COMPARATIVO ENTRE BANCO DE DADOS ORIENTADOS A OBJETO

COMPARATIVE AMONG DATABASE OBJECT ORIENTATION

*Reinaldo Carlos Godinho Tonani**
*Adail Roberto Nogueira***

RESUMO:
O desenvolvimento de aplicativos utilizando o Sistema Gerenciado de Banco de Dados Orientado a Objeto - SGBDOO vem crescendo muito, sendo utilizado por várias empresas de grande porte e também em estudos nas Universidades. O grande diferencial deste modelo de banco dados é que reúne todos os princípios da orientação a objeto e também faz uma junção ao modelo relacional favorecendo consultas aos dados, o que é uma característica de um modelo pós-relacional. Desta forma, este artigo apresenta um comparativo entre 9 diferentes SGBDOO, sendo direcionado principalmente para pessoas, empresas e universidades que necessitam de informações específicas com relação aos diversos SGBDOO encontrados no mercado desejando conhecer suas características e funcionalidades. Dos 9 SGBDOO estudados e detalhados, 4 foram eleitos para um estudo comparativo abordando suas principais características, como Arquitetura, Modelo de Dados, Evolução de Esquema, Acesso Multidimensional e Arquitetura WEB. O presente artigo inicia com um breve levantamento histórico sobre SGBDOO, desde a década de 60, com os modelos em rede e hierárquicos, passando pelas décadas de 70 e 80, com os modelos relacionais, até chegar aos dias atuais com os modelos objeto relacional e orientado a objetos. Em seguida são descritas as principais características dos 9 SGBDOO estudados, sendo eles ITASCA, GEMSTONE, VERSANT, OBJECTIVITY/DB, ONTOSDB, CACHÉ, JASMINE, OBJECTSTORE e O2. Então, é apresentado o estudo comparativo dos SGBDOO CACHÉ, OBJECTIVITY/DB, ITASCA e OBJECTSTORE. Finalmente, são explicitadas as conclusões do estudo aqui publicado.

PALAVRAS-CHAVE: Banco de Dados; Orientação a Objetos; SGBD; SGBDOO.

ABSTRACT:
The development of applicatory using the Managed System of Database Object Orientation – SGBDOO has grown lately, being used by many great companies and also used in studies at Universities. This database model great differential is that it congregates al the object orientation and it also makes a junction to the reactionary model in relation to the data consulting, which is a post-reactionary model characteristic. Thus, this article presents a comparative among 9 different SGBDOO's, being directed mainly to people, companies and universities that need specific information related to the several SGBDOO found in the market, so that it is possible to know their characteristics and functionalities. From the 9 SGBDOO's studied, 4 of them were elected for a comparative study approaching their main characteristics, as Architecture, Data Models, Scheme Evolution, Multidimensional Access and WEB Architecture. The present article stars off from a brief SGBDOO historical survey, since the 1960th decade, about net and hierarchic models, going through the 1970th and 1980th decades, about the reactionary models, up to recent times, with the reactionary and object orientation models. Following, it is reported the main characteristics of the 9 SGBDOO's studied, which are ITASCA, GEMSTONE, VERSANT, OBJECTIVITY/DB, ONTOSDB, CACHÉ, JASMINE, OBJECTSORE, and O2. Finally, it is explained the conclusions of this very article.

KEY WORDS: Database; Object Orientation; SGBD; SGBDOO.

*Acadêmico do Curso de Especialização em Engenharia de *Software* com UML, turma de 2004, do Centro Universitário Filadélfia – UniFil.
**Mestre pela UFSCar. Coordenador dos Cursos de Bacharelado em Sistemas de Informação, Bacharelado em Ciência da Computação e Tecnologia em Processamento de Dados da UniFil. Docente do Curso de Especialização em Engenharia de *SoIftware* com UML desta mesma instituição.

INTRODUÇÃO

A Engenharia do Software surgiu na década de 70 como uma junção de tecnologia e prática utilizadas na criação de *softwares,* com a finalidade do aumento da produção e qualidade do *software*.

A prática de engenharia de *software,* tratando-se de desenvolvimento de sistema, necessita certos requisitos para uma boa produção, sendo eles: metodologias, linguagens de programação, banco de dados, ferramentas, plataformas, bibliotecas e processo (WIKIPEDIA, 2005).

Este tipo de prática é utilizado para estabilizar sistemas mais complexos, tendo como característica um conjunto de componentes de *software* encapsulados em procedimentos, módulos, funções, e objetos sendo interconectados entre si, compondo assim uma arquitetura de *software* que será executada (WIKIPEDIA, 2005).

Antes de um sistema ser implantado, é necessário que passe por várias fases no decorrer do desenvolvimento da aplicação, sendo uma destas fases, a escolha de um banco de dados apropriado onde as informações serão processadas e armazenadas. Não basta somente um bom banco de dados, mas sim um banco que garanta a sincronização dos dados, e que estes não sejam redundantes, respeitando também a integridade dos dados (WIKIPEDIA, 2005).

Existem alguns tipos de modelos de dados para se adaptar a um sistema, sendo eles: hierárquico, rede, relacional e orientado a objeto.

Na década de 60, os computadores se tornaram parte efetiva do custo das empresas, juntamente com o crescimento da capacidade de armazenamento. Foram desenvolvidos dois principais modelos de dados: modelo em rede (CODASYL - Comitee *for Data* Systems *Language*) e o modelo hierárquico (IMS – Information Management System). O acesso ao BD é feito através de operações de ponteiros de baixo nível que unem (*link*) os registros. Detalhes de armazenamento dependiam do tipo de informação a ser armazenada; desta forma, a adição de um campo extra necessitaria de uma reescrita dos fundamentos de acesso/modificação do esquema (WIKIPEDIA, 2005).

Em 1976 o Dr. Peter Chen propôs o modelo Entidade-Relacionamento (ER) para projetos de banco de dados, possibilitando ao projetista concentrar-se apenas na utilização dos dados, sem se preocupar com estrutura lógica de tabelas.

Na década de 80, a comercialização de sistemas relacionais começou a ser dar entre as organizações, e a Linguagem Estruturada de Consulta – SQL (Structured Query Language) se tornou um padrão mundial.

Na década de 90, com crise econômica nas indústrias, o modelo cliente-servidor passou a ser uma regra para futuras decisões de negócio e viu-se que o desenvolvimento de ferramentas de produtividade como Excel/Access (Microsoft) e ODBC, também foi assinalado como o início dos protótipos de ODBMS (Object *Database Management Systems*) (WIKIPEDIA, 2005).

No meio da década de 90 surgiu a Internet o que fez aumentar o interesse pela tecnologia Web/BD, com o uso de soluções de código aberto (*open source*).

Na década de 80, havia surgido no mercado o banco de dados orientado a objeto, existindo atualmente no mercado mais de dez empresas de banco de dados orientado a objeto comercializando produtos com tais características.

Com a estimativa de aumento, o modelo relacional está agregando recursos da orientação a objeto em seus produtos, originando assim, uma arquitetura objeto relacional. Esses dois tipos de arquitetura tendem a prosseguir juntos, proporcionando assim poderosos recursos orientados a objeto (FRB, 2005a).

Os bancos de dados orientados a objetos integram o banco de dados à tecnologia de orientação a objetos. Tem a necessidade de realizar manipulações complexas para os bancos de dados existentes e uma nova geração de aplicações de bancos de dados, geralmente necessita mais diretamente de um banco de dados orientado a objeto (FRB, 2005b).

Por outro lado, aplicações de linguagens orientadas a objeto e sistemas, estão exigindo capacidades especiais de bancos de dados, tais como continuidade, simultaneidade e transações dos seus ambientes. Estas necessidades estão levando à criação de sistemas poderosos, chamados Bancos de Dados Orientados a Objeto (FRB, 2005b).

Este tipo de desenvolvimento teve origem na combinação de idéias dos modelos de dados tradicionais e de linguagens de programação orientada a objetos.

No SGBDOO, a noção de objeto é usada no nível lógico e possui características não encontradas nas linguagens de programação tradicionais, como operadores de manipulação de estruturas, gerenciamento de armazenamento, tratamento de integridade e persistência dos dados (FRB, 2005b).

O SGBDOO possui um módulo chamado de gerenciador de recuperação que administra as técnicas de recuperação de falhas de transação, falhas no sistema, e as falhas no meio.

Uma das estruturas mais utilizadas para o gerenciamento de recuperação é o log, que é utilizado para registrar e armazenar as imagens anteriores e posteriores dos objetos atualizados.

A idéia do banco de dados orientado a objeto surgiu para um simples suporte à programação orientada a objeto, ou seja, desistindo assim de tabelas e solicitando d um depósito para aquilo que eles chamavam de dados permanentes, ou seja, dados que permanecem mesmo depois que um processo é encerrado (FRB, 2005b).

Neste modelo, os dados só podem ser acessados através dos métodos que entrarão em ação no momento em que recebem uma solicitação ou uma invocação.

Pode-se dizer, então, que a orientação a objetos corresponde à organização de sistemas como uma coleção de objetos que integram estruturas de dados e comportamento. Além desta noção básica, a abordagem em questão inclui alguns conceitos, princípios e mecanismos básicos que a diferenciam das demais, sendo eles: Abstração, Objeto, Encapsulamento, Classes, Heranças, Métodos, Mensagens, Polimorfismo (FRB, 2005b).

Desta forma, considerando a evolução dos SGBDOO, diversas empresas disponibilizam suas soluções nesta plataforma, sendo que o presente trabalho apresenta um comparativo entre 9 diferentes SGBDOO, sendo direcionado, principalmente, para pessoas, empresas e universidades que necessitam de informações especificas com relação aos diversos SGBDOO encontrados no mercado para avaliaçãode suas características e funcionalidades. Dos 9 SGBDOO estudados e detalhados, 4 foram eleitos para um estudo comparativo abordando suas principais características, como Arquitetura, Modelo de Dados, Evolução de Esquema, Acesso Multidimensional e Arquitetura WEB. O presente texto inicia com um breve levantamento histórico sobre SGBDOO, desde a década de 60, com os modelos em rede e hierárquicos, passando pelas décadas de 70 e 80, com os modelos relacionais, até os dias atuais destacando os modelos objeto relacional e orientado

a objetos. Na Secção 2 são descritas as principais características dos 9 SGBDOO estudados, sendo eles ITASCA, GEMSTONE, VERSANT, OBJECTIVITY/DB, ONTOSDB, CACHÉ, JASMINE, OBJECTSTORE e O2. Na Secção 3 é apresentado o estudo comparativo dos SGBDOO CACHÉ, OBJECTIVITY/DB, ITASCA e OBJECTSTORE. Finalmente, na Secção 4 são expressas as conclusões e sugeridos os trabalhos futuros.

PRINCIPAIS SGBDOO ESTUDADOS

Os SGBDOO começaram a se tornar produtos comercialmente viáveis, pelo motivo de apresentar um alto desempenho, serem confiáveis e também se adaptarem a outros modelos de dados, tratando-se de consulta. Hoje algumas empresas já utilizam este modelo de dados para questões de gerenciamento das informações. Por este motivo iniciou-se uma pesquisa com alguns SGBDOO.

Com inúmeras pesquisas envolvendo bancos de dados, 9 foram então estudados, detalhando suas principais funcionalidades e, 4 de maior destaque foram comparados quanto a algumas de suas principais características.

Existem hoje vários tipos de SGBDOO, tais como: ITASCA, GEMSTONE, VERSANT, OBJECTIVITY/DB, ONTOS, CACHÉ, JASMINE, OBJECT STORE, O2, entre outros.

ITASCA

É baseado em uma série de protótipos para banco de dados orientados a objetos ORION, sendo executado sob plataforma UNIX e podendo ser utilizados: C++, C, CLOS ou *Common Lisp*. As pesquisas tiveram início no ano de 1985, pela MCC (*Microelectronics and Computer Techniology Corporation*) e Laboratório de Sistemas Distribuídos, em Austin, no Texas. Entretanto, em 1995 a *IBEX Corporation* adquiriu o ITASCA, Sistema Gerenciador de Bancos de Dados para Objetos Distribuídos – estendendo o protótipo para um produto comercial (ITASCA, 2001).

Seu modelo de dados suporta os princípios fundamentais de orientação a objetos, como: abstração de dados, encapsulamento, herança, identificadores de objetos e classes, incorporando um ambiente persistente e compartilhado. Cada objeto tem um identificador único, sendo que os atributos representam o estado e os métodos do comportamento. As classes apresentam objetos que compartilham os mesmos métodos e atributos e a definição do banco de dados tem como resultado uma hierarquia de classes, de forma que, subclasses podem ser derivadas de classes existentes, herdando todos os métodos e atributos, sendo permitida herança múltipla (ITASCA, 2001).

Este modelo oferece um mecanismo especial para suporte automático de versões de instâncias, onde os objetos modificados são mantidos no banco de dados, estando disponíveis para atualizações e consultas. Lembrando que as versões podem ser instâncias, somente, de classes declaradas como versionáveis. O ITASCA permite, ainda, operações para o controle de versões nos domínios das aplicações, tais como: *making, promoting, demoting, checking out* e *checking in*.

O ITASCA apresenta um mecanismo dinâmico para modificar o esquema, dando possibilidade de tratar as alterações do esquema como transações normais, enquanto o banco de dados permanece em execução. Com isso, cada mudança no esquema conduz o banco de dados de um estado estável para outro igualmente estável, sem que sua estrutura seja abalada (ITASCA, 2001).

Apresenta um mecanismo amplo e bem definido para o suporte de versões de objetos. Entretanto, não há possibilidade de aplicar esse mecanismo no tratamento de evolução de esquemas, impossibilitando a representação do histórico das modificações de esquemas (ITASCA, 2001).

Por outro lado, o modelo de dados do ITASCA suporta novas extensões com a finalidade de melhorar as capacidades de modelagem, introduzindo novos conceitos ao modelo orientado a objetos clássico. Por exemplo, o tratamento especial a objetos compostos, onde relacionamentos são aplicados nas classes, mantendo ligações inversas entre as classes e suas classes componentes, e vice-versa. No que se refere à evolução de esquemas, novas operações de modificação para objetos compostos são incluídas, ampliando o suporte à evolução disponível (ITASCA, 2001).

GEMSTONE

Além de sua característica evolutiva e o fato de trabalhar com diversas linguagens de dados, inclui baixa taxa de erros, recupera falhas no meio, no processo e no processador, enquanto mantém alta disponibilidade, tratando grandes quantidades de dados. Seu ponto principal é o *Class Graph*, que permite desenhar as estruturas de hierarquias de classes e os relacionamentos entre elas (GEMSTONE, 2005.)

Seu modelo de dados, trata apenas mecanismos de herança simples, ou seja, ao definir subclasses, dois principais tipos de modificação à definição herdada da superclasse podem ser exercidos, sendo elas: adição de atributos e adição e redefinição de métodos. Com isto, o GEMSTONE é classificado na categoria dos SGBD's no qual a persistência é uma propriedade ortogonal dos objetos, onde nem todos os objeto criados se tornam, automaticamente, persistentes (GEMSTONE, 2005).

Sua arquitetura, é dividida pelo *Gem Server*, onde o comportamento do objeto é executado, e também dentro do *server* estão tanto os objetos como as páginas de *Cache*. O *Stone* monitor que é o outro componente que integra a arquitetura, aloca novos identificadores de objetos e coordena o encerramento das transações. O GEMSTONE fazia uso do NFS (Network *File* System) para automatizar as réplicas dos bancos pela rede, o que foi alterado para promover uma melhor performance nos lançamentos seguintes. A máquina que possui a réplica assume as funcionalidades da máquina em falha, permitindo assim que o processamento seja retomado a partir do ponto em que foi interrompido. O ponto negativo desta versão é que a "coleta de lixo" era feita *off-line*, ou seja, o processamento tinha de ser interrompido. O mesmo tinha que acontecer com o *backup*. Esses problemas foram solucionados em versões posteriores (GEMSTONE, 2005).

O GEMSTONE usava a contagem de referência para identificar objetos temporários e assegurar que eles não seriam gravados no disco, não detectando ciclos desses objetos. Portanto, este controle de concorrência é feito pela combinação de concorrência otimista (implementado via paginação *shadow*) e travamento de objetos (GEMSTONE, 2005).

VERSANT

É um SGBDOO, de ambiente distribuído e suporte multiusuário, fornecido pela empresa VERSANT Object Technology, disponibilizando um conjunto de produtos, como: SGBD, ferramentas para desenvolvimento de aplicações e administração do banco de dados, interface para as linguagens de programação C++ e Smalltalk, SQL baseado na integração de dados legados, entre outros serviços e produtos, suportando também a evolução de esquemas *on-line*, através do comando: "sch2db –d database <schema_files> (VERSANT, 2001).

Com a execução desse comando, automaticamente, é examinado o esquema corrente com o objetivo de determinar as modificações necessárias, atualizando, as classes afetadas. O mecanismo de conversão é utilizado, tanto para evolução do esquema, quanto para adaptação da base de dados; nesse caso, a estratégia adotada consiste nas conversões adiadas, cujas instâncias são atualizadas somente quando requeridas pela aplicação (VERSANT, 2001).

Entretanto, considerando bancos de dados distribuídos, VERSANT permite a redefinição de classes. As instâncias das classes modificadas são atualizadas no instante em que são requeridas, consistindo em mecanismo de conversão adiada de banco de dados.

A bibliografia coletada a respeito do banco de dados VERSANT é um pouco limitada, impossibilitando a realização de uma análise mais rigorosa e aprofundada (VERSANT, 2001).

OBJECTIVITY/DB

É um SGBDOO comercial, baseado na linguagem C++, suportando outras linguagens, como: SmallTalk e JAVA. Sendo compatível com os conceitos definidos pelo grupo ODMG. Destaca-se por possuir uma arquitetura distribuída, cliente/servidor, que gerencia, corretamente os objetos distribuídos em ambientes heterogêneos e em múltiplos bancos de dados. É executado nas seguintes plataformas: Unix, Windows, NT, VMS, tendo como meta, prestar suporte a aplicações de grande escala, como, por exemplo, bancos de dados federados (OBJECTIVITY/DB, 2001).

Em relação ao modelo de Dados e Versões, os objetos são modelados utilizando estruturas na linguagem C++, e por classes adicionais. Neste modelo, um objeto simples é uma classe em C++, estando associado a um conjunto de métodos. O objeto complexo é representado por uma estrutura arbitrária, podendo sofrer modificações em seu tamanho, e um objeto composto é formando por uma rede de objetos relacionados, que atuam como objetos simples, através da propagação de comportamento para os objetos componentes. Os objetos simples podem, simultaneamente, atuar como componentes de vários objetos compostos, apresentando relacionamentos de 1:1, 1:n e n:n (OBJECTIVITY/DB, 2001).

O OBJECTIVITY/DB permite modificações no esquema do banco de dados, sendo elas: modificações lógicas, como por exemplo a troca de nome de uma classe, modificações em classes compostas, modificações em referências e associações, modificações em classes, como inclusão e exclusão e modificações nos relacionamentos de herança entre as classes existentes. Seu processo de evolução é o versionamento de classes (tipos). Toda vez que uma definição de classe é modificada, uma versão é derivada para a classe. Dessa forma, objetos podem ser instâncias de versões de tipos antigos ou novos. O processo de adaptação das instâncias vigentes no

banco de dados é realizado pela escrita de funções de conversão (métodos), instalados no sistema e acionados por ele (OBJECTIVITY/DB, 2001).

O principal destaque do banco de dados OBJECTIVITY/DB é a possibilidade de versionamento de classes durante o processo de evolução de esquemas, permitindo a representação do histórico das modificações.

Outro ponto importante é a definição de critérios *default*, tanto para o processo de evolução de esquemas, quanto para a adaptação da base de dados. Paralelamente, há a possibilidade de seleção, pelo usuário, de outras opções, ampliando a capacidade do processo de evolução e adaptação para o OBJECTIVITY/DB (OBJECTIVITY/DB, 2001).

ONTOSDB

É um produto orientado a objeto disponibilizado no mercado por ONTOS Inc., que fornece a persistência em uma base da classe, fornecendo também o objeto da classe que define os comportamentos para o armazenamento persistente dos objetos. Qualquer classe cujos exemplos devem ser armazenados de forma persistente, deve herdar do objeto da classe. Todos os exemplos de uma classe que herdam do objeto são persistentes sob o DB de ONTOS e são alcançados através dos ponteiros em uma aplicação. O DB de ONTOS fornece um número de funções que recuperam um objeto da base de dados e o colocam no espaço de memória de uma aplicação. As funções retornam um ponteiro ao objeto recuperado, podendo ser usado como um ponteiro a todo o objeto não persistente de C++. Através destes ponteiros, os membros dos dados podem ser alcançados e as funções do membro podem ser invocadas.

Todos os acessos do objeto aplicam-se a uma modalidade do fechamento do defeito ao objeto alcançado. A modalidade do fechamento do defeito é dependente do tipo de transação que a aplicação está executando atualmente (ONTOSDB, 2005).

Cada uma das funções que ativam objetos (isto é, acesso a objetos da base de dados da memória) fornece um parâmetro para o usuário cancelar a modalidade, travando do defeito.

Os objetos que são modificados pela aplicação devem ser explicitamente desativados (isto é, escrito para trás à base de dados). O DB de ONTOS fornece relações *deallocating*, objetos individuais e objetos agregados. Estas relações incluem parâmetros para especificar se os objetos irão permanecer no espaço de endereço da aplicação (ou se serão *deallocated* desse espaço de endereço) (ONTOSDB, 2005).

CACHÉ

É um banco de dados pós-relacional com o armazenamento de objetos, permitindo que seus dados sejam acessados de diversas maneiras, sendo elas: acesso via SQL/ODBC, acesso via objetos, ou acesso multidimensional. E possui as vantagens da tecnologia relacional em relação às consultas SQL, porém com maior poder de acesso, e sem as limitações de desempenho. Seu modelo multidimensional garante o desenvolvimento para aplicações complexas e transações para a Web através das CACHÉ *Server Pages* que são páginas em HTML ou XML padrão, que podem ser criadas ou modificadas com qualquer editor de texto ou ferramenta simple de criação de

páginas Web e interfaces gráficas. Com seu acesso otimizado via SQL padrão, objetos de alto desempenho e acesso multidimensional direto, oferece, assim, uma agilidade maior nas aplicações (CACHÉ, 2005).

Devido a essa característica, todos os dados são automaticamente acessíveis tanto como objetos quanto como tabelas. Esta tecnologia única significa que não há necessidade de se fazer sincronização entre as definições de objetos e tabelas, e nem há sobrecarga de processamento para a conversão entre as duas formas. A arquitetura unificada de dados aumenta a produtividade e a performance (CACHÉ, 2005).

Mesmo com seu eficiente núcleo multidimensional as consultas via SQL são rápidas e satisfatórias. O CACHÉ possui também um SQL *Gateway*, que permite que aplicações CACHÉ acessem dados armazenados em bases relacionais externas - o que é muito útil quando há necessidade de se integrar informações de diversas fontes (CACHÉ, 2005).

Um outro ponto positivo do cachê, é a conformidade com XML, que está se tornando o método preferido para compartilhar dados entre diferentes aplicações. É possível usar objetos CACHÉ como uma representação direta de documentos XML e vice-versa (CACHÉ, 2005).

O CACHÉ, além de ser um banco de dados, é rápido em desenvolvimento para Web, porque as aplicações podem ser programadas usando quaisquer ferramentas simples como um editor de texto. Torna-se rápido, porque com o CACHÉ, as páginas CSP herdam todo o código necessário para o gerenciamento de sessão. O código de programação é derivado dos objetos de sistema fornecidos pela InterSystems; o desenvolvedor escolhe o nível de segurança desejado para a sessão, e o CACHÉ se encarrega do resto (CACHÉ, 2005).

As páginas CSP (CACHÉ *Server Pages*) são executadas no servidor de dados, junto aos dados que as páginas irão acessar. Ou seja, a lógica de negócios e os dados estão estreitamente acoplados, tornando a comunicação muito rápida. Outra vantagem importante desta abordagem é a maior escalabilidade. Já que o servidor Web não fica sobrecarregado processando a lógica de negócios, ele fica livre para lidar com mais requisições que chegam dos navegadores Web (CACHÉ, 2005).

JASMINE

É um SGBDOO que armazena estruturas de classes com suas instâncias, sendo manipulado por uma linguagem de consulta a objetos, chamada ODQL, oferecendo interfaces que permitem o desenvolvimento de códigos em C e C++. Inclui ainda, uma ferramenta de desenvolvimento de aplicações chamada de JASMINE Studio. Sendo um banco bem versátil, o JASMINE possui uma biblioteca de classes multimídia que define classes para gerenciamento de dados multimídia, contendo classes com definições para vários tipos de objetos multimídia comuns, tais como: vídeo, áudio, figuras e diversas classes de suporte (JASMINE, 2005).

Com sua arquitetura cliente/servidor, o JASMINE permite que um servidor suporte múltiplas bases de dados, sendo que cada cliente pode acessar bases múltiplas em vários servidores (JASMINE, 2005).

Além dos conceitos básicos de orientação a objeto, outros conceitos são importantes para sua utilização, sendo eles: Codqlie: interpretador que permite a execução de comandos

ODQL e de métodos definidos para as classes; Família de Classes: representa o local lógico onde serão armazenadas as classes; JASMINE Studio: ambiente de desenvolvimento e gerenciamento de classes e objetos, permitindo a criação e manipulação de classes e objetos e aplicações; ODQL: é a linguagem de criação e manipulação de classes e objetos; e *Store*: sendo o local físico onde serão armazenados as classes e os objetos criados no JASMINE. Um *store* pode compreender um ou mais arquivos e pode armazenar uma ou mais famílias de classes (JASMINE, 2005).

OBJECTSTORE

O OBJECTSTORE é SGBDOO para aplicações Web e outros ambientes de computação distribuída, tendo como características principais a segurança, manuseio de grandes coleções de dados, alta velocidade, suporte a transações complexas e gerenciamento em redes, entre outras. Suas aplicações podem ser utilizadas tanto no sistema de rede Internet, quanto Intranet, sendo inseridas no sistema pelos próprios desenvolvedores da aplicação ou por fontes de dados externas, como usuários que utilizam a aplicação, através de *browsers* de navegação. Também provê aplicações em diversas áreas, como: Telecomunicações, Finanças, Saúde e Ciência, Internet, Manufatura, e Documento/Imagem (OBJECTSTORE, 2001).

O OBJECTSTORE segue os padrões definidos pela ODMG, sendo o mesmo baseado nos princípios fundamentais de orientação a objetos como: conceito de identidade de objetos, classes, composição, herança, funções virtuais, *templates*, encapsulamento, entre outros, destacando o eficiente gerenciamento de relacionamentos. Com seu conceito de OID (*Object Identifier*), fornece OIDs embutidos. Todo controle é realizado, automaticamente, pelo (SGBD), como, por exemplo, o OID de um objeto permanece válido mesmo quando a referência a ele é trocada. Possui ainda uma capacidade de integração de dados com outros sistemas, sendo possível importar dados para o SGBD ou apenas realizar o acesso a esses dados, disponibilizando para isto, ferramentas apropriadas para realização desta integração de dados, como, por exemplo, o *Dbconnect* que faz uma integração transparente com SGBDs relacionais (OBJECTSTORE, 2001).

Quanto à evolução de esquemas, é feita pela modificação da semântica dos objetos, modificação do esquema durante uma aplicação, enfatizando, a atualização dos objetos presentes no banco de dados, sendo possível modificar o esquema, e, automaticamente, são realizadas as atualizações nas representações de todas as instâncias na base de dados, conforme as novas definições, garantindo a integridade das informações armazenadas. É através de uma ferramenta chamada *Ossevol*, que é feito o processo de evolução de esquemas, que realiza a modificação e atualiza fisicamente a estrutura e a base de dados, sendo de importância que o usuário execute manualmente procedimentos de *backup* para garantir o estado anterior no caso de falhas (OBJECTSTORE, 2001).

O OBJECTSTORE apresenta um mecanismo de evolução de esquema bem definido, específico e documentado, que possibilita o tratamento de evolução de esquema, através de operações individuais ou sob um pacote de operações, agilizando assim o processo e fornecendo amplas possibilidades de modificação (OBJECTSTORE, 2001).

Considerando mecanismos de versões, o processo de evolução de esquemas do OBJECTSTORE apresenta, sempre, apenas duas versões, tanto da estrutura do esquema, quanto da base de dados. O processo de evolução é realizado sobre a segunda versão e, depois de conclu-

ído com sucesso, a versão antiga é totalmente excluída. Essa técnica assegura a validade do processo de evolução, uma vez que as atualizações são, irreversivelmente, realizadas quando o processo é finalizado com sucesso; caso contrário é recusado.

Entretanto, o estado anterior às modificações não é conservado, impedindo retornar aos estados anteriores da estrutura e da base de dados (OBJECTSTORE, 2001).

O2

Iniciou-se em 1986, sendo resultado de um programa que durou cinco anos e no qual se decidiu desenvolver um sistema de banco de dados. Em dezembro de 1987 os primeiros protótipos do sistema foram demonstrados, sendo liberados para testes em universidades e indústrias somente em 1989. Neste mesmo ano, foi feita uma análise profunda do projeto, o que levou a mudanças significativas. Em 1991, com o projeto finalizado, foi criada a O2 *Technology*, que então passou a comercializar, dar suporte e investir pesadamente na tecnologia O2. Já em 1992, a O2 *Tecnology* se juntou ao grupo ODMG'93. Desde então, a O2 *Technology* tem estado comprometida com a melhoria e o suporte ao padrão OQL, sendo a primeira a desenvolver um interpretador e uma ligação com o C++ compatíveis com este padrão (UCRJO2, 2005).

O O2 é mais utilizado para aplicações que necessitam de gerenciamento de dados complexos e/ou multimídia, sendo eles: telecomunicações, sistemas de gerenciamento de documentos, sistemas de informação geográfica, aplicações financeiras. Para isto, possui algumas metas importantes como: aumentar a produtividade no desenvolvimento de aplicações através das fases de análise, projeto, implementação, teste e evolução, utilizando a tecnologia orientada a objeto; desenvolver aplicações de armazenamento e transação de grandes volumes de dados complexos e multimídia; permitir a usuários que já utilizem banco de dados orientados a objetos, migrar facilmente para o O2, já que este segue o padrão ODMG; e melhorar a qualidade de aplicações finais, em termos de performance, aparência, manutenibilidade e customização (UCRJO2, 2005).

O2Engine, que é a base do sistema, é responsável pelo armazenamento de objetos estruturados e multimídia, fazendo gerenciamento de disco (incluindo buferização, indexação, clusterização e I/O), distribuição, gerenciamento de transações, recuperação, segurança e administração de dados. Suas aplicações podem ser desenvolvidas em C, C++, *Smalltalk* ou com a linguagem de quarta geração O2C, suportando também a linguagem de consulta OQL, estabelecida como padrão pela ODMG (UCRJO2, 2005).

Com uma arquitetura cliente/servidor baseada em um servidor que trata somente de páginas não entendendo a semântica de objetos, as consultas e métodos são executados no cliente, concentrando a maior parte da complexidade do sistema na estação de trabalho e deixando o servidor livre para tarefas que somente ele pode executar, melhorando assim o desempenho.

Um cliente pode acessar dados distribuídos sob o controle de uma transação global. O ambiente cliente/servidor é heterogêneo, pois o cliente e o servidor podem estar em diferentes *hardwares* (UCRJO2, 2005).

No O2, o gerenciador de versões é responsável por administrar as versões de objetos mantidos na base de dados O2. Utiliza a noção de *Unit Version* (UV), que consiste em uma coleção arbitrária de objetos que terão suas versões controladas, e de um grafo de derivação que descreve a evolução da UV. Assim que um objeto é incluído em uma aplicação, pode-se

acessar a base de dados e determinar explicitamente que versão deseja utilizar. Caso não indique a versão desejada, o próprio sistema determina qual versão do objeto à aplicação se deve acessar (UCRJO2, 2005).

A qualquer momento, a aplicação pode derivar uma nova versão de outra, seja esta uma folha do grafo ou não. Duas versões de um mesmo objeto podem ser assinaladas por diferentes transações, sem haver conflito (UCRJO2, 2005).

ESTUDO COMPARATIVO

Para se ter uma noção de como são os bancos de dados orientados a objeto, foi abordado neste estudo o funcionamento de cada um deles, sendo realizado um comparativo sobre as principais características de quatro bancos pesquisados.

Devido à dificuldade de se encontrar materiais e os próprios bancos em versão *fre*, para estudo, não foi possível seguir um padrão igual de comparação para todos os bancos e também aplicar testes reais. Sendo assim, alguns deles tiveram o mesmo tipo de comparação e outros não. A Tabela comparativa exibe os padrões de características que foram comparados, dos seguintes bancos de dados: CACHÉ, OBJECTIVITY/DB, ITASCA, OBJECT STORE.

Quadro Comparativo (CACHÉ, OBJECTIVITY/DB, ITASCA, OBJECT STORE).

Detalhamento da Tabela

Características	CACHÊ	OBJECTIVITY/DB	ITASCA	OBJECT STORE
Arquitetura	É pós-relacional com acesso via SQL	Gerencia ambientes heterogêneos	É independente; não necessita de linguagem específica	Ambiente de computação distribuído e aplicação Web
Modelo de dados / versão	Por ser uma tecnologia única, não necessita de sincronismo entre objeto e tabela	Suporta o versionamento de objetos	Suporta o versionamento e oferece operação para controle de versão	Controle de versão automática. Integração de dados com outros sistemas
Evolução de esquema	----------	Suporta várias modificações no esquema de BD	Evolução dinâmica no esquema	Capacidade de atualizar o esquema durante a aplicação
Acesso multidimensional	Integra dados armazenados em outras bases multidimensionais	---------	---------	----------
Arquitetura *WEB* única	Rápido acesso e comunicação e escalabilidade	---------	---------	---------

Com base na Tabela Comparativa dos bancos de dados abordados, será relatado detalhadamente cada característica dos bancos, de forma que será possível compreender, por exemplo, o funcionamento de cada banco de dados para uma determinada aplicação.

CACHÉ

O CACHÉ, sendo um banco de dados pós-relacional e possuindo uma arquitetura Web única, tem como principal virtude aplicações para WEB, tornando-se um acesso rápido e possível.

Arquitetura: sua arquitetura é pós-relacional, isto é, possui as vantagens da tecnologia relacional porém, com poderoso acesso e um alto desempenho. Além de ser um banco de dados com acesso otimizado via SQL padrão, objetos de alto desempenho e acesso multidimensional direto, oferecendo assim uma agilidade maior nas aplicações

Modelo de dados/versão: permite que todos os dados sejam automaticamente acessívei, tanto como objetos quanto como tabelas; sendo assim, significa que não há necessidade de se fazer sincronização entre as definições de objetos e tabelas, e nem há sobrecarga de processamento para a conversão entre as duas formas.

Acesso multidimensional**:** permite que os desenvolvedores tenham controle absoluto sobre a forma como os dados são armazenados, sendo útil para aplicações de processamento transacional que precisam integrar dados armazenados em outras bases multidimensionais.

Arquitetura Web única: comunicação rápida; outra vantagem é a maior escalabilidade, já que o servidor Web não fica sobrecarregado no processamento, permanecendo livre para lidar com mais requisições que chegam dos navegadores Web.

OBJECTIVITY/DB

É um banco de dados orientado a objetos, comercial, baseado na linguagem C++, que suporta várias linguagens, sendo compatível com os conceitos definidos pelo grupo ODMG.

Arquitetura: possui uma arquitetura distribuída cliente/servidor, que gerencia os objetos distribuídos em ambientes heterogêneos e em múltiplos bancos de dados, possibilitando o versionamento de classes durante o processo de evolução de esquemas, permitindo a representação do histórico das modificações.

Modelo de dados e versão: suporta características de versionamento, com a presença de referências estáticas e dinâmicas aos objetos versionados, definidas através de métodos.

Evolução de esquema: permite várias possibilidades de modificações em seu esquema, sendo elas: lógicas, modificações em classes compostas, modificações em referências e associações, modificações em classes (como inclusão e exclusão), e modificações nos relacionamentos de herança entre as classes existentes.

ITASCA

O ponto fraco deste banco é que não há possibilidade de aplicar esse mecanismo no tratamento de evolução de esquemas, impossibilitando a representação do histórico das modificações de esquemas.

Arquitetura: o ITASCA possui uma boa arquitetura, que é independente e não necessita de linguagens específicas e apresenta um bom mecanismo para suporte de versões de objetos.

Modelo de dados/versão: suporta os princípios fundamentais de orientação a objetos e incorpora a esses princípios um ambiente persistente e compartilhado para um gerenciador de banco de dados distribuído. Fornece um suporte automático de versões de instâncias, onde os objetos modificados são mantidos no banco de dados, estando disponíveis para atualizações e consultas.

O ITASCA permite, ainda, operações para o controle de versões nos domínios das aplicações, tais como: making, promoting, demoting, checking out e checking in.

Evolução de esquema: Fornece um mecanismo dinâmico para modificar o esquema, possibilitando tratar as alterações como transações normais, enquanto o banco de dados per-

manece em execução. Com isto, são gerados alguns benefícios como: facilidade aos desenvolvedores, bem como agilidade e rapidez nos processos de testes e prototipação; redução dos custos de manutenção; permite ao sistema permanecer em execução durante todo processo de modificação; e não há necessidade de recompilação das aplicações, entre outros.

OBJECT STORE

É um banco de dados que dispõe de aplicações em diversas áreas, podendo ser utilizado, tanto no sistema de rede Internet, quanto Intranet.

Para esses tipos de aplicações, as novas informações podem ser inseridas no sistema pelos próprios desenvolvedores da aplicação ou por fontes de dados externas, como usuários que utilizam a aplicação através de *browsers* de navegação.

Arquitetura: o OBJECTSTORE possui uma boa arquitetura de ambiente computacional distribuída, servindo também para aplicações WEB.

Modelo de dados/versão: este modelo segue os padrões definidos pela ODMG, baseado nos princípios fundamentais de orientação a objetos. Fornece OIDs embutidos, ou seja, todo controle de versão é realizado automaticamente pelo SGBD.

Tem a capacidade de integração de dados com outros sistemas, importando os dados para o SGBD, ou apenas realizando o acesso a esses dados.

Evolução de esquema: é possível modificar o esquema e, automaticamente, são realizadas as atualizações nas representações de todas as instâncias na base de dados, conforme as novas definições, garantindo a integridade das informações armazenadas.

CONCLUSÕES

Com o comparativo dos bancos de dados abordados conclui-se que vem crescendo muito o desenvolvimento de aplicativos utilizando o modelo de banco de dados orientados a objeto.

Os quatro bancos que foram comparados apresentaram características muito importantes, como, principalmente, no controle de versão e no desenvolvimento de aplicativos Web.

O ObjectivityDB, por exemplo, é muito bom em relação ao versionamento de classes durante o processo de evolução de esquemas, e com isso, permite um histórico das modificações.

Já o banco de dados ITASCA tende ao melhoramento em relação à capacidade de modelagem, com a iniciação dos conceitos ao modelo orientado a objetos clássico.

O banco de dados CACHÉ e o OBJECTSTORE são ótimos para aplicativos Web.

O CACHÉ destaca-se por ser rápido e muito acessível, com múltiplos acessos a dados, sendo o principal o acesso multidimensional, obtendo-se nisto um controle rigoroso sobre os dados.

Já o OBJECTSTORE, além de servir para aplicações Web, é um banco bom, considerando-se que seu esquema possui um mecanismo de evolução muito bem organizado, viabilizando o tratamento de evolução de esquemas através de operações individuais e fornecendo amplas possibilidades de modificação.

Com base nas características destes bancos abordados destaca-se, entre eles, o OBJECTSTORE porque além de ser bom para aplicações Web, possui uma ótima organização em seu esquema.

REFERÊNCIAS

WIKIPEDIA. Engenharia de *Software:- GNU Free Documentation License*. Disponível em <http://pt.wikipedia.org/wiki/Engenharia_de_software>. Modificado em 18 de Maio 2005. Acessado em 11 de maio de 2005.

FRB - FACULDADE RUI BARBOSA. *Banco de Dados Orientado a Objeto: Conclusão*. Disponível em <http://www.ufpa.br/sampaio/curso_de_sbd/bdoo/apostila/final.html>. Acessado em 19 de maio de 2005a.

FRB - FACULDADE RUI BARBOSA. *Banco de Dados Orientado a Objeto: Introdução*. Disponível em <http://www.ufpa.br/sampaio/curso_de_sbd/bdoo/apostila/intro.html>. Acessado em 19 de maio de 2005b.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. *Evolução de Esquema em BDOO com Emprego de Versões – 8.1.2 ITASCA*. Disponível em <http://www.inf.ufrgs.br/~galante/files/qualifying.pdf>. Acessado em 25/05/2005 – Porto Alegre, 2001.

BANCO DE DADOS NÃO CONVENCIONAIS (DIM310). *2000.1 – Banco de Dados GEMSTONE*. Disponível em <http://www.dimap.ufrn.br/~marciaj/BDNC001.htm>. Acessado em 24/05/05.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. *Sistema Gerenciador em BDOO com Emprego de Versões – 8.1.5 SGBOO VERSANT*. Disponível em <http://www.inf.ufrgs.br/~galante/files/qualifying.pdf>. Acessado em 25/05/2005.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. *Sistema Gerenciador em BDOO com Emprego de Versões – 8.1.3 OBJECTIVITY/DB*. Disponível em <http://www.inf.ufrgs.br/~galante/files/qualifying.pdf>. Acessado em 25/05/2005.

Object-Oriented Database Management Systems Revisited - 5.2.1 Application Development Issues - BDOO Ontos. Disponível em <http://www.dacs.dtic.mil/techs/oodbms2/ontos.shtml>. Acessado em 05/05/05.

Linha de código – Tecnologia CACHÉ. Disponível em: <http://www.linhadecodigo.com.br/artigos.asp?id_ac=71&pag=2>. Publicado em 19/04/03. Acessado em – 29/05/05.

UNIVERSIDADE FEDERAL DE SÃO CARLOS. *Manual de Utilização do JASMINE - 1. JASMINE*. Disponível em -<http://www.recope.dc.ufscar.br/recope/download/bd/jasminemanual.pdf>. Consultado em 29/05/05.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL. *Sistema Gerenciador em BDOO com Emprego de Versões – 8.1.4 OBJECTSTORE*. Disponível em <http://www.inf.ufrgs.br/~galante/files/qualifying.pdf>. Acessado em 25/05/2005.

UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO. *Armazenamento em Banco de Dados Orientado a Objeto – 4.1.1 O2 – Histórico*. Disponível em <http://www.inf.puc-rio.br/~casanova/TrabalhosCSGBD/992/fernanda%20jardim%20trabalho%201%20-%20992.pdf>. Acessado em 29/05/05.