODFILIAL=V.CODFILIAL)  AND
     (VD.CODVEND=VEND.CODVEND)
OQL -
SELECT F.NOME
FROM FILIAIS F
WHERE  EXISTS  (SELECT  V  FROM  F.OSVEND.ASVENDAS  V  WHERE  V.DATA  =
     '13/05/98')
OU
SELECT V.OVEND.AFILIAL.NOME
FROM VENDAS V
```

```
WHERE V.DATA = '13/05/98'
```

O OQL possui uma característica interessante, pois permite o aninhamento do resultado, facilitando a interpretação.

Ex. :

```
SELECT DISTINCT STRUCT (NOME:EMP.NOME, ALTOS_EMP:(SELECT S FROM
EMP.SUBORDINADOS S WHERE S.SALARIO>5000))
FROM EMPREGADOS EMP
```

O OQL já implementa um recurso que demorou a ser implementado no SQL o tradicional aninhamento na cláusula from, where e having.

Ex. :

```
SELECT STRUCT (ID:EMP_GRAD.IDADE, SX:EMP_GRAD.SEXO)
FROM (SELECT EMP FROM EMPREGADOS EMP WHERE EMP.NIVEL=10) AS
EMP_GRAD
WHERE EMP_GRAD.NOME = "JOSÉ "
```

Baseado-se agora no seguinte comando SQL ou OQL, pois a sintaxe está de acordo com ambos os padrões.

Ex. :

```
SELECT DISTINCT P.IDADE, P.SEXO
FROM PESSOAS AS P
WHERE P.NOME = "ROBERTO"
```

Segundo as definições do OQL, idade e sexo são atributos dos objetos que serão resultados da consulta. P.nome acessa e compara o atributo do objeto. Pessoas é o *extent* da classe pessoa, ou seja, o conjunto de todos os objetos desse tipo. Usualmente usa-se variáveis de interação. "P" no exemplo é uma variável de iteração que acessa os objetos.

Em OQL não é possível ter acesso a campos vazios, normalmente definidos como "nil". O campo que contém NIL é classificado como UNDEFINED e a consulta resulta em erro pois UNDEFNIED não pode aparecer no resultado. Para tratar isto, deve ser utilizada uma função para filtrar os dados UNDEFINIED.

Ex.:

```
SELECT EMP.ENDERECO.CIDADE
FROM EMPREGADOS EMP
WHERE IS_DEFINED(EMP.ENDERECO.CIDADE)
```

A função `IS_DEFINED()` trata as linhas com endereços nil. A consulta retorna um bag apenas com as cidades que foram definidas no banco.

Quando se trabalha com RDBMS e SQL, é obrigatório o uso do COMANDO SELECT ... FROM ... para ter acesso aos dados da tabela. Em SGBDOO, pode ser usado diretamente o nome do objeto ou extents.

Ex.:

```
OBTER O CONJUNTO DE TODAS AS PESSOAS
      PESSOAS
OBTER O OBJETO CUJO NOME É SABRINA:
      SABRINA
OBTER A IDADE DE SABRINA:
      SABRINA.IDADE
```

A entrada de dados em OQL não utiliza a sintaxes INSERT, UPDATE e DELETE. Deve ser utilizado a sintaxe NOME_DO_TIPO(...) para criar uma nova instância (objeto) de um tipo/classe.

Ex. :

```
EMPREGADO(NOME:"JOSÉ DA SILVA", SEXO:"M", DATA_NASC:"13/05/50",
SALARIO:325, CHEFE:PRESIDENTE)
```

A sintaxes dos comandos de agregação difere da sintaxe tradicional do SQL, utiliza-se a sintaxe <OPER>(EXPRESSÃO QUE REPORNA COLEÇÃO>)

Ex.:

```
MAX(SELECT EMP.SALARIO FROM EMPREGADOS EMP)
COUNT(PROFESSORES)
FIRST( SELECT F
      FROM FILIAIS F
      ORDER BY F.CALCTOTALVENDASMES(0198)ASC)
LAST( SELECT F
      FROM FILIAIS F
            ORDER BY F.CALCTOTALVENDASMES(0198)ASC)
```

11. RELAÇÃO DE ALGUNS SGBDOO EXISTENTES NO MERCADO

Abaixo encontra-se uma tabela baseado no trabalho feito por Zand, Collins E Caviness. Uma análise de alguns bancos de dados orientados a objetos disponíveis no mercado. Os produtos foram avaliados segundo as seguintes características :

1. **Usuário Primário:** Representa para qual tipo de usuário o produto está disponível atualmente
2. **Versionamento:** Indica se o SGDB possui controle de versões dos objetos
3. **Método de Recuperação:** Avalia qual método de recuperação o produto utiliza para realizar recuperação dos dados
4. **Gerenciamento de Transações:** Indica se o Sistema de Gerenciamento de Dados Orientados a Objetos faz o gerenciamento de transações
5. **Objetos Compostos:** Avalia se o SGBD tem a capacidade de criar objetos compostos, objetos formados por outros objetos
6. **Herança:** Indica se o SGBD é capaz de permitir herança em objetos
7. **Concorrência:** Avalia a capacidade de ser executada transações concorrentes sobre os mesmos objetos
8. **Distribuído:** Indica se o SGBD é capaz de realizar transações distribuídas, entre bancos de dados diferentes
9. **Evolução dinâmica:** Representa a capacidade do SGBD de suportar evolução dinâmica entre os objetos
10. **Dados multimídias:** Avalia a capacidade de gerenciamento de dados do tipo imagem, som e vídeo
11. **Linguagem de Interfaces**: Documentas as linguagens de interface aceita pelo SGBD
12. **Relacional ou Orientado a Objeto:** Avalia se o SGDB tem a capacidade de orientação a objetos ou implementa o método objeto –relacional
13. **Plataforma**: Indica em qual plataforma o produto está disponível
14. **Distribuição**: Indica em qual tipo de mercado o produto está disponível

	Avance	ENCORE ObServer	EXODUS EXTRA	GemStone	IRIS	KIWI	02	ObjectStore
Usuário Primário	Sistemas de Informação, Sistemas de Tolerância a Falhas	CAD	N/D	Ambientes Cooperativos	OIS, CAD/CAM, Sistemas de Gerenciamento de Conhecimento	Sistemas de Gerenc. de Conhecimento	CAD/CAM, GIS, OIS, Negócios e Processamento de Transações	Ambientes Cooperativos
Versionamento	Sim	Sim	Sim	Sim	Sim	Sim	Limitado	Sim
Método de Recuperação	Através de Ger. Versão	Log	Log	Log	HP-SQL	-	Sim	Log
Ger. Transação	Sim	Sim	Sim	Sim	Sim	-	Sim	Sim
Comp. Objetos	-	Sim	Sim	Não	Forçado	Sim	Sim	Forçado
Herança	Sim	Sim	Sim	Não	Sim	-	Sim	Sim
Concorrência	Two-Phase Commit	4 tipos Locks	Sim	3 Tipos Locks	Sim	Sim	Sim	2 Tipos Locks
Distribuído	Sim	-	Não	Sim	-	-	Sim	Não
Evolução Dinâmica	Dinâmica e Estática	Sim	-	Sim	Sim	Sim	Sim	Sim
Dados Multimídias	Não	Sim	Sim	Sim	Não	-	Sim	Sim
Ling. Interfaces	PAL	C	C++	C, C++, OPAL,	C, LISP, OSQL	OOPS+	C	C,C++

	Avance	ENCORE ObServer	EXODUS EXTRA	GemStone SmallTalk	IRIS	KIWI	02	ObjectStore
Relacional/OO	OO	OO	OO	OO	Relacional	-	Relacional	-
Plataforma	-	Sun e VMS	-	Sun, Apple, VMS, IBM, PCs	HPs	Sun	Sun	Sun, HP, DEC
Distribuição	Protótipos	Universidades	Universidades	Comercial	Protótipos	Protótipos	Comercial	Comercial
Características Especiais	-	Notificação de Mudanças, Ger. De Objeto	Notificação de Mudanças, Gerador de BD	Notificador de Mudanças	BD Privados e Compartilhados	Interface Visual	Interface Visual, Poderosa Linguagem de Consultas	BD Privados e Compartilhados

Tabela 1 – Comparativo de Bancos de Dados

	ONTOS	ORION	OZ+	POSTGRES	Starburst	Statice	VERSANT
Usuário Primário	CAD/CAM	CAD/CAM, AI, OIS, Informações Multimídias	Sistemas de Informação	CAD/CAM, Sistemas de Gerenc. de Conhecimento	CAD/CAM, AI, Sistemas Específicos, Sistemas de Gerenciamento de Conhecimento	-	Engenharia Coloborativa
Versionamento	Sim	Sim	Sim	Sim	Não	Limitada	Sim
Recuperação	Sim	Log	Periodical update	NoOverwrite	Rollback	Redo Log	-
Ger. Transação	Sim	Sim	-	Sim	Sim	Sim	Sim
Comp. Objetos	Não	Sim	Objetos Complexos	Sim	Objetos Complexos	Sim	Sim
Herança	Sim	Sim	Não	Sim	Sim	Sim	Sim
Concorrência	4 Tipos Lock	5 Tipos Locks	Sim		Regras e Rollback	2-Phase Lock	2-Phase Lock 4 Tipos de Lock
Distribuído	Sim	Sim	Sim	Não	Sim	Sim	Sim
Evolução Dinâmica	Sim	Sim	-	-	-	-	Sim
Dados Multimídias	Não	Sim	Não	Sim	Sim	Sim	Não
Ling. Interfaces	C++	LISP, C	OZ+	C, C++, LISP	C, C++	LISP	C, C++

	ONTOS	ORION	OZ+	POSTGRES	Starburst	Statice	VERSANT
Relacional/OO	OO	OO	Relacional	Relacional	Relacional	OO	OO
Plataforma	SUN, VMS	Sun, HP	SUN	SUN	IBM PCs, RISC 6000	Máquinas LISP	Sun
Distribuição	Comercial	Comercial	Protótipos	Protótipos	IBM Testes	Comercial	Comercial
Características Especiais	Objetos SQL	Notificador de Mudanças, BD Compartilhados e Privados					

Tabela 2 – Comparativo de Bancos de Dados (Continuação)

12. CONCLUSÃO

Os sistemas de gerenciamento de bancos de dados orientados a objetos estão evoluindo gradativamente e assumindo as capacidades das linguagens de orientação a objeto, incorporando-se aos sistemas de gerenciamento de dados. Assumir as características pode tornar a solução dependente de uma determinada solução, como a linguagem C, JAVA ou SmalTalk, por isso o trabalho dos consórcios de fornecedores de software é importante para fortalecer o conceito dos SGBDO e definir um padrão de modo a atender todas as soluções de mercado.

Enquanto um padrão não for definido, acredito que o sucesso dos SGBDOO estará comprometido, uma vez que o mercado já está acostumado com a versatilidade de não ter dependência de uma linguagem e que o desenvolvimento seja feito com um banco de dados especifico, este podemos considerar um dos motivos mais importantes que tornou tal popular os Sistemas de Gerenciamento de Dados Relacional.

Atualmente um aplicativo pode ser desenvolvido para operar em diferentes sistemas de gerenciamento de dados, com o mínimo de alterações e personalizações, na maior parte das situações, tendo em vista os padrões definidos pelo ANSI X3H2 (SQL), padrões estes implementados por todos os sistemas de gerenciamento de dados relacionais disponíveis. Os Sistemas de Gerenciamento de Dados Orientados a Objetos somente se tornarão uma alternativa viável para todas os tipos de soluções quando alcançarem o mesmo grau de padronização e flexibilidade.

As características definidas pelo Manifesto do Sistema de Gerenciamento de Dados apresentam as características que norteiam o desenvolvimento dos Sistemas de Gerenciamento de Dados Orientados a Objetos e que devem servir como referência no momento de escolher um sistema disponível no mercado. A característica principal é como os sistemas tratam a persistência dos objetos e a torna transparente para o usuário desenvolvedor, assim práticas de desenvolvimento utilizadas atualmente poderão ser facilmente convertidas quando os programas utilizarem bancos de dados orientados a objetos. Outra característica de destaque é a habilidade de definição de objetos de maior complexidade, tornando a manipulação de arquivos de multimídia mais amigável.

Apesar dos Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos apresentar inúmeras vantagens e as suas características o indicarem como a solução perfeita para os novos desafios dos novos sistemas, a aceitação do produto no mercado possui os mesmo desafios que as linguagens de programação orientadas a objetos vem enfrentando e por isso ainda não conseguiu se tornar o padrão de desenvolvimento atual. A falta de profissionais capacitados, dificuldade de aprendizado, a falta de padronização da linguagem são os principais problemas enfrentados pelos Sistemas Gerenciadores de Bancos de Dados Orientados a Objetos. Os SGBD Relacionais e os SGBD Orientados a objetos ainda irão compartilhar o mercado de banco de dados por muito tempo, e demorará bastante tempo para os sistemas atuais migrarem para o SGDB Orientados a Objetos.

Isto leva a conclusão que a tecnologia dos SGBD Orientados a Objetos apresenta características importantes para os atuais sistemas em desenvolvimento, e que as equipes de desenvolvimento devem investir no aprendizado e na utilização de tecnologias orientadas a objetos, pois sua utilização será necessária devido a complexidade dos sistemas e os requisito dos aplicativos atuais.

13. BIBLIOGRAFIA

Neto ,R. N.; Pereira Neto ,F. G.. **Banco de dados Orientados a Objetos**, 1998

Third-Generation Database System Manifesto – The Committee for Advanced DBMS Function

IDC, número 22542. **"Enterprise Database Management Systems Market"**- Forecast and Analysis, 2000-2004, disponível em <http://www.idc.com>, maio 2000

Market Overview – Trajectory Analysis : ODBMS Vendors – dezembro, 1997

Brian, J. **The Object Database Management Group** – DBMS, julho 1997

Obasanjo, D.. **An Exploration of Object Oriented Database Management Systems**, Julho 1997

Wade, A. E., Ph.D. **Object Query Standards**. SIGMOD Record, Vol.25, No. 1, março 1996

Zand, M.: Collins, V.; Caviness, D. **A Survey of Current Object-Oriented Databases.** Em : DataBase Advances in Information Systems, Vol. 26. No. 1, fevereiro,1995.