**UNIVERSIDADE DO EXTREMO SUL CATARINENSE – UNESC**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**PRISSILA GOMES HAHN**

# TRANSFORMAÇÃO DE ESQUEMAS DE BANCOS DE DADOS RELACIONAIS EM ORIENTADOS A OBJETOS

**CRICIÚMA, NOVEMBRO DE 2009**

**PRISSILA GOMES HAHN**

# TRANSFORMAÇÃO DE ESQUEMAS DE BANCOS DE DADOS RELACIONAIS EM ORIENTADOS A OBJETOS

Trabalho de Conclusão de Curso apresentado para obtenção do Grau de Bacharel em Ciência da Computação da Universidade do Extremo Sul Catarinense.

Orientadora: Profa. MSc. Cristiane Raquel Woszezenki

Co-Orientador: Prof. Esp. Fabricio Giordani

**CRICIÚMA, NOVEMBRO DE 2009**

**PRISSILA GOMES HAHN**

**Transformação de Esquemas de Bancos de Dados Relacionais em Orientados a Objetos**

Submetido ao corpo docente do Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense como um dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

**Profa. MSc. Ana Claudia Garcia Barbosa**
Coordenadora do Curso de Ciência da Computação

Banca Examinadora:

**Profa. MSc. Cristiane Raquel Woszezenki (UNESC)**
Orientadora

**Prof. Esp. Fabricio Giordani (UNESC)**
Co-Orientador

**Prof. MSc. Daniel Pezzi da Cunha (UNICRUZ – Cruz Alta/RS)**

**Prof. MSc. Paracelso de Oliveira Caldas (UNESC)**

*Aos meus pais, Eládio e Rosita, pela minha formação moral e por me proporcionarem a realização deste sonho.*

## AGRADECIMENTOS

Agradeço a todos que, de alguma forma, contribuíram para a realização deste trabalho. Em especial a Deus por minha saúde e força para jamais desistir de meus objetivos e superar os obstáculos.

Aos meus pais Eládio e Rosita, por acreditarem em minha capacidade e me apoiarem nos momentos difíceis.

A minha Orientadora Cristiane, meu Co-Orientador Fabricio e ao Professor Daniel Pezzi, pela dedicação e tempo disponibilizados, e desta forma, terem me direcionado ao caminho do sucesso.

Ao meu namorado Luiz Artur, pelo apoio, paciência e compreensão dedicados a mim.

Aos meus colegas de curso, pelos momentos de alegria compartilhados dentro e fora da sala de aula. Também aos colegas de trabalho, que sempre estiveram prontos para me ajudar no que fosse necessário.

## RESUMO

A grande utilização dos sistemas de bancos de dados por empresas dos mais variados segmentos demanda maneiras eficientes de tratar dados mais complexos. Ao longo do tempo o modelo relacional pode apresentar limitações para tal, tornando-se necessária a utilização dos bancos de dados orientados a objetos, por atenderem tal necessidade. Desta forma, faz-se necessário o estudo das diferenças entre projetos de bancos de dados relacional e orientados a objetos de forma a criar um método de transformação que facilite o processo de migração entre esquemas, bem como o torne mais rápido e eficaz. Objetivando solucionar o problema, são realizados os estudos necessários e, em seguida é criado e incorporado o método de transformação de esquemas relacionais para esquemas orientados a objetos à ferramenta de apoio à engenharia reversa desenvolvida por Samuel Brognoli. Com isto, esta ferramenta, além de gerar o modelo conceitual a partir de metadados extraídos de bancos de dados relacionais, também transforma o esquema relacional para um esquema orientado a objetos gerando os scripts correspondentes. Tal ferramenta está limitada aos sistemas gerenciados de bancos de dados Oracle e Firebird e sua transformação se dá para o padrão orientado a objetos ODMG e para o sistema gerenciador de bancos de dados Caché.

.



**Palavras-chave:** Bancos de Dados Relacionais, Bancos de Dados Orientados a Objetos, Transformação de Esquemas de Bancos de Dados, Padrão ODMG.

**ABSTRACT**

The great usage of database systems by firms of the most different sectors demands efficient ways of treating more complex data. In the course of time, the relational model can present limitations concerning this, which makes the use of object-oriented databases necessary, in order to comply with this necessity. This makes it necessary to study the differences between projects of relational and object-oriented databases with the intention of creating a transformation method, which facilitates the process of migration between schemes and also makes it faster and more effective. Aiming at solving this problem, the necessary studies are made and then, the method of transformation of relational schemes into object-oriented schemes is created and adapted, using the supporting tool of reverse engineering, developed by Samuel Brognoli. Thus, this tool not only creates the conceptual model from meta-data that are extracted from relational schemes, but also transforms the relational scheme into object-oriented scheme, creating the corresponding scripts. Such a tool is limited to the database management systems Oracle and Firebird and its transformation works for the object-oriented standard ODMG and for the database management system Caché.

**Key words**: Relational Databases, Object-Oriented Databases, Transformation of Database Schemes, ODMG Standard.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE QUADROS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| BD | Bancos de Dados |
| BDOO | Bancos de Dados Orientados a Objetos |
| BDR | Bancos de Dados Relacionais |
| CAD | *Computer-Aided Design* |
| CAM | *Computer-Aided Manufacturing* |
| CDL | *Class Definition Language* |
| COS | *Caché Object Script* |
| DDL | *Data Definition Language* |
| DML | *Data Manipulation Language* |
| E-R | Entidade Relacionamento |
| JDBC | *Java Database Connectivity* |
| JDK | *Java Development Kit* |
| LPOO | Linguagens de Programação Orientadas a Objetos |
| ODL | *Object Definition Language* |
| ODMG | *Object Data Management Group* |
| OID | *Object Identifier* |
| OO | Orientação a Objetos |
| OQL | *Object Query Language* |
| SGBD | Sistemas Gerenciadores de Banco de Dados |
| SGBDOO | Sistema Gerenciador de Banco de Dados Orientado a Objetos |
| SQL | *Structured Query Language* |
| UML | *Unified Modeling Language* |

**SUMÁRIO**

# 1 INTRODUÇÃO

A necessidade das organizações pelo crescente volume de dados a serem armazenados demandou o desenvolvimento dos bancos de dados, que são gerenciados por sistemas conhecidos como sistemas gerenciadores de bancos de dados, de forma a organizarem e proverem informações relativas a estes dados armazenados (CHEN, 1990; HARRINGTON, 2002; SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Dentre os bancos de dados atuais está o relacional, que começou a ser utilizado nos anos 70 e até hoje é o mais amplamente utilizado. Entretanto este modelo apresentou limitações ao longo do tempo ao tentar tratar a complexidade de armazenamento de grandes itens de dados e como solução surgiram os bancos de dados orientados a objetos.

O projeto de banco de dados relacional pode-se dizer que é um processo de descoberta de como representar objetos do mundo real nos limites das tabelas, de forma a atingir um bom desempenho e garantir a integridade dos dados. O projeto de banco de dados orientados a objetos se torna, na maioria das vezes, parte do projeto geral da aplicação. As classes de objetos usadas pelas linguagens de programação são as mesmas usadas no modelo de dados (BOSCARIOLI et al, 2006).

Nos bancos de dados relacionais os dados são armazenados em tabelas e relacionados por chave, já nos bancos de dados orientados a objetos cada objeto possui um estado e comportamentos e é diferenciado por um identificador de objetos (HARRINGTON, 2002; HEUSER, 2001). Diferentemente do modelo de dados relacional, este modelo dá suporte a herança, polimorfismo e encapsulamento (BOSCARIOLI et al, 2006).

Diante das vantagens do modelo orientado a objetos é realizado um estudo de seu padrão e das diferenças entre este e o modelo relacional para, a partir dos metadados extraídos por meio de engenharia reversa, gerar o esquema correspondente para bancos de dados

orientados a objetos, facilitando a migração de bancos de dados relacionais para uma nova plataforma de implementação.

Os metadados utilizados são extraídos utilizando a ferramenta para apoio à engenharia reversa de banco de dados relacional desenvolvida por Samuel Brognoli em 2008, utilizando linguagem JAVA, que representa-os em forma de diagrama Entidade Relacionamento.

## 1.1 OBJETIVO GERAL

Transformar esquemas de bancos de dados relacionais para esquemas orientados a objetos a partir de metadados extraídos por ferramenta de Engenharia Reversa.

## 1.2 OBJETIVOS ESPECÍFICOS

A fim de atingir o objetivo geral deste trabalho destacam-se os seguintes objetivos específicos:

a) estudar o projeto de bancos de dados orientados a objetos;
b) analisar as diferenças entre o projeto de banco de dados relacional e o orientado a objetos;
c) compreender a estrutura de armazenamento dos metadados extraídos com a ferramenta de engenharia reversa desenvolvida por Samuel Brognoli em 2008;
d) elaborar o método de transformação do esquema relacional para o esquema orientado a objetos;
e) implementar o método de transformação do modelo relacional para o modelo orientado a objetos.

## 1.3 JUSTIFICATIVA

A crescente necessidade de maneiras eficazes de tratar dados mais complexos e as limitações impostas pelo modelo de dados relacional motivou o desenvolvimento dos bancos de dados orientados a objetos, que geram grande interesse pela facilidade em efetuar alterações nas implementações de acordo com as mudanças nos requisitos dos sistemas.

Dessa forma, decidiu-se incorporar a funcionalidade da transformação de um esquema relacional para um esquema orientado a objetos à ferramenta de apoio à engenharia reversa desenvolvida por Samuel Brognoli. Com isto, esta ferramenta, além de gerar o modelo conceitual a partir de metadados extraídos de bancos de dados relacionais, também estará apta a transformar o esquema relacional para um esquema orientado a objetos, conforme mostra a Figura 1.

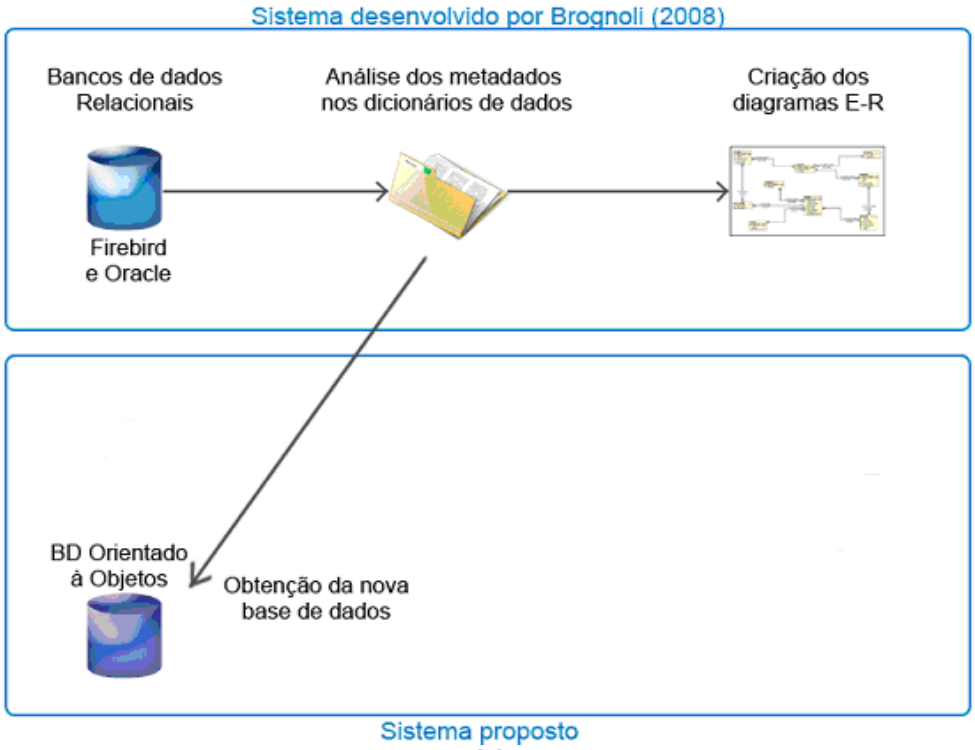


Figura 1. Visão geral do processo proposto.

Para tanto, se faz necessário o estudo das diferenças entre projetos de bancos de dados relacional e orientados a objetos de forma a criar um método de transformação que facilite o processo de migração entre esquemas, bem como o torne mais rápido e eficaz.

Estudar as diferenças entre projeto de bancos de dados relacional e banco de dados orientado a objetos e agregar a transformação entre esquemas será muito útil, considerando que a utilização de banco de dados é de fundamental importância para as organizações e mais ainda a sua adaptação as novas necessidades que surgem com o passar do tempo, como por exemplo, a migração entre bancos de dados.

## 1.4 ESTRUTURA DO TRABALHO

Este trabalho possui seis capítulos, sendo o primeiro composto pela introdução, objetivos e justificativa.

A pesquisa realizada sobre bancos de dados orientados a objetos, com seus padrões atuais, suas vantagens e desvantagens em relação aos bancos de dados relacionais e a descrição dos mais utilizados é abordada no Capítulo 2.

No Capítulo 3 são apresentadas as diferenças entre o projeto de banco de dados relacional e de banco de dados orientado a objetos, que são fundamentais para o entendimento e elaboração do método de transformação do esquema relacional para o esquema orientado a objetos.

O Capítulo 4 apresenta alguns trabalhos correlatos nos quais foram utilizadas diferentes metodologias para a realização da transformação entre esquemas de bancos de dados.

O Capítulo 5 descreve brevemente o trabalho proposto, as características da ferramenta de engenharia reversa utilizada bem como as alterações necessárias nesta para

incluir o processo de transformação de modelos. Este capítulo apresenta ainda as etapas do processo, os recursos utilizados, um exemplo de utilização da ferramenta e os resultados obtidos.

# 2 BANCOS DE DADOS ORIENTADOS A OBJETOS

Os Bancos de Dados Orientados a Objetos (BDOO) são assim denominados por permitirem acesso direto aos dados por meio de Linguagens de Programação Orientadas a Objetos (LPOO), que são cada vez mais utilizadas no desenvolvimento de aplicações de software (SILBERSCHATZ; KORTH; SUDARSHAN, 2006). Este modelo de Banco de Dados (BD) foi proposto para atender as necessidades de aplicações cada vez mais complexas (ELMASRI; NAVATHE, 2002).

As aplicações mais recentes possuem características diferentes das convencionais, pois incluem estruturas complexas para objetos, transações de duração mais longa e armazenamento de imagens e grandes itens de texto (ELMASRI; NAVATHE, 2002). Dessa forma, a utilização de BDOO se deu a partir do momento em que os modelos relacionais apresentaram limitações ao tratar tais complexidades (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

A Orientação a Objetos (OO) proporciona maior flexibilidade para lidar com exigências de sistemas, por exemplo, de projetos de *design* de engenharia e arquitetura como *Computer-Aided Design* (CAD) e *Computer-Aided Manufacturing* (CAM), experiências científicas, telecomunicações, sistemas de informações geográficas e multimídia (ELMASRI; NAVATHE, 2002; MARTIN, 1994).

## 2.1 PARADIGMA DE ORIENTAÇÃO A OBJETOS

A OO surgiu a partir das LPOO, no entanto o conceito de OO também é utilizado atualmente nas áreas de BD, Engenharia de Software, Bases de Conhecimento, Inteligência

Artificial e Sistemas Informatizados (ELMASRI; NAVATHE, 2002; MARTIN; ODELL, 1995).

As LPOO têm suas origens da linguagem SIMULA de 1960, onde nesta, uma classe define a estrutura interna de um objeto. Em seguida, pesquisadores propuseram agregar à OO o conceito de tipo de dados abstrato, de forma a esconder a estrutura interna de um objeto e especificar as possíveis operações externas que podem ser aplicadas a ele, dando origem ao conceito de encapsulamento (ELMASRI; NAVATHE, 2002; KHOSHAFIAN, 1994; MARTIN; ODELL, 1995).

Em 1970 foi desenvolvida a linguagem de programação SMALLTALK, sendo uma das primeiras linguagens de programação projetadas para ser OO, que incorporou conceitos como herança e transmissão de mensagens. A linguagem de programação $C^{++}$, ao contrário da SMALLTALK, incorporou conceitos de OO à linguagem já existente. Dentre as LPOO mais utilizadas atualmente pode-se citar a linguagem JAVA (ELMASRI; NAVATHE, 2002; KHOSHAFIAN, 1994).

Alguns dos conceitos de LPOO também são aplicados a modelos de dados orientados a objetos para bancos de dados. No entanto, BD possuem algumas peculiaridades, requerendo assim, alguns conceitos específicos. Em LPOO os objetos existem somente durante a execução de um programa, por isso, são chamados de objetos transientes. Ao contrário em BDOO a existência de um objeto pode ser estendida, de forma que eles sejam permanentemente armazenados, ou seja, persistentes, podendo ser posteriormente recuperados após a terminação do programa (ELMASRI; NAVATHE, 2002).

### 2.1.1 Objetos, Identidade e Classes

Um objeto geralmente é composto por estado (valor) e comportamento (operações), tendo um identificador único para cada objeto que é gerado pelo sistema,

chamado de O*bject Identifier* (OID) (ELMASRI; NAVATHE, 2002). “O estado do objeto depende do valor de cada uma de suas propriedades e o comportamento é especificado pelas operações que podem ser executadas pelo objeto, possivelmente, modificando o estado do mesmo.” (BOSCARIOLI et al, 2006, p. 3).

Um OID serve para identificar um objeto em uma base de dados inteira e deve ser imutável, ou seja, o valor do OID não pode ser alterado, de forma a preservar a identidade do objeto. O identificador deverá ser utilizado apenas uma vez, mesmo que o objeto seja removido o OID deste não deve ser reutilizado para outro objeto (BOSCARIOLI et al, 2006).

Cada objeto é instanciado a partir de uma classe que armazena as características em comum entre objetos, a mesma estrutura interna, os mesmos atributos e métodos (BOSCARIOLI et al, 2006).

### 2.1.2 Herança

A herança é um mecanismo de especialização de classes, que permite a uma determinada classe possuir características particulares e ainda herdar as características de outra classe. A classe geral é conhecida como superclasse, já as classes especializadas são chamadas de subclasses (KHOSHAFIAN, 1994; SILVA, 2002).

Há duas formas de herança, a simples e a múltipla (SILVA, 2002):

a) **herança simples**: uma classe herda as características de uma única classe;

b) **herança múltipla**: uma classe herda as características de mais de uma classe.

Exemplificando-se o conceito de herança, tem-se a Figura 2, que apresenta duas classes, *Caminhão* e *Ônibus*, com suas respectivas características.

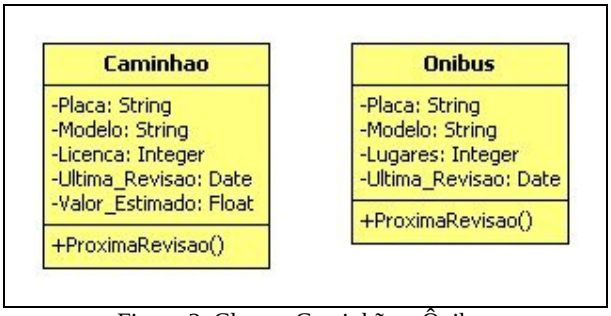


Figura 2. Classes Caminhão e Ônibus
Fonte: Adaptado de BOSCARIOLI, C. et al (2006)

Uma vez que ambas as classes possuem características em comum pode-se criar uma classe *Veículo*, que contenha tais características e somente as que são particulares de cada classe sejam mantidas nas mesmas como representado na Figura 3.

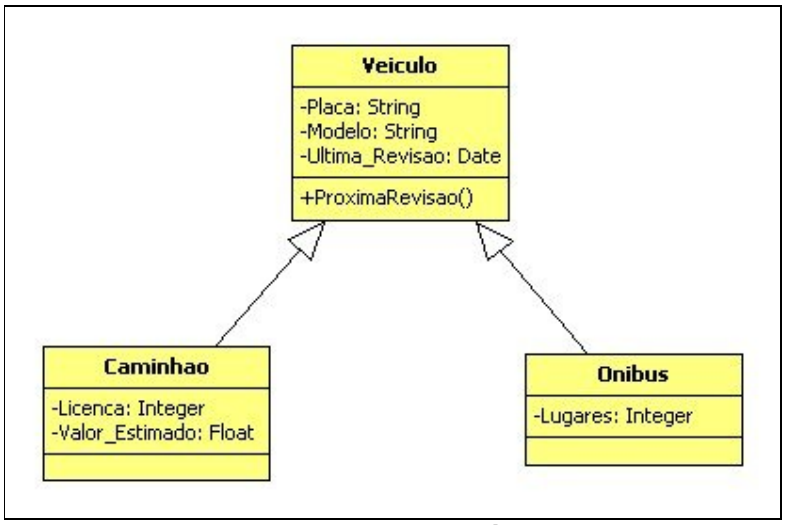


Figura 3. Herança simples.
Fonte: Adaptado de BOSCARIOLI, C. et al (2006)

A subclasse *Ônibus* ainda pode herdar as características de outra classe, caracterizando a herança múltipla conforme Figura 4.

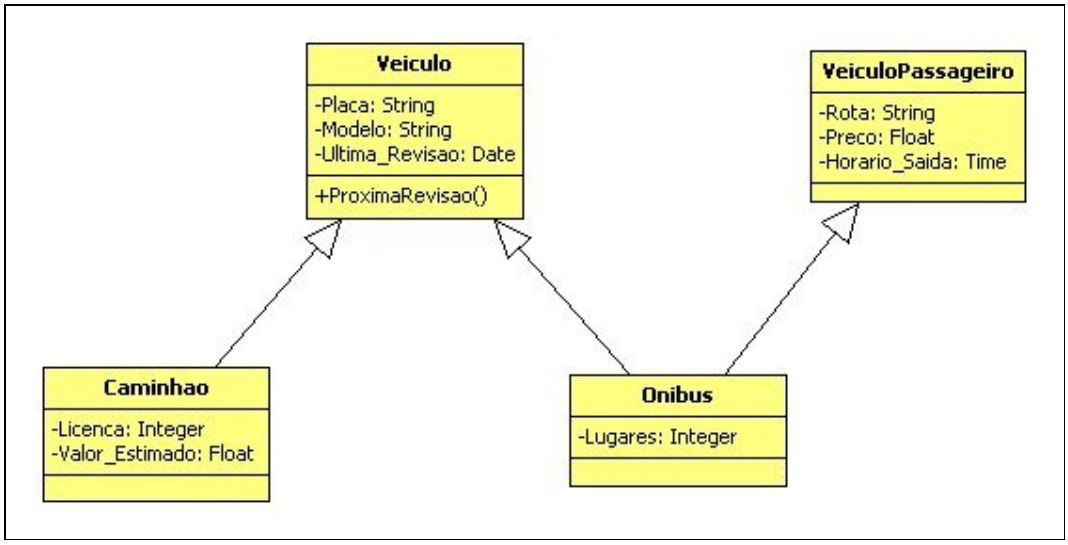


Figura 4. Herança múltipla.
Fonte: Adaptado de BOSCARIOLI, C. et al (2006)

### 2.1.3 Polimorfismo

O polimorfismo possibilita que subclasses sobrescrevam métodos herdados de suas superclasses, desde que o método tenha a mesma assinatura, de forma que o novo comportamento implementado melhor se adeque às necessidades da subclasse. Para identificar qual método deve ser executado (se é o método definido na superclasse ou o método sobrescrito na subclasse) ele verifica para qual tipo de objeto (tipo da superclasse ou tipo da subclasse) o método está sendo chamado (BOSCARIOLI et al, 2006; SILVA, 2003).

Considerando-se, por exemplo, uma superclasse *Veículo* que tenha um método chamado *Concertar* e suas subclasses *Caminhão* e *Ônibus* que herdam este método, estas subclasses podem especializá-lo de modo que quando o método for invocado o sistema decida qual implementação deverá usar, conforme o tipo do objeto que o está invocando (BOSCARIOLI et al, 2006).

### 2.1.4 Encapsulamento

O encapsulamento consiste em ocultar a estrutura interna do objeto, de forma que esta só se torne acessível por meio de operações predefinidas (ELMASRI; NAVATHE, 2002; MARTIN; ODELL, 1995).

O encapsulamento de todos os objetos em BD é muito rígido, para tanto, pode-se dividir a estrutura de um objeto em atributos visíveis e ocultos. Os atributos visíveis podem ser acessados diretamente, por operadores externos ou por linguagens de consulta de alto nível. No entanto os ocultos são completamente encapsulados e só podem ser acessados utilizando operações predefinidas (ELMASRI; NAVATHE, 2002).

### 2.1.5 Objetos Complexos

Objetos complexos são assim denominados por necessitarem de uma grande área de armazenamento e não são tipos de dados padrões fornecidos pelos Sistemas Gerenciadores de Bancos de Dados (SGBD) convencionais. Sendo o objeto muito grande, o SGBD pode recuperar parte dele e fornecê-lo à aplicação antes que todo o objeto seja recuperado (ELMASRI; NAVATHE, 2002; MARTIN; ODELL, 1995).

Os tipos principais de objetos complexos são os não-estruturados e os estruturados. Objetos complexos não-estruturados permitem a um SGBD armazenar e recuperar grandes objetos, como *imagens bitmap* e *strings de texto longo*. O SGBD não conhece a estrutura de tais objetos, apenas as aplicações que os utilizam têm a possibilidade de interpretá-los (ELMASRI; NAVATHE, 2002).

Objetos complexos estruturados diferem dos não-estruturados porque sua estrutura é definida por meio da aplicação repetida dos construtores de tipo do Sistema

Gerenciador de Banco de Dados Orientado a Objetos (SGBDOO), sendo tal estrutura definida e conhecida pelo SGBDOO (ELMASRI; NAVATHE, 2002).

## 2.2 PADRÕES DE PROJETO DE BANCOS DE DADOS ORIENTADOS A OBJETOS

A definição de padrões de BDOO é de grande importância, por possibilitar a portabilidade de aplicações de BD, ou seja, uma aplicação pode ser executada em diferentes SGBDOO, necessitando de mínimas modificações (ELMASRI; NAVATHE, 2002).

Um padrão de BDOO foi proposto em 1993 por um consórcio de vendedores de SGBD chamado *Object Data Management Group* (ODMG), tal padrão é conhecido como ODMG 1.0, tendo como sua última revisão o ODMG 3.0 de 2000 (CATTELL; BARRY, 2000, tradução nossa).

O padrão ODMG é composto principalmente por: modelo de objetos, Linguagem de Definição do Objeto ou O*bject Definition Language* (ODL) e Linguagem de Consulta ao Objeto ou *Object Query Language* (OQL) (CATTELL; BARRY, 2000, tradução nossa). Estas duas últimas, ODL e OQL, são respectivamente similares à *Data Definition Language* (DDL) e à *Data Manipulation Language* (DML), que compõem a *Structured Query Language* (SQL), utilizada em Bancos de Dados Relacionais (BDR) (GELATTI, 2002).

### 2.2.1 Modelo de Objetos

O modelo de objetos tem como blocos básicos os objetos e os literais. Objetos diferem principalmente de literais por terem um identificador e um estado; um literal tem apenas um valor. Tanto o valor de um objeto como o valor de um literal pode ter uma estrutura complexa. Contudo, o valor de um objeto pode ser modificado caso seu estado mude

ao longo do tempo e, ao contrário, o valor de um literal não se altera. O Quadro 1 apresenta as características de objetos e literais (ELMASRI; NAVATHE, 2002; SANTANA; NUNES; OLIVEIRA, 2008).

| ***Objeto*** | |
|---|---|
| ***Identificador*** | Único em todo o sistema<br>Obrigatório a todos os objetos |
| ***Nome*** | Único em todo o sistema<br>De uso opcional<br>Pode ser utilizado para o sistema localizar o objeto |
| ***Tempo de Vida*** | Especifica se o objeto é persistente ou transiente |
| ***Estrutura*** | Define como o objeto é construído por meio de construtores de tipo<br>Especifica se o objeto é atômico ou de coleção |
| ***Literal*** | |
| ***Valor*** | ***Tipos de literal*** |
| Estrutura simples ou complexa | *Atômico:* corresponde aos tipos de dados simples (long, long long, short, unsigned long, unsigned short, float, double, boolean, octet, char, string e enum) |
| | *Estruturado:* criado usando o construtor Struct (Date, Interval, Time e Timestamp) |
| | *De coleção:* criado com os construtores Set <t>, Bag <t>, List <t>, Array <t>, Dictionary <k, t> onde t é o tipo de objetos ou valores na coleção e k é o tipo da chave |

Quadro 1. Características de objetos e literais
Fonte: ELMASRI, R; NAVATHE, S. (2002)

O modelo de objetos define que todos os objetos herdam uma interface básica da interface Objeto. As operações básicas herdadas são: (i) *copy*, que cria uma nova cópia do objeto; (ii) *delete* que exclui o objeto e; (iii) *same* que compara a identidade do objeto com um outro objeto. As interfaces não são instanciáveis (ELMASRI; NAVATHE, 2002).

As interfaces, por não serem instanciáveis, são utilizadas para especificar operações abstratas que podem ser herdadas por classes, que são instanciáveis, ou até mesmo outras interfaces, isso é chamado de herança de comportamento, que utiliza ":" para especificá-la (ELMASRI; NAVATHE, 2002).

Outro tipo de herança utiliza a palavra EXTENDS para especificá-la. Esta é utilizada para herdar estado e comportamento entre classes. A herança múltipla com

EXTENDS não é permitida, no entanto, ela é possível na herança de comportamento (ELMASRI; NAVATHE, 2002).

No padrão ODMG um objeto definido pelo usuário e que não seja um objeto de coleção – que herda a interface básica Coleção – é um objeto atômico. Um objeto atômico é definido como uma classe e tem especificadas suas propriedades e operações. As propriedades são seus atributos e relacionamentos (ELMASRI; NAVATHE, 2002).

Um atributo define alguns aspectos de um objeto. Um relacionamento é definido utilizando a palavra-chave *relationship* (relacionamento), e pode ser declarado como inverso utilizando-se da palavra-chave *inverse* (inverso), que permite ao sistema manter automaticamente a integridade referencial do relacionamento (ELMASRI; NAVATHE, 2002).

### 2.2.2 *Object Definition Language*

A ODL utiliza o modelo de objetos para criar um esquema de BDOO, ou seja, para criar as especificações de objetos (classes e interfaces). Ela é independente que qualquer linguagem de programação (ELMASRI; NAVATHE, 2002).

A Figura 5 apresenta a notação gráfica utilizada para representar esquemas de interface, classe, relacionamentos e herança em ODL (ELMASRI; NAVATHE, 2002).

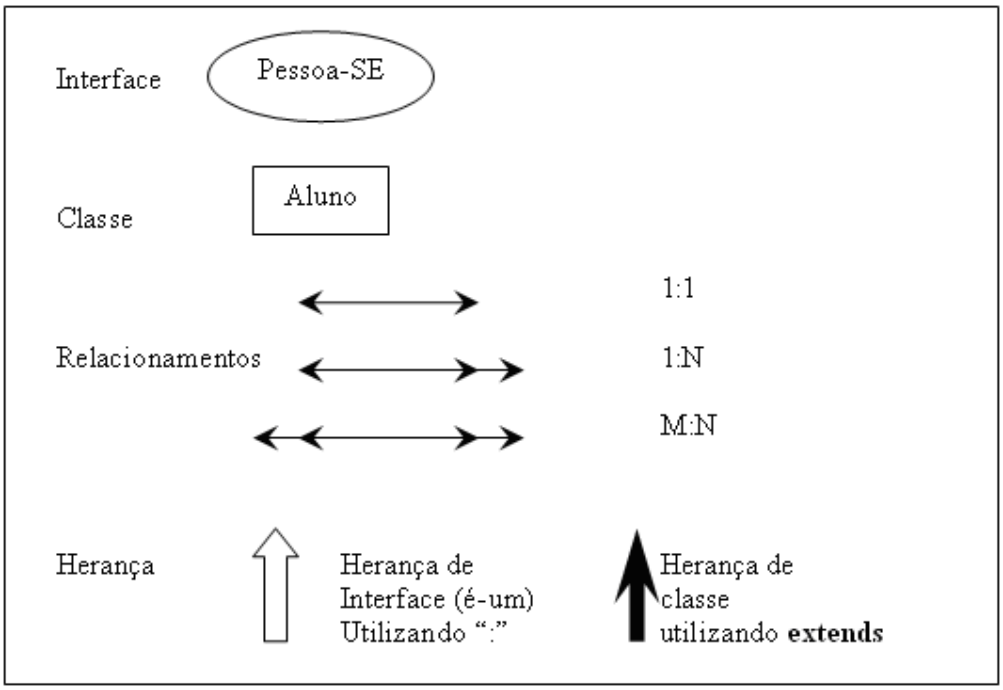


Figura 5. Notação gráfica de representação de esquemas em ODL.
Fonte: ELMASRI, R; NAVATHE, S. (2002)

O protótipo a seguir refere-se a definição de uma classe, sendo ainda detalhada no Quadro 2:

```
Class <nome> [extends <nome_pai>] [: <nomes_interfaces>]
[(extent <nome_extensao>)] { <elementos: atributos,
relacionamentos e métodos> [<declaracao_chaves>]}
```

| ***Declaração de atributos*** |
|---|
| `attribute <tipo> <nome>;` |
| ***Declaração de relacionamentos*** |
| `relationship <rangetype> <nome>`<br>`  [inverse <nome_classe_inv>::<metodo_inv>];` |
| ***Declaração de tipo de coleção*** |
| `Set<tipo_conj>` |
| ***Declaração de métodos*** |
| `<tipo_retorno> <nome_metodo>(<lista_parametros>)`<br>`  [raises(<lista_excecoes))]` |
| ***Declaração de parâmetros dos métodos*** |
| `[in \| out \| inout] <nome_parametro>` |
| ***Declaração de chave*** |
| `key(<lista_de_atributos) ou (key <lista_de_chaves>)` |

Quadro 2. Protótipo de definição de classe
Fonte: JONES, L; ALBERTO, L; LIMA, M. (2009)

## 2.3 VANTAGENS E DESVANTAGENS EM RELAÇÃO AO MODELO RELACIONAL

Uma vantagem que os BDOO têm sobre os BDR é a forma de representar os dados como objetos, ao invés de tabelas relacionadas separadamente, o que torna o entendimento muito mais fácil. Outro ponto importante a se destacar está relacionado com o modelo conceitual de BDOO que utiliza o mesmo modelo que a análise, o projeto e a programação baseada em objetos (BOSCARIOLI et al, 2006; MARTIN; ODELL, 1995).

Na Figura 6 é possível observar o processo de desenvolvimento dos modelos conceituais de BDR e de BDOO.

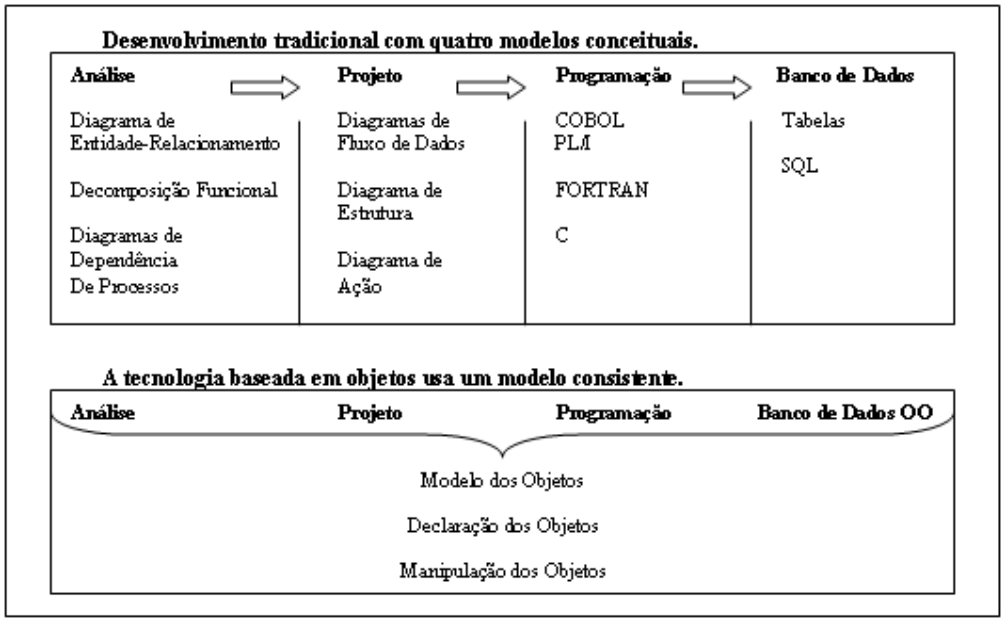


Figura 6. Modelos conceituais de BDR e de BDOO.
Fonte: MARTIN, J. (1994)

Como visto na Figura 6 o projeto de BDR é separado do projeto do programa, sendo o BD um espaço à parte do programa. Com OO deixa de existir a fronteira conceitual entre programação e BD, utilizando-se um modelo conceitual unificado, de forma a se obter (MARTIN, 1994):

a) produtividade, uma vez que a tradução dos paradigmas torna-se necessária;
b) menor quantidade de erros, já que erros de tradução entre os paradigmas deixam de existir;
c) melhor comunicação entre usuários, analistas e programadores;
d) qualidade;
e) flexibilidade;
f) maior capacidade criativa.

Os BDOO apresentam melhor desempenho que os relacionais quando de trata de aplicativos com muitas associações, devido ao uso de ponteiros para que objetos se refiram uns aos outros, o que em BDR é efetuado com junção (MARTIN; ODELL, 1995).

O armazenamento e acesso de grandes itens de dados em BDR não podem ser eficientemente efetuados de forma tabular. No entanto em BDOO, que utilizam diversas estruturas de armazenamento, os resultados têm mostrado maior eficácia no suporte a aplicações multimídia e CAD (MARTIN, 1994).

A fim de evitar redundâncias de métodos a tecnologia baseada em objetos utiliza herança e encapsulamento. Também utiliza classes que podem ser reaproveitadas em inúmeros aplicativos. Tais recursos diminuem custos com desenvolvimento e manutenção (MARTIN, 1994).

## 2.4 SISTEMAS GERENCIADORES DE BANCOS DE DADOS ORIENTADOS A OBJETOS

Alguns dos SGBDOO comerciais e protótipos experimentais desenvolvidos por fabricantes, laboratórios de pesquisa e universidades nos últimos anos podem ser observados na Tabela 1, que apresenta informações básicas dos principais SGBDOO.

Tabela 1. Informações básicas sobre os principais SGBDOO

| Nome | *Open Source* | Sistemas Operacionais | Fabricante | Site Oficial |
|---|---|---|---|---|
| EnterpriseDB | SIM | Linux, Mac OS, Solaris e Windows | EnterpriseDB | http://www.enterprisedb.com/ |
| Objectivity/DB | NÃO | Linux, LynxOS, Solaris, Windows, Unix | Objectivity Database Systems | http://www.objectivity.com |
| GemStone | NÃO | AIX, Linux, Solaris, Unix e Windows | GemStone System Inc. | http://www.gemstone.com |
| Versant | NÃO | Windows, Solaris, Linux e HP-UX | Versant Corp. | http://www.versant.com |
| Caché | NÃO | Windows, Open VMS, Linux e UNIX | Intersystems Software | http://www.intersystems.com.br |
| EyeDB | SIM | Linux, Solaris e Mac OS | Sysra Informatique | http://www.eyedb.org/ |
| ORION | SIM | Linux e UNIX | Orion Group (Purdue University) | http://orion.cs.purdue.edu/ |
| ObjectStore | NÃO | AIX, HP-UX, Linux, Solaris, Windows | Progress Software | http://www.objectstore.com |

Fonte: BOSCARIOLI, C. et al (2006)

A maioria dos SGBDOO relacionados na Tabela 1 não são *OpenSource*. Todos dão suporte à herança, polimorfismo e encapsulamento. As informações obtidas a partir da referência citada foram atualizadas em virtude de modificações que tais SGBDOO sofreram.

### 2.4.1 Sistema Gerenciador de Banco de Dados Caché

O SGBD Caché é um BD pós-relacional, que agrega as funcionalidades dos modelos relacional e OO, possuindo suporte a tabelas e objetos. Por meio de sua arquitetura unificada de dados (Figura 7) é construída uma camada de descrição para objetos e tabelas relacionais e estes são diretamente mapeados em uma estrutura multidimensional. Desta

forma, aplicações relacionais podem coexistir por meio da tecnologia de objetos (INTERSYSTEMS, 2002).

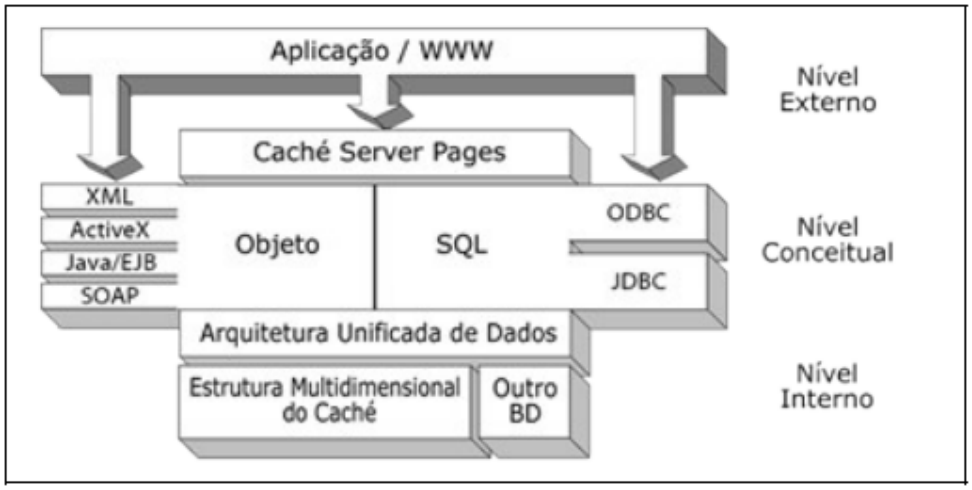


Figura 7. Arquitetura do Caché
Fonte: ROCHA, C. et al (2008)

O modelo de objeto do BD Caché é baseado no padrão ODMG, abordado no item 2.2. As definições de classes do Caché são realizadas utilizando a Linguagem de Definição de Classes ou *Class Definition Language* (CDL). Também é possível acessar os dados utilizando SQL e defini-los utilizando DDL, ambos provenientes do modelo relacional (INTERSYSTEMS, 2007).

2.4.1.1 *Class Definition Language*

A CDL permite definir, as classes, propriedades, métodos, parâmetros, relacionamentos entre outros. No Quadro 3 pode-se verificar alguns exemplos de tais definições (INTERSYSTEMS, 2009).

| | |
|---|---|
| Definição de Classe Persistente | Class NomePacote.NomeClasse Extends %Persistent<br>{<br><br>} |
| Definição de Classe Derivada | Class NomePacote.NomeSubClasse Extends NomePacote.NomeClasse<br>{<br><br>} |
| Definição de Propriedade | Property Propriedade As %String; |
| Definição de Método | Method NomeMetodo(Argumento As %String) As %Float<br>{<br>} |
| Definição de Relacionamento | Relationship NomeRelacionamento As NomePacote.NomeClasse [ Cardinality = one, Inverse = Propriedade ]; |

Quadro 3. Definições utilizando CDL
Fonte: Adaptado de INTERSYSTEMS (2009)

A CDL é baseada na ODL do padrão ODMG, porém possui algumas particularidades em relação a sua sintaxe de definição de classes. As principais características de sintaxe da linguagem CDL estão descritas no Anexo A, elaborado em 2000 por Danilo Kramel.

Os tipos de dados suportados pelo Caché são classes que representam cada uma um literal específico. O Caché permite que o usuário crie seus próprios tipos de dados. Na tabela 2 pode-se observar as classes de tipos disponíveis (BORBA, 2006).

Tabela 2. Classes de Tipos de Dados do Caché

| Class Name | Holds | Analogous SQL Type(s) |
|---|---|---|
| %Binary | binary data | VARBINARY |
| %Boolean | a boolean value | N/A |
| %Currency | a currency value | MONEY, SMALLMONEY |
| %Date | a date | DATE |

(continua)

Tabela 2. Classes de Tipos de Dados do Caché (conclusão)

| Class Name | Holds | Analogous SQL Type(s) |
|---|---|---|
| %Double | an IEEE floating point value | DOUBLE, DOUBLE PRECISION |
| %Float | a floating point value | FLOAT, REAL |
| %Integer | an integer | BIT, INT, INTEGER, SMALLINT, TINYINT |
| %List | data in $List format | N/A |
| %Name | a name in the form“Lastname,Firstname” | N/A |
| %Numeric | a numeric values of varying precision | DEC, DECIMAL, NUMBER, NUMERIC |
| %Status | an error status code | N/A |
| %String | a string | CHAR, CHAR VARYING, CHARACTER, CHARACTER VARYING, NATIONAL CHAR, NATIONAL CHAR VARYING, NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, NATIONAL VARCHAR, NCHAR, NVARCHAR, VARCHAR, VARCHAR2 |
| %Text | searchable text | N/A |
| %Time | a time value | TIME |
| %TimeStamp | a value for a time and date | TIMESTAMP |

Fonte: INTERSYSTEMS (2009)

Na Tabela 2 também é possível observar a descrição de cada classe e quais os tipos análogos em SQL. Tais classes se diferenciam das demais por não poderem ser instanciadas nem armazenadas independentemente, não conterem propriedades e por suportarem um conjunto específico de métodos (BORBA, 2006).

# 3 PROJETO DE BANCO DE DADOS RELACIONAL X ORIENTADO A OBJETOS

Os BDR apresentam algumas diferenças importantes em relação aos BDOO. A tecnologia de BDR visa à independência de dados, de forma que as estruturas de dados sejam independentes dos processos que usam os dados. Em BDOO o encapsulamento é sua principal meta e suas classes visam a reusabilidade (MARTIN; ODELL, 1995).

A estrutura de dados de uma classe muitas vezes pode-se tornar complexa, devido ao fato dos objetos serem compostos de objetos, que também são compostos de outros objetos e assim por diante. Os dados de cada objeto devem ser agrupados em conjunto, para facilitar o acesso aos mesmos. Em BDR isso pode não acontecer (KORTH; SILBERSCHATZ, 1995; MARTIN; ODELL, 1995).

A redundância em BDR é tratada com a normalização dos dados, que também ajuda a evitar as anomalias causadas pela redundância dos dados, porém a normalização não evita a redundância de código da aplicação. BDOO utilizam a herança como uma forma de reduzir a redundância de métodos, criando também classes para serem reutilizadas nas aplicações, de forma a diminuir ainda o custo de desenvolvimento e manutenção (MARTIN; ODELL, 1995).

No Quadro 4, a seguir, elaborado por James Martin e James Odell em 1995, pode-se observar um resumo das diferenças entre BDR e BDOO.

| **Bancos de Dados Relacionais** | **Bancos de Dados Orientados a Objeto** |
|---|---|
| Meta principal: independência de dados | Meta principal: encapsulamento |
| *Somente dados* - O banco de dados geralmente armazena somente dados. | *Dados mais métodos* - O banco de dados armazena dados mais métodos. |
| *Compartilhamento de dados* - Os dados podem ser compartilhados por quaisquer processos. Os dados são projetados para qualquer tipo de uso. | *Encapsulamento* - Os dados podem ser usados somente pelos métodos das classes. Os dados são projetados para ser usados somente por métodos específicos. |
| *Dados passivos* - Os dados são passivos. Certas operações limitadas podem ser automaticamente acionadas quando os dados são usados. | *Objetos ativos* - Os objetos são ativos. As solicitações fazem com que os objetos executem seus métodos. Alguns métodos podem ser |

| | |
|---|---|
| | altamente complexos, como por exemplo, aqueles que usam regras e uma máquina de inferência. |
| *Mudança constante* - Os processos que usam dados constantemente mudam. | *Classes projetadas para reuso* - As classes projetadas para alta reusabilidade raramente mudam. |
| *Independência de dados* - Os dados podem ser fisicamente reorganizados sem afetar a forma como são usados. | *Independência de classe* - As classes podem ser reorganizadas sem afetar a maneira como são usadas. |
| *Simplicidade* - Os usuários percebem os dados como colunas, linhas e tabelas. | *Complexidade* - As estruturas de dados podem ser complexas. Os usuários não têm consciência da complexidade por causa do encapsulamento. |
| *Tabelas separadas* - Cada relação (tabela) é separada. Comandos UNIR (*JOIN*) relacionam dados em tabelas separadas. | *Dados interligados* - Os dados podem estar interligados de forma que os métodos da classe consigam bom desempenho. As tabelas são uma das muitas estruturas que podem ser usadas. Grandes objetos binários são usados para som, imagens e vídeo. |
| *Dados não-redundantes* - A normalização de dados é feita para ajudar a eliminar a redundância de dados. (Ela nada faz para ajudar a eliminar a redundância no desenvolvimento de aplicações.) | *Métodos não-redundantes* - Dados e métodos não-redundantes são conseguidos com encapsulamento e herança. A herança ajuda a diminuir a redundância de métodos, e o reuso de classes ajuda a diminuir a redundância no desenvolvimento. |
| *SQL* - A linguagem SQL é usada para a manipulação de tabelas. | *Solicitações OO* - As solicitações provocam a execução de métodos. Métodos diversos podem ser usados. |
| *Desempenho* - O desempenho é uma preocupação com estruturas de dados altamente complexas. | *Otimização de classe* - Os dados para um objeto podem ser interligados e armazenados juntos de forma que possam ser acessados a partir de uma posição de mecanismos de acesso. Os BDOO oferecem um desempenho muito mais elevado do que os BDR para certas aplicações com dados complexos. |
| *Modelo conceitual diferente* - O modelo da estrutura de dados e acesso representados por tabelas *Join* é diferente daquele para análise, projeto e programação. O projeto deve ser convertido em tabelas relacionais, e o acesso, em estilo SQL. | *Modelo conceitual coerente* - Os modelos usados para análise, projeto, programação, e o acesso e a estrutura do banco de dados são semelhantes. Os conceitos de aplicação são diretamente representados por classes no BDOO. Quanto mais complexa a aplicação e suas estruturas de dados, mais economia em tempo e dinheiro no desenvolvimento da aplicação. |

Quadro 4. Resumo das diferenças entre BDR e BDOO
Fonte: Adaptado de MARTIN, J; ODELL, J. (1995)

## 3.1 ETAPAS DO PROJETO DE BANCO DE DADOS

O projeto de BD, atualmente, é feito por meio de uma modelagem de dados, que tem por objetivo representar de forma clara, resumida e ao mesmo tempo objetiva os dados de uma aplicação. Hoje estes projetos podem ser elaborados por quaisquer profissionais que

tenham conhecimento em técnicas estruturadas como a Modelagem Conceitual de Dados (MACHADO; ABREU, 2002).

A complexidade de elaboração de um projeto de sistema de informações requer planejamento, para se obter maior produtividade e confiabilidade dos sistemas desenvolvidos, evitando assim falhas de sistema (MACHADO; ABREU, 2002).

O tempo investido em um bom projeto de BD pode proporcionar inúmeras vantagens, pois se torna fácil alterar e manter a estrutura, modificar os dados e trabalhá-los, além de não desperdiçar tempo com erros do BD quando se pode aproveitar para projetar aplicações úteis ao negócio (HERNANDEZ; PARENTI; TOTELLO, 2000).

A metodologia correta para desenvolvimento de um projeto de BD é dividida em três etapas, o projeto conceitual, projeto lógico e projeto físico, de forma a tornar o desenvolvimento do projeto uma seqüência, conforme Figura 8 (MACHADO; ABREU, 2002).

O projeto físico descreve efetivamente como os dados serão armazenados no BD, detalhando-se complexas estruturas de dados de baixo nível (KORTH; SILBERSCHATZ, 1995).

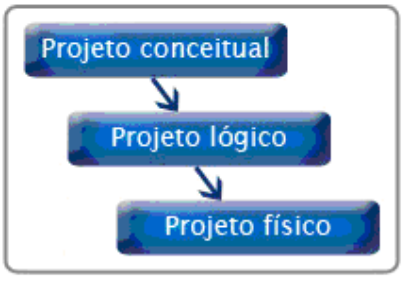


Figura 8. Etapas do projeto de banco de dados

### 3.1.1 Projeto Conceitual

O projeto conceitual é um modelo abstrato de um BD, que descreve a estrutura do mesmo, tendo por objetivo apresentar que dados podem conter no BD, sem informar como estes serão armazenados (HEUSER, 2001).

O modelo mais amplamente utilizado para o projeto conceitual de BDR é modelo Entidade Relacionamento (E-R) (Figura 9), que permite documentar as entidades de um BD junto com seus atributos (RAMAKRISHNAN; GEHRKE, 2008). O modelo E-R é inadequado para o projeto de BDOO, pois ele não possui nenhuma maneira de representar classes, mesmo havendo técnicas para se utilizar diagramas E-R OO. O modelo mais utilizado é o *Unified Modeling Language* (UML) (Figura 10) (HARRINGTON, 2002).

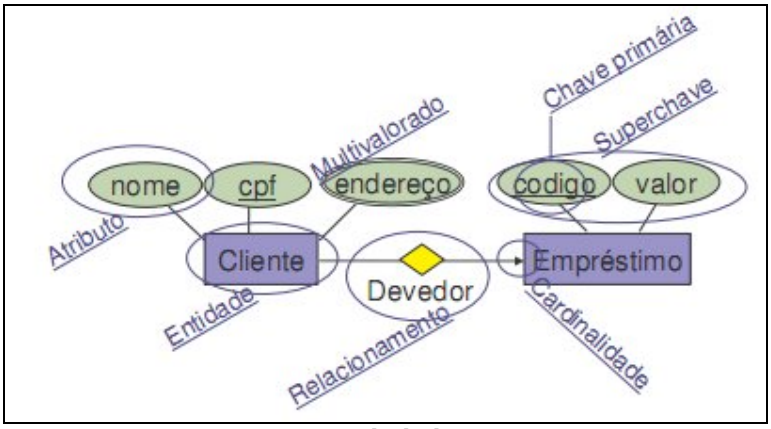


Figura 9. Exemplo de diagrama E-R

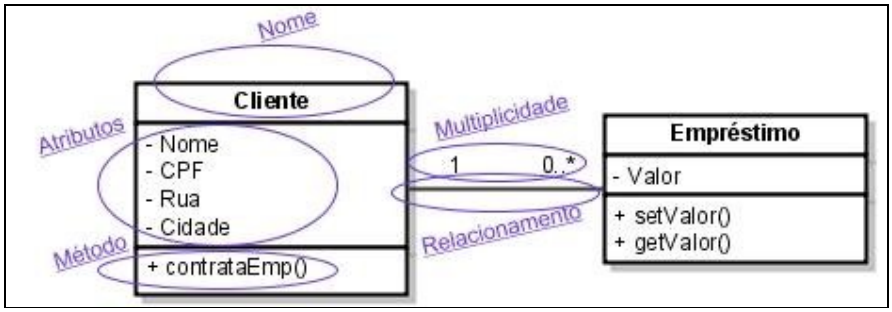


Figura 10. Exemplo de diagrama de classe UML

As Figuras 9 e 10 representam, respectivamente, um projeto lógico de BDR utilizando o diagrama E-R e de BDOO utilizando diagrama de classes UML. Quando se trata de projeto conceitual de BDOO além de se definir a estrutura dos objetos, também se definem os métodos a serem utilizados na manipulação destes objetos (ELMASRI; NAVATHE, 2002).

### 3.1.2 Projeto Lógico

O projeto lógico é uma descrição da estrutura do BD conforme o SGBD específico para o qual está sendo projetado, sendo assim, este depende totalmente do tipo de SGBD a ser utilizado (HEUSER, 2001).

Os BDR possuem chaves para identificar cada tabela e também para realizar o relacionamento entre elas, sendo estas chaves: primária, secundária, candidata e estrangeira (KORTH; SILBERSCHATZ, 1995; MACHADO; ABREU, 2002).

O modelo relacional requer regras de integridade, que visam garantir que as tabelas guardem informações compatíveis, uma das regras é a integridade de identidade, onde uma chave primária deve sempre possuir algum valor, não podendo conter um valor nulo, outra regra é a integridade referencial, que não permite que o valor de uma chave estrangeira seja diferente do seu valor como chave primária e por fim a de domínio, que especifica um conjunto de valores válidos para um dado. BDOO normalmente utilizam os tipos de dados padrões de linguagem de programação (KORTH; SILBERSCHATZ, 1995; MACHADO; ABREU, 2002).

# 4 TRABALHOS CORRELATOS

Neste capítulo serão apresentados trabalhos relacionados à transformação de esquemas de BD, com metodologias diferentes e aplicadas para diferentes SGBD. O primeiro realiza a transformação de esquema relacional para esquema OO e o segundo apresenta uma metodologia para implantação de modelos multidimensionais mapeados em BDOO.

## 4.1 TRANSFORMAÇÃO DE ESQUEMA RELACIONAL PARA ESQUEMA ORIENTADO A OBJETO EM SISTEMAS DE BANCOS DE DADOS HETEROGÊNEOS

Aqueo Kamada elaborou em 1996 uma Dissertação de Mestrado na Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica que apresenta um algoritmo de transformação de esquemas relacionais Sybase para esquemas OO O2, de forma semi-automática, interagindo com o usuário quando o sistema necessita de informações adicionais.

> Os esquemas transformados são típicos esquemas orientados a objetos que podem conter hierarquia de classes, hierarquia de composição, hierarquia de tuplas e conjuntos de atributos e o número de classes no esquema orientado a objetos transformado é relativamente pequeno, comparado com o número de relações no equivalente esquema relacional. (KAMADA, 1996, p. 5).

O algoritmo desenvolvido por Aqueo Kamada constitui-se de sete regras de transformação, com exemplos apresentados após cada regra.

A transformação é realizada utilizando dois programas distintos, primeiramente o programa LEX, proveniente do ambiente UNIX, utiliza como entrada um arquivo com as informações do SGBD de origem para extrair apenas as informações necessárias e então as armazena em um arquivo intermediário (KAMADA, 1996).

O segundo programa, tendo como entrada o arquivo intermediário gerado pelo primeiro programa, reconstitui o esquema relacional em um formato próximo da definição SQL original do BDR, após isto ele transforma o esquema relacional obtido em um esquema OO, utilizando as regras de transformação desenvolvidas por Aqueo Kamada.

## 4.2 METODOLOGIA PARA IMPLANTAÇÃO DE MODELOS MULTIDIMENSIONAIS EM BANCO DE DADOS ORIENTADO A OBJETOS

A Tese de Doutorado desenvolvida por Sueli de Fátima Poppi Borba em 2006 na Universidade Federal de Santa Catarina no Programa de Pós-Graduação em Engenharia de Produção propõe uma metodologia para implantação de modelos multidimensionais mapeados em BDOO representados pela UML. As etapas utilizadas são representadas na Figura 11.

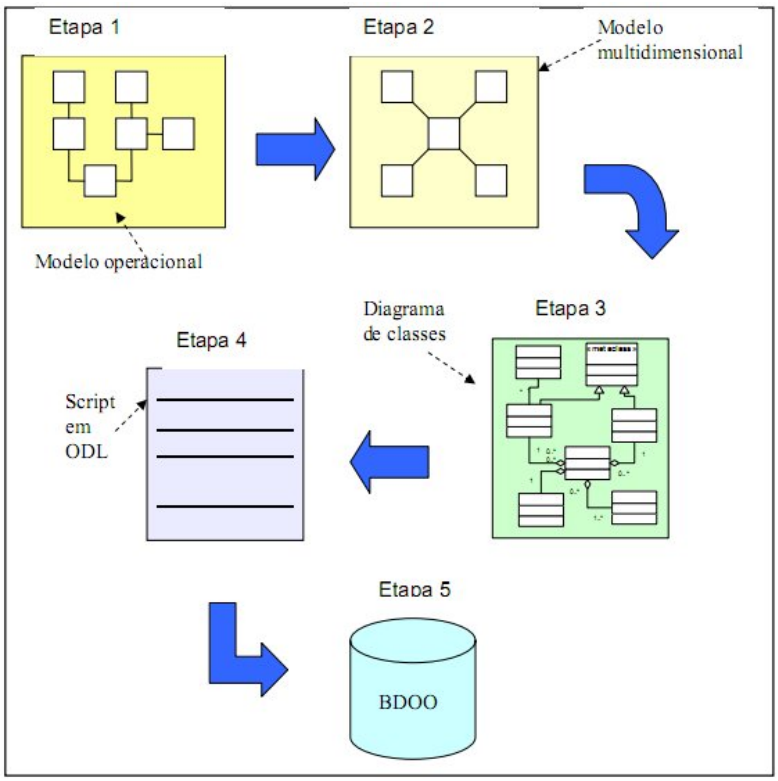


Figura 11. Resumo da metodologia utilizada
Fonte: BORBA, S. (2006)

A Etapa 1 tem por objetivo definir o negócio a ser modelado, observar estruturas genéricas e verificar os esquemas E-R existentes. Na Etapa 2 os objetivos são: definir o esquema conceitual preliminar e gerar o esquema Estrela ou Floco de neve.

O objetivo da Etapa 3 é gerar o diagrama de classes do esquema conceitual. Já o da Etapa 4 é mapear o modelo conceitual multidimensional para o paradigma da OO e finalmente o da Etapa 5 é implementar o modelo proposto em um BDOO.

# 5 TRANSFORMAÇÃO DE ESQUEMA RELACIONAL PARA ORIENTADO A OBJETOS

A transformação de esquemas relacionais de BD em esquemas OO compreende a elaboração de um método de transformação a partir dos metadados extraídos pela ferramenta desenvolvida por Samuel Brognoli em 2008 e também a incorporação deste método a esta ferramenta.

## 5.1 FERRAMENTA DE APOIO A ENGENHARIA REVERSA DE BANCOS DE DADOS RELACIONAIS

A necessidade de se obter o modelo conceitual de BDR motivou o desenvolvimento de uma ferramenta de engenharia reversa que extrai os metadados de BDR Oracle e Firebird para gerar o diagrama E-R. Para situações em que há mais de uma resposta possível, se faz necessária a interação com o usuário (BROGNOLI, 2008).

### 5.1.1 Características e Funcionamento

O processo de engenharia reversa utilizado na ferramenta desenvolvida por Samuel Brognoli em 2008 define o processo de engenharia reversa de modelos relacionais em quatro etapas:

a) identificação da construção do modelo E-R correspondente a cada tabela;

b) definição dos relacionamentos 1:N e 1:1;

c) definição de atributos;

d) definição de identificadores de entidades e relacionamentos.

Na realização do processo de engenharia reversa é necessário efetuar uma comunicação com a base de dados e extrair os metadados que estão armazenados no dicionário de dados (BROGNOLI, 2008).

A ferramenta foi desenvolvida utilizando a linguagem de programação Java por meio do ambiente de desenvolvimento Netbeans 6.0.1. O modelo E-R foi gerado utilizando uma biblioteca de recursos gráficos chamada *draw2d* do Eclipse, um ambiente integrado de desenvolvimento Java. Para o desenvolvimento da interface do programa foi utilizado o conjunto de componentes SWT. Já a conexão da aplicação com os SGBD Oracle e Firebird é efetuada por meio da *Java Database Connectivity* (JDBC) (BROGNOLI, 2008).

As etapas do sistema para realizar a engenharia reversa podem ser identificadas na Figura 12.

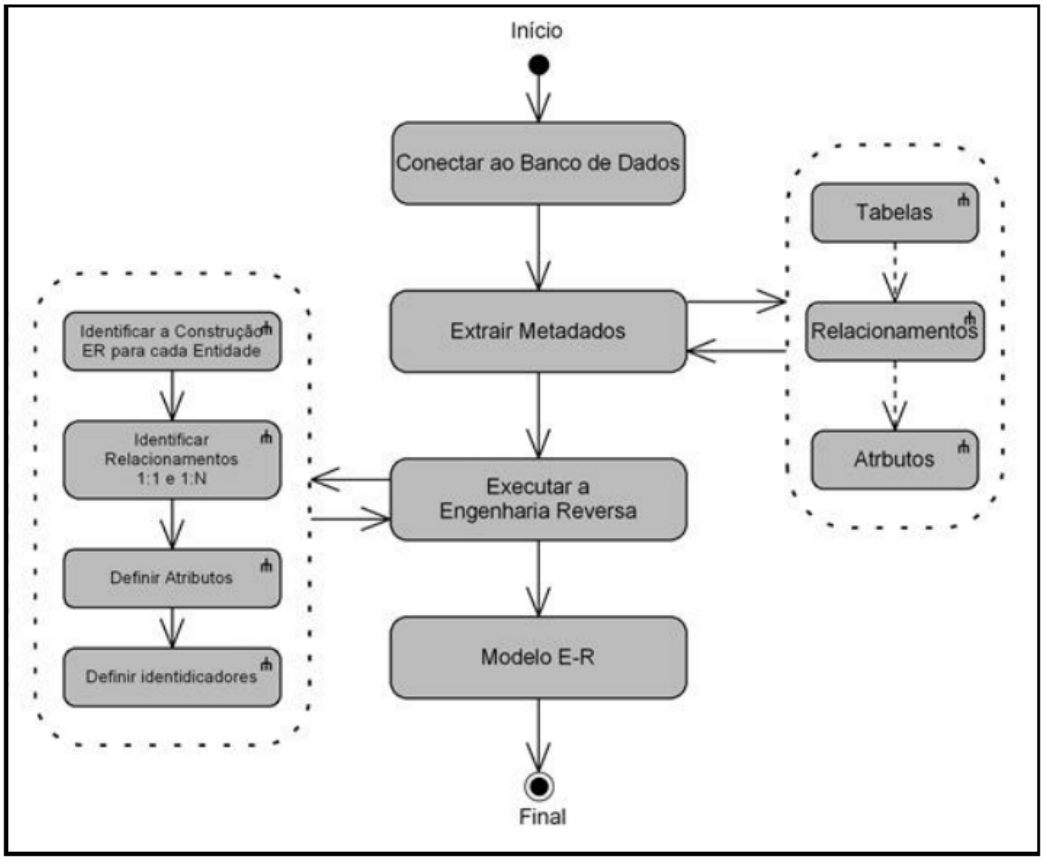


Figura 12. Diagrama de Atividade para a Engenharia Reversa
Fonte: BROGNOLI, S. (2008)

As principais etapas do processo destacados por Samuel Brognoli são a extração dos metadados e a execução da engenharia reversa. A extração dos metadados é realizada em três etapas, sendo elas: a extração das tabelas, a extração dos relacionamentos e a extração dos atributos. Para cada uma delas é executada uma consulta ao dicionário de dados e os resultados são armazenados em memória. A execução da engenharia reversa já foi citada anteriormente.

Para o armazenamento e manipulação das informações foram criadas classes internas, sendo as principais: *Entidade*, *Atributos* e *Relacionamentos*, que podem ser visualizadas na Figura 13.

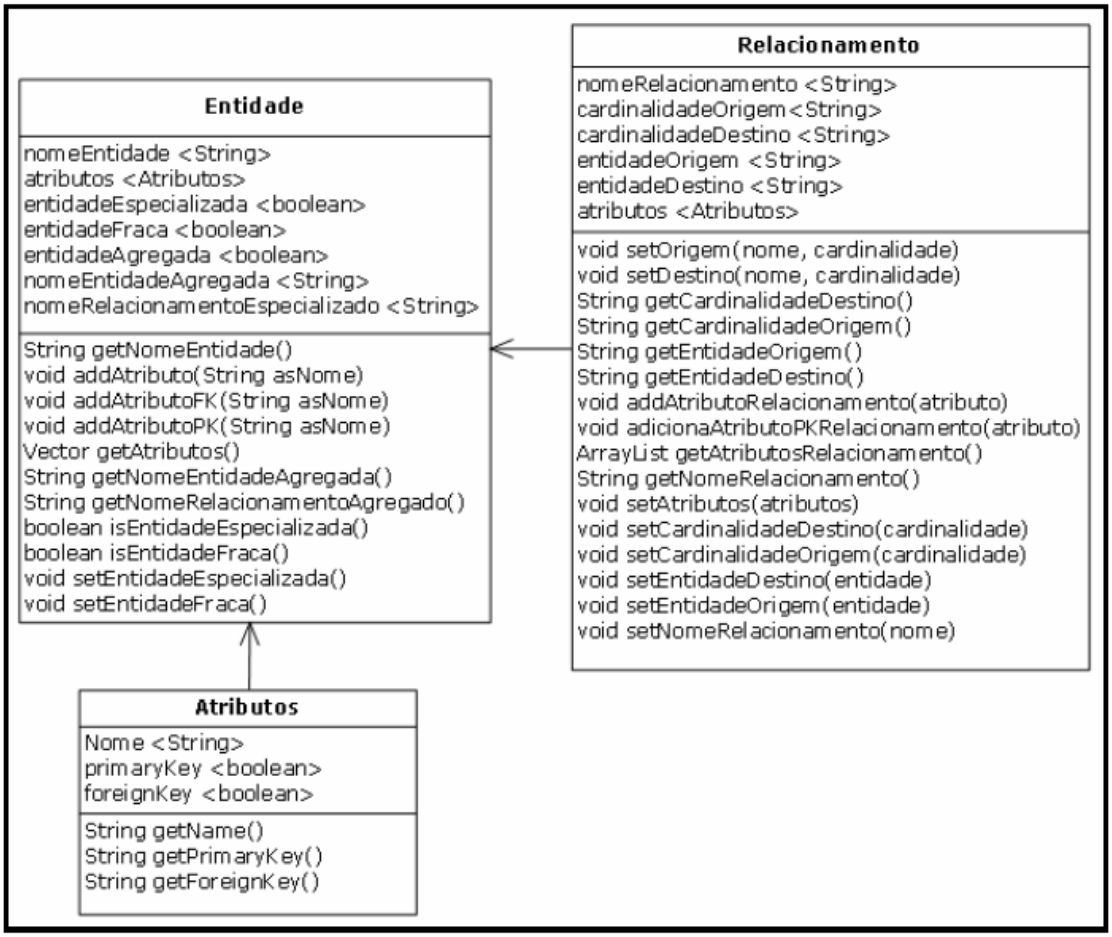


Figura 13. Diagrama de Classes – Estrutura de Armazenamento dos Metadados
Fonte: BROGNOLI, S. (2008)

A classe *Entidade* armazena os dados das tabelas, já a classe *Relacionamento* os dados das chaves estrangeiras, nome das tabelas de origem e destino e a cardinalidade do relacionamento. A classe *Atributos* por sua vez contém as informações dos atributos das tabelas e dos atributos dos relacionamentos (BROGNOLI, 2008).

Como resultado do processo anteriormente descrito, realizado pela ferramenta, é apresentado na Figura 14 um exemplo de parte do modelo E-R gerado.

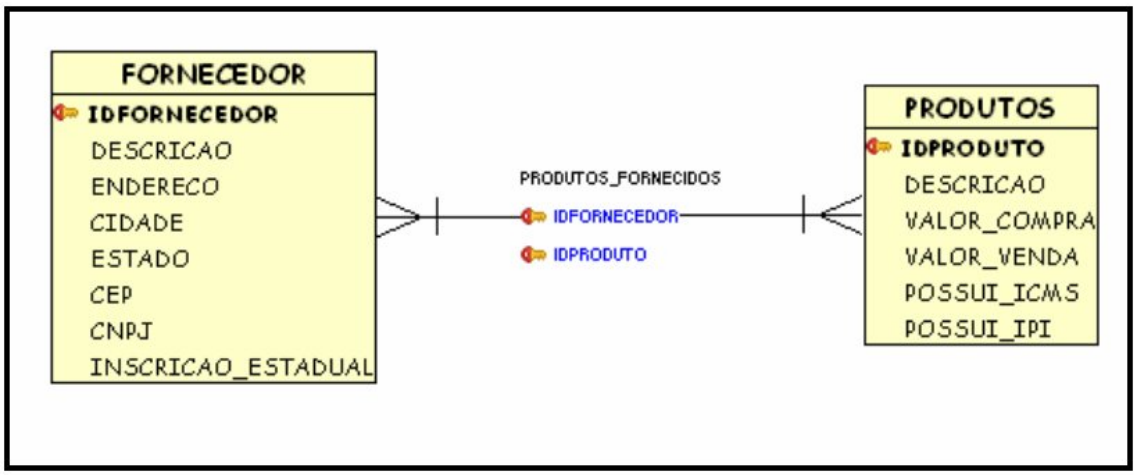


Figura 14. Parte de um diagrama gerado pela ferramenta
Fonte: BROGNOLI, S. (2008)

As informações obtidas pela ferramenta de Samuel Brognoli, depois de manipuladas utilizando as regras definidas por Carlos Heuser em 2001, são as seguintes:

a) entidades (fortes, fracas e especializadas);

b) relacionamentos N:N e relacionamentos 1:1 e 1:N com interação com usuário;

c) atributos das entidades e relacionamentos;

d) identificadores de entidades e relacionamentos.

A ferramenta citada não necessita dos metadados referentes aos tipos dos atributos, pois eles não estão presentes no modelo conceitual de BD. Contudo, para a geração do modelo OO se faz necessária a extração dos mesmos para os BD Oracle e Firebird.

### 5.1.2 Metadados - Tipos de Dados

A extração dos tipos de dados necessita da inclusão de sub-consultas SQL na consulta - já existente na ferramenta - que retorna as informações referentes aos atributos. A sintaxe das consultas SQL utilizadas para cada SGBD pode ser observada na Tabela 3.

Tabela 3. Consultas SQL para extrair os tipos de dados dos atributos

| SGBD | Consulta SQL |
|---|---|
| FIREBIRD | SELECT RDB$FIELD_NAME,<br>...<br>(SELECT<br>T2.RDB$TYPE_NAME<br>FROM<br>RDB$FIELDS T JOIN RDB$TYPES T2<br>ON<br>T.RDB$FIELD_TYPE = T2.RDB$TYPE )<br>WHERE<br>(F.RDB$RELATION_NAME = 'NOME_DA_TABELA' )AND<br>(T.RDB$FIELD_NAME = F.RDB$FIELD_SOURCE)AND<br>(T2.RDB$TYPE = T.RDB$FIELD_TYPE)AND<br>(T2.RDB$FIELD_NAME = 'RDB$FIELD_TYPE') ) AS TIPO,<br>(SELECT<br>T.RDB$FIELD_PRECISION<br>FROM<br>RDB$FIELDS T JOIN RDB$TYPES T2<br>ON<br>T.RDB$FIELD_TYPE = T2.RDB$TYPE )<br>WHERE<br>(F.RDB$RELATION_NAME = 'NOME_DA_TABELA' )AND<br>(T.RDB$FIELD_NAME = F.RDB$FIELD_SOURCE)AND<br>(T2.RDB$TYPE = T.RDB$FIELD_TYPE)AND<br>(T2.RDB$FIELD_NAME = 'RDB$FIELD_TYPE') ) AS PRECISAO,<br>(SELECT<br>T.RDB$FIELD_SCALE<br>FROM<br>RDB$FIELDS T JOIN RDB$TYPES T2<br>ON<br>T.RDB$FIELD_TYPE = T2.RDB$TYPE )<br>WHERE<br>(F.RDB$RELATION_NAME = 'NOME_DA_TABELA' )AND<br>(T.RDB$FIELD_NAME = F.RDB$FIELD_SOURCE)AND<br>(T2.RDB$TYPE = T.RDB$FIELD_TYPE)AND<br>(T2.RDB$FIELD_NAME = 'RDB$FIELD_TYPE') ) AS ESCALA,<br>...<br>FROM<br>RDB$RELATION_FIELDS F<br>WHERE<br>F.RDB$RELATION_NAME = 'NOME_DA_TABELA' |

(continua)

Tabela 3. Consultas SQL para extrair os tipos de dados dos atributos (conclusão)

| SGBD | Consulta SQL |
|---|---|
| ORACLE | SELECT COLUMN_NAME,<br>...<br>DATA_TYPE AS TIPO<br>DATA_PRECISION AS PRECISAO,<br>DATA_SCALE AS ESCALA<br>...<br>FROM<br>USER_TAB_COLUMNS T<br>WHERE<br>TABLE_NAME = ' NOME_DA_TABELA' |

De posse de todas as consultas necessárias, os metadados são extraídos, armazenados nas classes correspondentes e manipulados de forma a gerar o diagrama E-R correspondente. Após este processo, é dado início às etapas de transformação do esquema relacional para o esquema orientado a objetos.

## 5.2 ETAPAS DO PROCESSO DE TRANSFORMAÇÃO

A realização do processo de transformação foi dividida em seis etapas, sendo a primeira a identificação de falsos relacionamentos, que correspondem a relacionamentos de herança. A segunda etapa tem por objetivo tratar os relacionamentos N:N, para que estes se tornem classes de ligação entre as classes que se relacionam.

Na terceira etapa são escritas as classes correspondestes às entidades especializadas, seguidas pelas entidades fortes e fracas na quarta etapa.

### 5.2.1 Etapa 1 – Identificar Falsos Relacionamentos

A Etapa 1 consiste em identificar relacionamentos denominados falsos por serem ligações entre subclasses com suas superclasses. Desta forma, estes não podem ser tratados

como relacionamentos e serão posteriormente descritos nas classes especializadas utilizando-se herança.

```
/*************************************************************
 * Percorre o vetor que contém Relacionamentos para identificar
 * os relacionamentos falsos das Entidades Especializadas
 *************************************************************/
int i = 0;
Vector falsosRelacionamentos = new Vector();
while (ent.size() > i) {
    if (((Entidade) ent.elementAt(i)).isEntidadeEspecializada()) {
        int j = 0;
        Vector atributos = ((Entidade) ent.elementAt(i)).getAtributos();
        while (rel.size() > j) {
            if ((((Relacionamento) rel.elementAt(j)).getEntidadeOrigem()).equals(((Entida
             || (((Relacionamento) rel.elementAt(j)).getEntidadeDestino()).equals(((Entid
                int k = 0;
                while (atributos.size() > k){
                    if ((((Atributos) atributos.elementAt(k)).getPrimaryKey() == true &&
                    && ((((Atributos) atributos.elementAt(k)).getNomeEntidadeReferencia()
                    ||  (((Atributos) atributos.elementAt(k)).getNomeEntidadeReferencia()
                        falsosRelacionamentos.add((Relacionamento) rel.elementAt(j));
                    }
                    k++;
                }
            }
            j++;
        }
    }
    i++;
}
```

Figura 15. Parte do código para identificação dos relacionamentos falsos

Na Figura 15 é apresentada a parte do código onde é verificado se cada relacionamento indica uma entidade especializada. Em caso afirmativo este é adicionado a um vetor que é utilizado para tratar relacionamentos entre as classes.

### 5.2.2 Etapa 2 – Tratar Relacionamentos N:N

Esta etapa tem por objetivo identificar os relacionamentos N:N (muitos-para-muitos) e tratá-los, sendo que, para cada um, é criada uma classe correspondente, de intermédio entre as classes que se relacionam (Figura 16).

```
if ((((Relacionamento) vRel.elementAt(i)).cardinalidadeRelacionamento().equals("N:N")

    String nomeRel = ((Relacionamento) vRel.elementAt(i)).getNomeRelacionamento();
    Entidade entidade = new Entidade(nomeRel);
    entidade.addVetorAtributos(((Relacionamento) vRel.elementAt(i)).getAtributosRelac
    entidade.setEntidadeRelacionamento();

    String entOrigem  = ((Relacionamento) vRel.elementAt(i)).getEntidadeOrigem();
    String entDestino  = ((Relacionamento) vRel.elementAt(i)).getEntidadeDestino();
    String cardOrigem  = ((Relacionamento) vRel.elementAt(i)).getCardinalidadeOrigem(
    String cardDestino = ((Relacionamento) vRel.elementAt(i)).getCardinalidadeDestino

    ((Relacionamento) vRel.elementAt(i)).setEntidadeDestino(nomeRel);

    if(!entOrigem.equals(entDestino)){

        Relacionamento relacionamento = new Relacionamento(nomeRel);
        relacionamento.setEntidadeOrigem(entDestino);
        relacionamento.setEntidadeDestino(nomeRel);
        relacionamento.setCardinalidadeOrigem(cardOrigem);
        relacionamento.setCardinalidadeDestino(cardDestino);
        rel.addElement(relacionamento);
    }else
        ((Relacionamento) vRel.elementAt(i)).setAutoRelacionamento();
    ent.addElement(entidade);
}
```

Figura 16. Parte do código para tratar relacionamentos N:N

A Figura 16 demonstra, de forma resumida, o código utilizado para identificar relacionamentos N:N e tratá-los, de forma a ser criada uma nova classe idêntica ao relacionamento, com o mesmo nome e atributos.

Supondo-se, por exemplo, um relacionamento N:N entre as entidades A e B. Uma nova classe C será criada para representar este relacionamento, e as classes A e B se relacionarão por meio da nova classe (a classe C).

### 5.2.3 Etapa 3 – Escrever Entidades Especializadas

Uma entidade especializada torna-se uma subclasse, que necessita herdar estado e comportamento de sua superclasse, para tal utiliza-se a seguinte sintaxe: *class nomeSubClasse extends nomeSuperclasse*. Desta forma, é realizada a identificação da superclasse e escrita conforme a sintaxe da ODL e da CDL (Figura 17).

```
if ((((Relacionamento) relEntidade.elementAt(i)).getEntidadeOrigem()).equals(((Er
    superClasse = ((Relacionamento) relEntidade.elementAt(i)).getEntidadeDestino
    superClasse = validadorString(superClasse);
} else
{
    superClasse = ((Relacionamento) relEntidade.elementAt(i)).getEntidadeOrigem()
    superClasse = validadorString(superClasse);
}

String txtDoArquivoODMG = "class " + nomeEntidade + " extends " + superClasse + "
                          "{ \r\n" +
                          this.escreverAtributosODMG(atributos, tiposODMG) + "\r\
                          this.escreverRelacionamentosODMG(relEntidade, entidade,
                          "};\r\n" + "\r\n";

String txtDoArquivoCache = "Class bd." + nomeEntidade + " Extends bd." + superCla
                          "{ \r\n" + "\r\n"+
                          this.escreverAtributosCache(atributos, tiposCache) + "\
                          this.escreverRelacionamentosCache(relEntidade, entidade
                          "}\r\n" + "\r\n";

saidaODMG.write(txtDoArquivoODMG.getBytes());
saidaCache.write(txtDoArquivoCache.getBytes());
```

Figura 17. Parte do código de escrita de entidades especializadas

Conforme a entidade de origem e destino do relacionamento da entidade é possível verificar qual a sua superclasse e assim escrevê-la utilizando-se herança. A escrita conforme a CDL necessita da inclusão do nome do pacote da classe antes do nome da mesma, para isto foi adotado o nome *bd*.

### 5.2.4 Etapa 4 – Escrever Entidades Fortes e Fracas

As entidades fortes e fracas tornam-se classes e são escritas conforme a ODL utilizando-se a seguinte sintaxe: *classe nomeClasse*. Já para a CDL é utilizada: *Class nomePacote.nomeClasse Extends %Persistent*. Em ODL não é necessário especificar que a classe é persistente, pois o padrão já supõe que seja.

```
String txtDoArquivoODMG = "class " + nomeEntidade + "\r\n" +
                          "{ \r\n" +
                          this.escreverAtributosODMG(atributos, tiposODMG) + "\r\n
                          this.escreverRelacionamentosODMG(relEntidade, entidade,
                          "};\r\n" + "\r\n";

String txtDoArquivoCache = "Class bd." + nomeEntidade + " Extends %Persistent"
                          "{ \r\n" +
                          this.escreverAtributosCache(atributos, tiposCache) + "\r
                          this.escreverRelacionamentosCache(relEntidade, entidade,
                          "}\r\n" + "\r\n";

saidaODMG.write(txtDoArquivoODMG.getBytes());
saidaCache.write(txtDoArquivoCache.getBytes());
```

Figura 18. Parte do código de escrita de entidades especializadas

A Figura 18 demonstra parte do código utilizado para escrever as entidades fortes e fracas como classes, conforme as linguagens do padrão ODMG e do SGBD Caché. Ainda, em cada uma das classes são escritos seus respectivos atributos e relacionamentos.

### 5.2.5 Etapa 5 – Escrever Atributos

Os atributos de cada entidade necessitam de uma conversão para um tipo compatível no padrão ODMG e no SGBD Caché. Nem todos os tipos de dados contidos nos SGBD Firebird e Oracle (Tabela 4 e Tabela 5) podem ser mapeados para um tipo compatível. Sendo assim, estes foram escritos utilizando *UNDEFINED*, para que possam ser identificados posteriormente e alterados para um tipo válido a critério do desenvolvedor.

Tabela 4. Tipos de Dados do SGBD Firebird

| Tipo de Dados | Descrição |
|---|---|
| Binary Large Objects (BLOB) | dados binários |
| Char (n) | caracteres com valor de 1 ao máximo 32767 |
| Date | datas |

(continua)

Tabela 4. Tipos de Dados do SGBD Firebird (conclusão)

| Tipo de Dados | Descrição |
|---|---|
| Decimal (precisão – escala) | números com precisão entre 1 a 18 dígitos e escala entre 1 a 18 casas decimais |
| Double Precision | valores de ponto flutuante com precisão de 15 dígitos, ocupando 64 bits |
| Float | valores de ponto flutuante com 7 dígitos de precisão, ocupando 32 bits |
| Integer | números inteiros de -2.147.283.648 até 2.147.483.647 |
| Numeric (precisão – escala) | números com precisão entre 1 a 18 dígitos e escala entre 1 a 18 casas decimais |
| Smallint | números inteiros de -32.768 até 32.767 |
| Time | valores de tempo de 00:00 AM até 23:59:9999 PM |
| Timestamp | data e hora em uma única coluna |
| Varchar (n) | caracteres com valor de 1 ao máximo 32767 |

Fonte: WILDEROM, B.; WILDEROM, S. (2002)

Tabela 5. Tipos de Dados do SGBD Oracle

| Tipo de Dados | Descrição |
|---|---|
| BFILE | localizados para arquivo binário externo de até 4 Gbytes |
| BLOB | objeto binário de até 4 Gbytes |
| BINARY_DOUBLE | tipo de dado de ponto flutuante de até 64 bits |
| BINARY_FLOAT | tipo de dado de ponto flutuante de até 32 bits |
| CHAR (size) | campo fixo de até 2.000 bytes |
| CLOB | objeto binário contendo caracteres do tipo *single byte* de até 4 Gbytes |

(continua)

Tabela 5. Tipos de Dados do SGBD Oracle (conclusão)

| Tipo de Dados | Descrição |
|---|---|
| DATE | data de 1,4712 a.C até 31,4712 |
| LONG | caractere de até 2 Gbytes |
| LONG RAW | variável do tipo *raw binary* de até 2 Gbytes |
| NCHAR (size) | dado do tipo caractere de até 2.000 bytes |
| NCLOB | objeto contendo caracteres do tipo *multibyte* de até 4 Gbytes |
| NVARCHAR2 (size) | dado do tipo caractere de até 4.000 bytes |
| NUMBER (p, s) | dado do tipo número com precisão de 1 a 38 e escala de -84 a +127 |
| RAW (size) | dado do tipo *raw binary* de até 2.000 bytes |
| ROWID | string hexadecimal que representa o endereço único de uma linha da tabela |
| MLSLABEL | formato binário de um rótulo (label) do sistema operacional |
| TIMESTAMP | valores que são precisos para segundos fracionários |
| VARCHAR2 (size) | dado do tipo caractere de até 4.000 bytes |

Fonte: RAMALHO, J. (2005)

As Tabelas 4 e 5 apresentam, respectivamente, os tipos de dados dos SGBD Firebird e Oracle, bem como a descrição de cada um desses. Já as Tabelas 6 e 7, apresentam, respectivamente, o mapeamento dos tipos de dados do Firebird e do Oracle para o padrão ODMG e o SGBD Caché.

Tabela 6. Tipos de dados análogos do Firebird para ODMG e Caché

| Firebird | ODMG | Caché |
|---|---|---|
| Binary Large Objects (BLOB) | UNDEFINED | UNDEFINED |
| Char (n) | String | String |
| Date | Date | Date |
| Decimal (precisão – escala) | Double | Numeric |
| Double Precision | Double | Double |
| Float | Float | Float |
| Integer | Long | Integer |
| Numeric (precisão – escala) | Double | Numeric |
| Smallint | Short | Integer |
| Time | Time | Time |
| Timestamp | Timestamp | Timestamp |
| Varchar (n) | String | String |

Tabela 7. Tipos de dados análogos do Oracle para ODMG e Caché

| Oracle | ODMG | Caché |
|---|---|---|
| BFILE | UNDEFINED | UNDEFINED |
| BLOB | UNDEFINED | UNDEFINED |
| BINARY_DOUBLE | Double | Double |
| BINARY_FLOAT | Float | Float |
| CHAR (size) | String | String |
| CLOB | UNDEFINED | UNDEFINED |

(continua)

Tabela 7. Tipos de dados análogos do Oracle para ODMG e Caché (conclusão)

| Oracle | ODMG | Caché |
|---|---|---|
| DATE | Date | Date |
| LONG | UNDEFINED | UNDEFINED |
| LONG RAW | UNDEFINED | UNDEFINED |
| NCHAR (size) | String | String |
| NCLOB | UNDEFINED | UNDEFINED |
| NVARCHAR2 (size) | String | String |
| NUMBER (p, s)<br>NUMBER (38<br>NUMBER | Double<br>Long<br>Double | Numeric<br>Integer<br>Double |
| RAW (size) | UNDEFINED | UNDEFINED |
| ROWID | UNDEFINED | UNDEFINED |
| MLSLABEL | UNDEFINED | UNDEFINED |
| TIMESTAMP | Timestamp | Timestamp |
| VARCHAR2 (size) | String | String |

Para se obter o tipo compatível do padrão ODMG e do SGBD Caché para o tipo NUMBER(p,s) do Oracle, e necessário realizar a verificação da precisão (p) e escala (s).

Os atributos, após a identificação de seus tipos, são escritos conforme ODL utilizando a sintaxe: *attribute tipoAtributo nomeAtributo;* para a CDL é utilizado: *Property nomeAtributo As %tipoAtributo;*.

### 5.2.6 Etapa 6 – Escrever Relacionamentos

Os relacionamentos então divididos em 1:1, 1:N e N:N. A escrita de tais relacionamentos necessita primeiramente de uma verificação para conhecer se os mesmos não fazem parte dos falsos relacionamentos. Caso não sejam, estes podem ser escritos como relacionamentos conforme a sintaxe dos modelos OO.

Relacionamentos 1:1 são escritos conforme a ODL seguindo a sintaxe:

```
relationship classeRelacionada referencia
inverse classeRelacionada::eReferenciadoPor;
```

Tal definição é inserida em ambas as classes que participam do relacionamento. Os identificadores *referencia* e *eReferenciadoPor* foram escolhidos como palavras-chave para nomear as participações no relacionamento.

A escrita de relacionamentos 1:1 utilizando CDL difere da ODL por não ter suporte direto utilizando relacionamento, de forma que este deve ser escrito como uma propriedade na classe do lado de origem. A sintaxe utilizada em CDL é a seguinte:

```
Property classeRelacionada As bd.classeRelacionada;
```

Um relacionamentos 1:N em ODL é escrito, na classe que referencia N objetos da outra classe, conforme a sintaxe:

```
relationship set<classeRelacionada> referencia
inverse classeRelacionada::eReferenciadoPor;
```

A utilização de *set<>* indica uma coleção que não permite elementos repetidos, esta poderia ser ainda de outro tipo coleção do padrão ODMG. Já na classe que é referenciada N vezes a sintaxe é a seguinte:

```
relationship classeRelacionada referencia
inverse classeRelacionada::eReferenciadoPor;
```

Utilizando a CDL o mesmo relacionamento é escrito, na classe que referencia N objetos da outra classe, seguindo a sintaxe:

```
Relationship  classeRelacionada As bd.classeRelacionada
[ Cardinality = many, Inverse = classe ];
```

Já na classe referenciada N vezes a sintaxe do relacionamento é escrita:

```
Relationship classeRelacionada As bd.classeRelacionada
[ Cardinality = one, Inverse =  classe ];
```

Os relacionamentos N:N em ODL são descritos utilizando uma coleção em ambas as classes envolvidas. Este tipo de relacionamento pode ser representado de duas formas: no caso do relacionamento não conter atributos próprios este pode ser representado diretamente entre as classes; caso hajam atributos no relacionamento - além das chaves estrangeiras das entidades – é necessária uma classe intermediária que também recebe as coleções das entidades relacionadas, conforme a sintaxe:

```
relationship set<classeRelacionada> referencia
inverse classeRelacionada::eReferenciadoPor;
```

Contudo, de forma a tornar a escrita dos relacionamentos N:N em ODL coerente e mais próxima dos mesmos relacionamentos em CDL, foi optado por representá-los da mesma forma em ambas as linguagens, ou seja, mesmo que o relacionamento não possua atributos próprios é criada uma classe intermediária para este.

Em CDL, o mesmo relacionamento necessita de uma classe intermediária referenciando as classes do relacionamento. As referências são escritas nesta utilizando:

```
Relationship classe1Relacionada As bd.classe1Relacionada
[ Cardinality = one, Inverse =  classeRelacionamento];

Relationship classe2Relacionada As bd.classe2Relacionada
[ Cardinality = one, Inverse =  classeRelacionamento];
```

As duas outras classes recebem, cada uma, uma referência para a classe intermediária, seguindo a sintaxe a seguir:

```
Relationship  classeRelacionamento As bd.classeRelacionamento
[ Cardinality = many, Inverse = classe1Relacionada ];
```

## 5.3 RECURSOS UTILIZADOS

A inclusão das etapas de transformação entre os modelos relacional e orientado a objetos na ferramenta de engenharia reversa se deu com a utilização da linguagem de programação Java, com a qual a mesma havia sido desenvolvida. Para tanto foi utilizado o ambiente de desenvolvimento NetBeans 6.7.1, disponível gratuitamente em: http://download.netbeans.org/netbeans/6.7.1/final/.

Utilizando-se o NetBeans, é necessária ainda a instalação de uma versão da *Java Development Kit* (JDK) que pode ser obtida - também gratuitamente - por meio do endereço: http://java.sun.com/. A versão utilizada foi a JDK 1.6.

A versão do Oracle utilizada na realização deste projeto foi a 10g Express Edition, a qual pode ser obtida de forma gratuita para download no site da Oracle em: http://www.oracle.com/technology/software/products/database/xe/index.html. O Firebird utilizado foi o 2.1 Server Manager que também pode ser obtido de forma gratuita no site: http://www.firebirdsql.org/index.php?op=files.

## 5.4 UTILIZAÇÃO DA FERRAMENTA

A utilização da ferramenta se dá primeiramente pela conexão com o banco de dados, logo após, a extração dos metadados, a execução da engenharia reversa, e por fim, a realização da transformação do modelo relacional para o orientado a objetos, gerando-se os scripts para o padrão ODMG e para o SGBD caché. Tal processo pode ser melhor compreendido observando-se o diagrama de atividades na Figura 19.

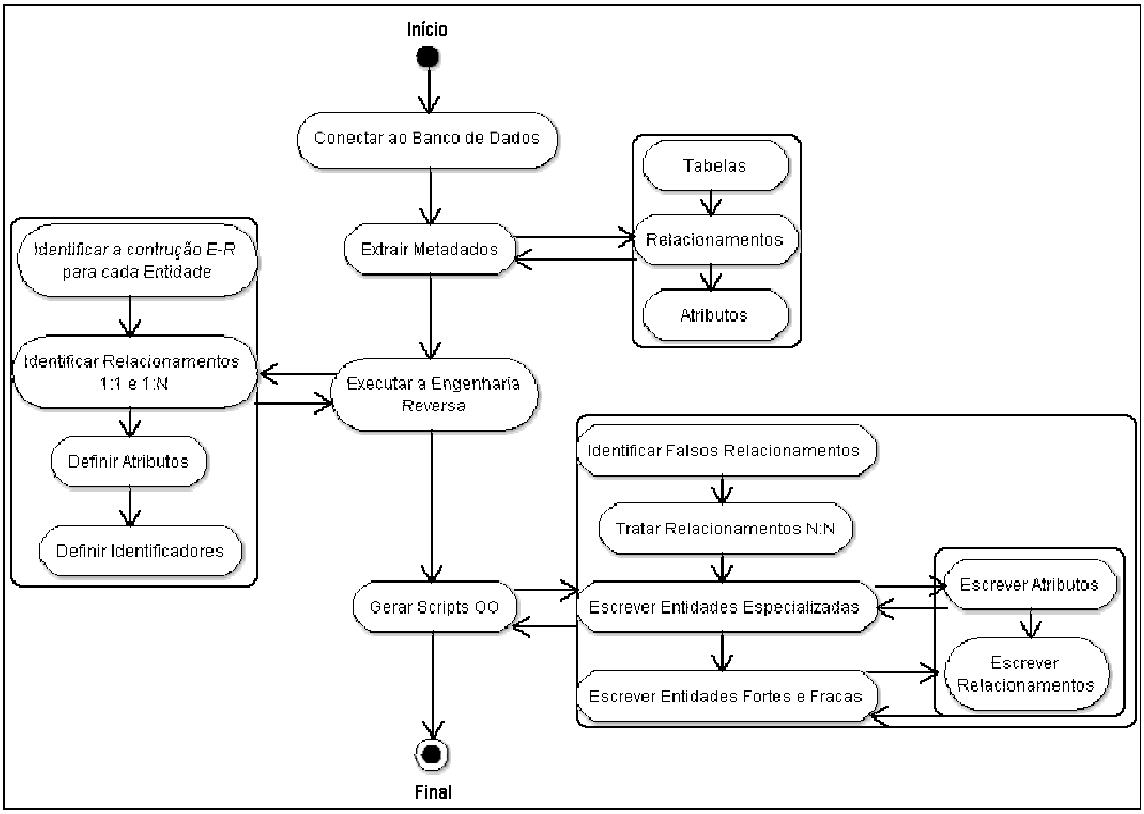

Figura 19. Diagrama de atividades da ferramenta

No menu *Processos* e na *barra de ferramenta*s (Figura 20) da aplicação foi incluída a opção Gerar Scripts OO (opção f). Permanecendo as demais funcionalidades inalteradas, opção (a): abrir um arquivo com extensão "*.der"; (b): salvar o modelo E-R em um arquivo com a extensão *.der; (c): imprimir o modelo E-R; (d): conectar o sistema ao BD e iniciar a extração do dicionário de dados; (e): iniciar o processo de engenharia reversa e gerar o modelo E-R; (g): exibir informações sobre o sistema, e por fim, a opção (h) finalizar o sistema.

Figura 20. Barra de ferramentas

Como exemplo de utilização da ferramenta na realização da transformação do modelo relacional para orientado a objetos foi utilizado um diagrama E-R (Figura 21) obtido a partir de um pequeno BD desenvolvido em Firebird.

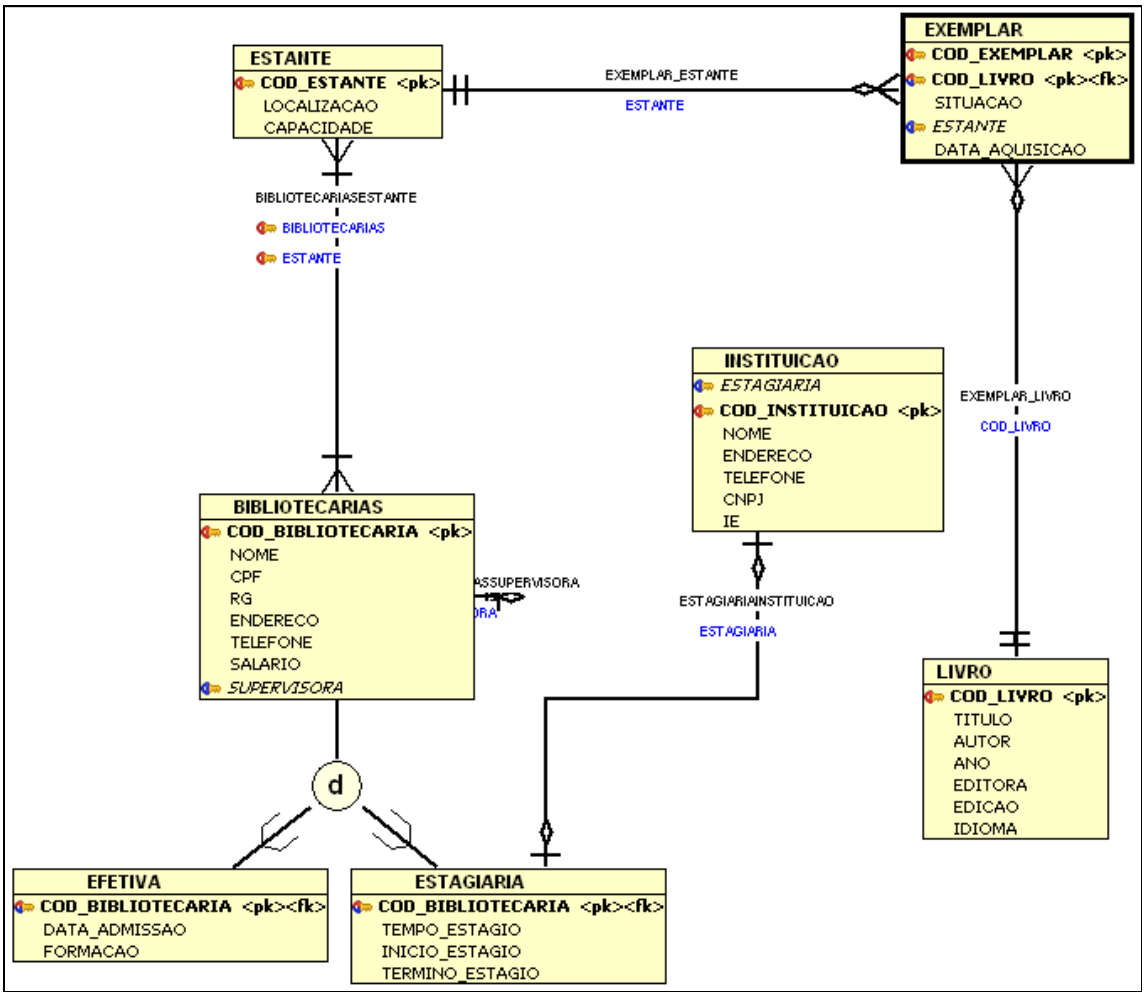

Figura 21. Diagrama E-R de uma Bliblioteca

A Figura 21 demonstra o diagrama E-R do BD de uma biblioteca. Neste pode-se observar um relacionamento 1:1 entre as entidades *Estagiaria* e *Instituição*, 1:N entre as entidades *Estante* e *Exemplar* e entre *Livro* e *Exemplar*, sendo que *Exemplar* representa uma entidade fraca.

Há também um auto relacionamento 1:N na classe *Bibliotecárias* e esta possui ainda um relacionamento N:N entre *Estante* e *Bibliotecárias*, sendo que a classe

*Bibliotecárias* é super classe de *Efetiva e Estagiária*, caracterizando herança de estado e comportamento.

A escrita das entidades fortes e fracas como, por exemplo, *Estante* e *Exemplar* respectivamente é apresentada na Tabela 8, juntamente com seus atributos.

Tabela 8. Geração de classes Estante e Exemplar

| ODMG | Caché |
|---|---|
| class ESTANTE<br>{<br>attribute long CODESTANTE;<br>attribute string LOCALIZACAO;<br>attribute long CAPACIDADE;<br>}; | Class bd.ESTANTE Extends %Persistent<br>{<br>Property CODESTANTE As %Integer;<br>Property LOCALIZACAO As %String;<br>Property CAPACIDADE As %Integer;<br>} |
| class EXEMPLAR<br>{<br>attribute long CODEXEMPLAR;<br>attribute string SITUACAO;<br>attribute date DATAAQUISICAO;<br>}; | Class bd.EXEMPLAR Extends %Persistent<br>{<br>Property CODEXEMPLAR As %Integer;<br>Property SITUACAO As %String;<br>Property DATAAQUISICAO As %Date;<br>} |

As entidades especializadas *Efetiva e Estagiaria* são escritas conforme a Tabela 9 com seus respectivos atributos, utilizando a herança por meio de *extends*.

Tabela 9. Geração de subclasses Efetiva e Estagiária

| ODMG | Caché |
|---|---|
| class EFETIVA extends BIBLIOTECARIAS<br>{<br>attribute date DATAADMISSAO;<br>attribute string FORMACAO;<br>}; | Class bd.EFETIVA Extends bd.BIBLIOTECARIAS<br>{<br>Property DATAADMISSAO As %Date;<br>Property FORMACAO As %String;<br>} |
| class ESTAGIARIA extends BIBLIOTECARIAS<br>{<br>attribute long TEMPOESTAGIO;<br>attribute date INICIOESTAGIO;<br>attribute date TERMINOESTAGIO;<br>}; | Class bd.ESTAGIARIA Extends bd.BIBLIOTECARIAS<br>{<br>Property TEMPOESTAGIO As %Integer;<br>Property INICIOESTAGIO As %Date;<br>Property TERMINOESTAGIO As %Date;<br>} |

Um relacionamento 1:N entre as entidades *Livro* e *Exemplar* é descrito em suas classes conforme apresenta a Tabela 10.

Tabela 10. Geração de relacionamento 1:N entre as classes Livro e Exemplar

| ODMG | Caché |
|---|---|
| class LIVRO<br>{<br>attribute long CODLIVRO;<br>attribute string TITULO;<br>attribute string AUTOR;<br>attribute date ANO;<br>attribute string EDITORA;<br>attribute long EDICAO;<br>attribute string IDIOMA;<br><br>relationship set<EXEMPLAR> referencia<br>inverse EXEMPLAR::eReferenciadoPor;<br>}; | Class bd.LIVRO Extends %Persistent<br>{<br>Property CODLIVRO As %Integer;<br>Property TITULO As %String;<br>Property AUTOR As %String;<br>Property ANO As %Date;<br>Property EDITORA As %String;<br>Property EDICAO As %Integer;<br>Property IDIOMA As %String;<br><br>Relationship EXEMPLAR As bd.EXEMPLAR<br>[ Cardinality = many, Inverse = LIVRO ];<br>} |
| class EXEMPLAR<br>{<br>attribute long CODEXEMPLAR;<br>attribute string SITUACAO;<br>attribute date DATAAQUISICAO;<br><br>relationship ESTANTE referencia<br>inverse ESTANTE::eReferenciadoPor;<br><br>relationship LIVRO referencia<br>inverse LIVRO::eReferenciadoPor;<br>}; | Class bd.EXEMPLAR Extends %Persistent<br>{<br>Property CODEXEMPLAR As %Integer;<br>Property SITUACAO As %String;<br>Property DATAAQUISICAO As %Date;<br><br>Relationship ESTANTE As bd.ESTANTE<br>[ Cardinality = one, Inverse = EXEMPLAR ];<br><br>Relationship LIVRO As bd.LIVRO<br>[ Cardinality = one, Inverse = EXEMPLAR ];<br>} |

Entre as entidades *Estagiaria* e *Instituição* cujo relacionamento é 1:1 é possível observar a escrita dos mesmos nas suas respectivas classes na Tabela 11.

Tabela 11. Geração de relacionamento 1:1 entre as classes Estagiaria e Instituição

| ODMG | Caché |
|---|---|
| class ESTAGIARIA extends BIBLIOTECARIAS<br>{<br>attribute long TEMPOESTAGIO;<br>attribute date INICIOESTAGIO;<br>attribute date TERMINOESTAGIO;<br><br>relationship INSTITUICAO referencia<br>inverse INSTITUICAO::eReferenciadoPor;<br>}; | Class bd.ESTAGIARIA Extends bd.BIBLIOTECARIAS<br>{<br>Property TEMPOESTAGIO As %Integer;<br>Property INICIOESTAGIO As %Date;<br>Property TERMINOESTAGIO As %Date;<br>} |

(continua)

Tabela 11. Geração de relacionamento 1:1 entre as classes Estagiaria e Instituição (conclusão)

| ODMG | Caché |
|---|---|
| class INSTITUICAO<br>{<br>attribute long CODINSTITUICAO;<br>attribute string NOME;<br>attribute string ENDERECO;<br>attribute string TELEFONE;<br>attribute string CNPJ;<br>attribute string IE;<br><br>relationship ESTAGIARIA referencia<br>inverse ESTAGIARIA::eReferenciadoPor;<br>}; | Class bd.INSTITUICAO Extends %Persistent<br>{<br>Property CODINSTITUICAO As %Integer;<br>Property NOME As %String;<br>Property ENDERECO As %String;<br>Property TELEFONE As %String;<br>Property CNPJ As %String;<br>Property IE As %String;<br><br>Property ESTAGIARIA As bd.ESTAGIARIA;<br>} |

O relacionamento N:N entre as classes *Estante* e *Bibliotecárias* necessita de uma classe intermediária (denominada BIBLIOTECARIASESTANTE) para se efetivar. Ele é apresentado na Tabela 12.

Tabela 12. Geração de relacionamento N:N entre as classes Estante e Bibliotecárias

| ODMG | Caché |
|---|---|
| class ESTANTE<br>{<br>attribute long CODESTANTE;<br>attribute string LOCALIZACAO;<br>attribute long CAPACIDADE;<br><br>relationship set<BIBLIOTECARIASESTANTE> referencia<br>inverse BIBLIOTECARIASESTANTE::eReferenciadoPor;<br><br>relationship set<EXEMPLAR> referencia<br>inverse EXEMPLAR::eReferenciadoPor;<br>}; | Class bd.ESTANTE Extends %Persistent<br>{<br>Property CODESTANTE As %Integer;<br>Property LOCALIZACAO As %String;<br>Property CAPACIDADE As %Integer;<br><br>Relationship BIBLIOTECARIASESTANTE As bd.BIBLIOTECARIASESTANTE [ Cardinality = many, Inverse = ESTANTE ];<br><br>Relationship EXEMPLAR As bd.EXEMPLAR [ Cardinality = many, Inverse = ESTANTE ];<br>} |

(continua)

Tabela 12. Geração de relacionamento N:N entre as classes Estante e Bibliotecárias (conclusão)

| ODMG | Caché |
|---|---|
| class BIBLIOTECARIAS<br>{<br>attribute long CODBIBLIOTECARIA;<br>attribute string NOME;<br>attribute string CPF;<br>attribute string RG;<br>attribute string ENDERECO;<br>attribute string TELEFONE;<br>attribute double SALARIO;<br><br>relationship set<BIBLIOTECARIAS> referencia<br>inverse BIBLIOTECARIAS::eReferenciadoPor;<br><br>relationship BIBLIOTECARIAS referencia<br>inverse BIBLIOTECARIAS::eReferenciadoPor;<br><br>relationship set<BIBLIOTECARIASESTANTE> referencia<br>inverse BIBLIOTECARIASESTANTE::eReferenciadoPor;<br>}; | Class bd.BIBLIOTECARIAS Extends %Persistent<br>{<br>Property CODBIBLIOTECARIA As %Integer;<br>Property NOME As %String;<br>Property CPF As %String;<br>Property RG As %String;<br>Property ENDERECO As %String;<br>Property TELEFONE As %String;<br>Property SALARIO As %Numeric;<br><br>Relationship BIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = many, Inverse = possuiBIBLIOTECARIAS ];<br><br>Relationship possuiBIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = one, Inverse = BIBLIOTECARIAS ];<br><br>Relationship BIBLIOTECARIASESTANTE As bd.BIBLIOTECARIASESTANTE [ Cardinality = many, Inverse = BIBLIOTECARIAS ];<br>} |
| class BIBLIOTECARIASESTANTE<br>{<br>relationship set<ESTANTE> referencia<br>inverse ESTANTE::eReferenciadoPor;<br><br>relationship set<BIBLIOTECARIAS> referencia<br>inverse BIBLIOTECARIAS::eReferenciadoPor;<br>}; | Class bd.BIBLIOTECARIASESTANTE Extends %Persistent<br>{<br>Relationship ESTANTE As bd.ESTANTE [ Cardinality = one, Inverse = BIBLIOTECARIASESTANTE ];<br><br>Relationship BIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = one, Inverse = BIBLIOTECARIASESTANTE ];<br>} |

A Tabela 12 ainda apresenta a definição de um auto-relacionamento 1:N para a classe *Bibliotecarias*. Esta classe contém as duas participações deste relacionamento, as quais referenciam a própria classe.

## 5.5 ANÁLISE DOS RESULTADOS OBTIDOS

Durante o desenvolvimento desta pesquisa surgiram inúmeras dificuldades. A principal delas foi a falta de livros publicados na área de BDOO, de forma que esta foi superada com artigos científicos, trabalhos de conclusão de curso, dissertações e teses de mestrado e teses de doutorado.

Também foram encontradas dificuldades na compreensão do padrão ODMG, pois o mesmo não deixa claro muitas informações, como a descrição dos tipos de dados suportados. Desta forma, a definição da compatibilidade destes com outros modelos foi realizada intuitivamente e por meio de pesquisas a outros trabalhos já realizados na área.

Em decorrência desta dificuldade, surgiu outra: a de como escrever (no script resultante) os tipos de dados do Firebird e do Oracle, quando não fossem encontrados tipos compatíveis com o padrão ODMG e o SGBD Caché. A solução encontrada foi escrevê-los como *UNDEFINED* para que o desenvolvedor possa identificá-los e tratá-los posteriormente.

No processo de escrita dos relacionamentos 1:1 e N:N utilizando CDL, a qual não possui suporte direto a estes relacionamentos, houveram dificuldades em definir como estes seriam realizados. Este problema foi solucionado com a colaboração de desenvolvedores que trabalham com o SGBD Caché e que apresentaram uma solução para a definição de cada um destes relacionamentos.

Nos testes realizados na ferramenta foram utilizados pequenos BD já utilizados na primeira versão desenvolvida por Samuel Brognoli e novos que foram obtidos de exemplos dos padrões, entre outros. Em tais testes a ferramenta obteve sucesso.

Como resultados do teste realizado com o esquema de BD de uma Biblioteca, apresentado anteriormente em seu diagrama E-R na Figura 21, tem-se os Apêndices A e B. O Apêndice A possui o *script* completo gerado pela ferramenta para o padrão ODMG. No Apêndice B está o *script* gerado para o SGBD Caché.

Diante do estudo, dos testes realizados e dos resultados obtidos com os mesmos, conclui-se que o objetivo geral e os objetivos específicos foram atingidos, uma vez que foram gerados *scripts* correspondentes OO para modelos relacionais utilizando metadados. E ainda os *scripts* dos testes gerados para o Caché foram executados corretamente na interface de desenvolvimento do SGBD denominada *Caché Studio*.

# CONCLUSÃO

O uso de modelo OO para BD é de grande valia ao se trabalhar com o armazenamento de grandes itens de dados com objetos complexos. E ainda, utilizando-se o mesmo paradigma durante todo o projeto deixa de existir a fronteira conceitual entre programação e BD, utilizando-se um modelo conceitual unificado, obtendo-se principalmente produtividade e redução de erros.

O alcance dos objetivos desta pesquisa foi conseguido com estudo dos BDOO, seu padrão e suas diferenças para com os BDR. Ainda foi estudada a CDL para a geração do script do SGBD Caché. Também foi necessária a compreensão das estruturas de armazenamento da ferramenta de engenharia reversa para sua adaptação as necessidades da transformação.

O desenvolvimento do trabalho foi subdividido em várias etapas, as quais iniciaram com o levantamento bibliográfico, posteriormente com os estudos já citados e ainda de trabalhos correlatos. Ainda, a ferramenta de apoio à engenharia reversa teve de ser adequada às necessidades deste trabalho para posteriormente implementar o método de transformação e, finalmente, realizar os testes práticos.

Vários desafios foram encontrados, principalmente no levantamento bibliográfico, que para ser realizado necessitou de pesquisa em artigos científicos, trabalhos de conclusão de curso, dissertações e teses de mestrado e teses de doutorado, publicados em congressos e em bases de dados na internet.

Outra dificuldade encontrada, relacionada com a compatibilidade dos tipos de dados, também foi superada com a definição de um tipo que possibilita uma posterior identificação destes no *script* para então serem tratados pelo desenvolvedor. Também, a

definição de relacionamentos que a CDL não especifica, foram escritos conforme orientação de profissionais da área.

Desta forma, pode-se concluir que os objetivos desta pesquisa foram alcançados, uma vez que os estudos foram realizados e a ferramenta realizou a transformação do modelo relacional para o OO, gerando os *scripts* correspondentes para o padrão ODMG e para o SGBD Caché, sendo que estes foram executados corretamente na interface de desenvolvimento *Caché Studio*.

Diante dos resultados obtidos, podem-se sugerir alguns trabalhos futuros:

a) adicionar mais funcionalidades na ferramenta para permitir a adição de métodos, por exemplo, nas classes;
b) desenvolver a geração de diagrama UML para o modelo OO;
c) adaptar a ferramenta para poder suportar outros SGBDR e outros SGBDOO.

# REFERÊNCIAS


BORBA, Sueli de Fátima Poppi. **Metodologia para implantação de modelos multidimensionais em banco de dados orientado a objetos.** 2006. 228 f. Tese (Doutorado) - Curso de Engenharia de Produção, Universidade Federal de Santa Catarina, Florianópolis, 2006. Disponível em: <http://www.tede.ufsc.br/tedesimplificado//tde_busca/arquivo.php?codArquivo=39>. Acesso em: 15 jun. 2009.

BOSCARIOLI, Clodis et al. Uma reflexão sobre banco de dados orientados a objetos. In: CONGRESSO DE TECNOLOGIAS PARA GESTÃO DE DADOS E METADADOS DO CONE SUL, 4., 2006, Ponta Grossa. **Artigo**. Ponta Grossa: UEPG, 2006. p. 2 - 8. Disponível em: <http://conged.deinfo.uepg.br/artigo4.pdf>. Acesso em: 21 abr. 2009.

BROGNOLI, Samuel Nicolau. **Ferramenta de apoio a engenharia reversa de bancos de dados relacionais**. 2008. 87 f. Trabalho de Conclusão de Curso (Bacharel) - Curso de Ciência da Computação, Universidade do Extremo Sul Catarinense, Criciúma, 2008.

CATTELL, R. G. G; BARRY, Douglas K. (Ed.). **The Object Data Standard:** ODMG 3.0. San Francisco: Morgan Kaufmann, 2000.

CHEN, Peter. **Gerenciando banco de dados:** a abordagem entidade-relacionamento para projeto lógico. São Paulo: Mcgraw-hill, 1990. 80 p.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados:** fundamentos e aplicações. 3. ed. Rio de Janeiro: LTC, 2002. 837 p.

GELATTI, Paôla Cristina. **Extensão do padrão ODMG para suportar tempo e versões.** 2002. 133 f. Dissertação (Mestrado) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002. Disponível em: <http://www.lume.ufrgs.br/handle/10183/3372>. Acesso em: 26 maio 2009.

HARRINGTON, Jan L. **Projetos de bancos de dados relacionais:** teoria e prática. 2. ed. Rio de Janeiro: Campus, 2002. 360 p.

HERNANDEZ, Michael J; PARENTI, Patrízia Tallia; TOTELLO, João Eduardo Nóbrega. **Aprenda a projetar seu próprio banco de dados.** São Paulo: Makron Books, 2000. 411 p.

HEUSER, Carlos Alberto. **Projeto de banco de dados.** 4. ed. Porto Alegre: Sagra Luzzatto, 2001. 204 p.

INTERSYSTEMS. **InterSystems Caché Technology Guide.** 2007. Disponível em: <http://www.intersystems.com/cache/technology/techguide/cache_tech-guide_01.html#05>. Acesso em: 20 maio 2009.

_____. **InterSystems Caché:** Um banco de dados de terceira geração. 2002. Disponível em: <http://www.intersystems.com.br/isc/downloads/vantagensbeneficios/WritePaperBDdeterceiraGeracao.pdf>. Acesso em: 20 maio 2009.

_____. **InterSystems Online Documentation.** 2009. Disponível em: < http://docs.intersystems.com/cache20091/csp/docbook/DocBook.UI.Page.cls >. Acesso em: 17 jun. 2009.

_____. **Introduction to Caché.** 2008. Disponível em: <http://www.intersystems.com/cache/documentation/2008_1/pdfs/GIC.pdf>. Acesso em: 01 jun. 2009.

JONES, Leonardo; ALBERTO, Luis; LIMA, Murilo de. **Bancos de dados orientados a objetos.** Universidade Federal da Bahia – Instituto de Matemática (UFBA). Disponível em: <https://disciplinas.dcc.ufba.br/pub/MATA60/WebHome/bdoo_20072.pdf>. Acesso em: 01 jun. 2009.

KAMADA, Aqueo. **Transformação de esquema relacional para esquema orientado a objeto em sistemas de bancos de dados heterogêneos.** 1996. 128 f. Dissertação (Mestrado) - Curso de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, 1996. Disponível em: <http://libdigi.unicamp.br/document/?code=vtls000103503>. Acesso em: 17 mar. 2009.

KHOSHAFIAN, Setrag. **Banco de dados orientado a objetos**. Rio de Janeiro: Infobook, 1994. 353 p.

KORTH, Henry F.; SILBERSCHATZ, Abraham. **Sistema de bancos de dados.** 2. ed. São Paulo: Makron Books, 1995. 754 p.

KRAMEL, Danilo. **Protótipo de software para a geração de código CDL através do repositório da ferramenta case System Architect.** 2000. 83 f. Trabalho de Conclusão de Curso (Bacharel) - Curso de Ciências da Computação, Universidade Regional de Blumenau, Blumenau, 2000. Disponível em: <http://campeche.inf.furb.br/tccs/2000-II/2000-2danilokramelvf.pdf>. Acesso em: 25 jun. 2009.

MACHADO, Felipe Nery Rodrigues; ABREU, Maurício Pereira de. **Projeto de banco de dados:** uma visão prática. 8. ed. São Paulo: Érica, 2002. 298 p.

MARTIN, James; ODELL, James J. **Análise e projeto orientados a objeto.** São Paulo: Makron Books, 1995. 639 p.

MARTIN, James. **Princípios de análise e projeto baseado em objetos**. Rio de Janeiro: Campus, 1994. 486 p.

RAMAKRISHNAN, Raghu; GEHRKE, Johannes. **Sistemas de gerenciamento de banco de dados.** 3. ed. São Paulo: Mcgraw-hill, 2008. 884 p.

RAMALHO, José Antonio. **Oracle 10g.** São Paulo: Thomson, 2005.

ROCHA, Carlos A. S. et al. Desenvolvimento de ferramenta de apoio à atualização de sistemas de informação baseados no sistema de banco de dados pós-relacional Caché. In: ESCOLA REGIONAL DE BANCO DE DADOS - ERBD, 4., 2008, Florianópolis. **Artigo.** Criciúma: UFSC, 2008. p. 2 - 3. Disponível em: <http://www.inf.ufsc.br/erbd2008/artigos/4.pdf>. Acesso em: 20 maio 2009.

SANTANA, Alex Novaes de; NUNES, Bruno Rios Patriarca; OLIVEIRA, Vitor Fernandes Ribeiro de. **Banco de dados orientado a objetos.** 2008. Artigo. Disponível em: <https://disciplinas.dcc.ufba.br/pub/MATA60/WebHome/bdoo_2008.1.pdf>. Acesso em: 13 abr. 2009.

SILBERSCHATZ, Abraham; KORTH, Henry F; SUDARSHAN, S. **Sistema de banco de dados.** 5. ed. Rio de Janeiro: Elsevier, 2006. 781 p.

SILVA, Anderson Evandro Simeão da. **Uma análise comparativa das tecnologias de banco de dados relacional e de banco de dados nativamente orientado a objetos.** 2003. 207 f. Tese (Mestrado) - Curso de Engenharia Eletrônica e Computação, Instituto Tecnológico de Aeronáutica, São José Dos Campos, 2003. Disponível em: <http://www.bd.bibl.ita.br/tde_busca/arquivo.php?codArquivo=455>. Acesso em: 25 jun. 2009.

SILVA, Arídio. **Dominando a tecnologia de objetos:** programação - implementação - soluções - problemas. Rio de Janeiro: BookExpress, 2002. 478 p.

WILDEROM, Bastiaan Pieter Marinus; WILDEROM, Stella Martinez. **Firebird/InterBase 6.0:** cliente/servidor com Delphi 6 - tópicos avançados. São Paulo: Érica, 2002.

# APÊNDICE A – SCRIPT ODMG DO ESQUEMA DE UMA BIBLIOTECA

Script de um BDR de uma biblioteca gerado pela ferramenta para o padrão ODMG.

```
class LIVRO
{
attribute long CODLIVRO;
attribute string TITULO;
attribute string AUTOR;
attribute date ANO;
attribute string EDITORA;
attribute long EDICAO;
attribute string IDIOMA;

relationship set<EXEMPLAR> referencia
inverse EXEMPLAR::eReferenciadoPor;


};

class EXEMPLAR
{
attribute long CODEXEMPLAR;
attribute string SITUACAO;
attribute date DATAAQUISICAO;

relationship ESTANTE referencia
inverse ESTANTE::eReferenciadoPor;

relationship LIVRO referencia
inverse LIVRO::eReferenciadoPor;


};

class ESTANTE
{
attribute long CODESTANTE;
attribute string LOCALIZACAO;
attribute long CAPACIDADE;

relationship set<BIBLIOTECARIASESTANTE> referencia
inverse BIBLIOTECARIASESTANTE::eReferenciadoPor;

relationship set<EXEMPLAR> referencia
inverse EXEMPLAR::eReferenciadoPor;


};

class BIBLIOTECARIAS
{
attribute long CODBIBLIOTECARIA;
attribute string NOME;
attribute string CPF;
```

```
attribute string RG;
attribute string ENDERECO;
attribute string TELEFONE;
attribute double SALARIO;

relationship set<BIBLIOTECARIAS> referencia
inverse BIBLIOTECARIAS::eReferenciadoPor;

relationship BIBLIOTECARIAS referencia
inverse BIBLIOTECARIAS::eReferenciadoPor;

relationship set<BIBLIOTECARIASESTANTE> referencia
inverse BIBLIOTECARIASESTANTE::eReferenciadoPor;


};

class EFETIVA extends BIBLIOTECARIAS
{
attribute date DATAADMISSAO;
attribute string FORMACAO;


};

class ESTAGIARIA extends BIBLIOTECARIAS
{
attribute long TEMPOESTAGIO;
attribute date INICIOESTAGIO;
attribute date TERMINOESTAGIO;

relationship INSTITUICAO referencia
inverse INSTITUICAO::eReferenciadoPor;


};

class INSTITUICAO
{
attribute long CODINSTITUICAO;
attribute string NOME;
attribute string ENDERECO;
attribute string TELEFONE;
attribute string CNPJ;
attribute string IE;

relationship ESTAGIARIA referencia
inverse ESTAGIARIA::eReferenciadoPor;


};

class BIBLIOTECARIASESTANTE
{

relationship set<ESTANTE> referencia
inverse ESTANTE::eReferenciadoPor;

relationship set<BIBLIOTECARIAS> referencia
inverse BIBLIOTECARIAS::eReferenciadoPor;


};
```

## APÊNDICE B – SCRIPT CACHÉ DO ESQUEMA DE UMA BIBLIOTECA

Script de um BDR de uma biblioteca gerado pela ferramenta para o SGBD Caché.

```
Class bd.LIVRO Extends %Persistent
{

Property CODLIVRO As %Integer;

Property TITULO As %String;

Property AUTOR As %String;

Property ANO As %Date;

Property EDITORA As %String;

Property EDICAO As %Integer;

Property IDIOMA As %String;

Relationship EXEMPLAR As bd.EXEMPLAR [ Cardinality = many, Inverse = LIVRO ];

}

Class bd.EXEMPLAR Extends %Persistent
{

Property CODEXEMPLAR As %Integer;

Property SITUACAO As %String;

Property DATAAQUISICAO As %Date;

Relationship ESTANTE As bd.ESTANTE [ Cardinality = one, Inverse = EXEMPLAR ];

Relationship LIVRO As bd.LIVRO [ Cardinality = one, Inverse = EXEMPLAR ];

}

Class bd.ESTANTE Extends %Persistent
{

Property CODESTANTE As %Integer;

Property LOCALIZACAO As %String;

Property CAPACIDADE As %Integer;

Relationship BIBLIOTECARIASESTANTE As bd.BIBLIOTECARIASESTANTE [ Cardinality = many,
Inverse = ESTANTE ];

Relationship EXEMPLAR As bd.EXEMPLAR [ Cardinality = many, Inverse = ESTANTE ];

}

Class bd.BIBLIOTECARIAS Extends %Persistent
```

```
{
Property CODBIBLIOTECARIA As %Integer;

Property NOME As %String;

Property CPF As %String;

Property RG As %String;

Property ENDERECO As %String;

Property TELEFONE As %String;

Property SALARIO As %Numeric;

Relationship BIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = many, Inverse = possuiBIBLIOTECARIAS ];

Relationship possuiBIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = one, Inverse = BIBLIOTECARIAS ];

Relationship BIBLIOTECARIASESTANTE As bd.BIBLIOTECARIASESTANTE [ Cardinality = many, Inverse = BIBLIOTECARIAS ];

}

Class bd.EFETIVA Extends bd.BIBLIOTECARIAS
{

Property DATAADMISSAO As %Date;

Property FORMACAO As %String;

}

Class bd.ESTAGIARIA Extends bd.BIBLIOTECARIAS
{

Property TEMPOESTAGIO As %Integer;

Property INICIOESTAGIO As %Date;

Property TERMINOESTAGIO As %Date;

}

Class bd.INSTITUICAO Extends %Persistent
{
Property CODINSTITUICAO As %Integer;

Property NOME As %String;

Property ENDERECO As %String;

Property TELEFONE As %String;

Property CNPJ As %String;

Property IE As %String;

Property ESTAGIARIA As bd.ESTAGIARIA;
```

```
}

Class bd.BIBLIOTECARIASESTANTE Extends %Persistent
{

Relationship ESTANTE As bd.ESTANTE [ Cardinality = one, Inverse = BIBLIOTECARIASESTANTE ];

Relationship BIBLIOTECARIAS As bd.BIBLIOTECARIAS [ Cardinality = one, Inverse = BIBLIOTECARIASESTANTE ];

}
```

# ANEXO A – GUIA DE REFERÊNCIA CDL

Danilo Kramel elaborou em 2000 um Trabalho de Conclusão de Curso na Universidade Regional de Blumenau no curso de Ciência da Computação onde estão descritas as principais características para a criação de classes por meio da CDL que podem ser visualizadas a seguir:

## COMENTÁRIOS

Comentários não podem aparecer dentro de alguma parte do arquivo CDL que é formado por código *Caché Object Script* (COS).

**Comentário de Linha**: / /

**Comentários em Geral**: abertura com /* fechamento com *\

## DELIMITADOR DE BLOCO DE COMANDOS

Abertura com {  fechamento com }

## COMANDO CREATE CLASS

**Sintaxe**: CREATE CLASS **classname** {classkeyword1; classkeyword2; ...}

O comando CREATE CLASS cria uma nova definição de classe no *Class Dictionary*. Se a classe que esta sendo definida já existe, um erro ocorre. **Classname** é uma palavra alfanumérica, que começa com a letra ou sinal de percentual (%), e define o nome da

classe que esta sendo criada. **Classkeywordn** correspondem a palavras reservadas para definição de classes e são explicadas a seguir.

COMANDO DROP CLASS

**Sintaxe**: DROP CLASS **classname**

O comando DROP CLASS apaga uma definição de classe no Class Dictionary. **Classname** define o nome da classe à ser apagada.

ABSTRACT

**Sintaxe**: ABSTRACT;

Define que a classe é abstrata.

FINAL

**Sintaxe**: FINAL;

Define que a classes é final. Não pode ter subclasses.

PERSISTENT

**Sintaxe**: PERSISTENT;

Define que a classe é persistente.

SUPER

**Sintaxe**: SUPER = **classename1**, **classname2**, ..., **classnamen**;

Define a(s) superclasse(s) da classe.

SYSTEM

**Sintaxe**: SYSTEM;

Define que a classe é uma classe de sistema.

DESCRIPTION

**Sintaxe**: DESCRIPTION = **descrição**

Define uma descrição para a classe.

PARAMETER

**Sintaxe**: PARAMETER **ParameterName** {**ParameterDefinition**}

Define os parâmetros da classe.

**ParameterName** define o nome do parâmetro.

**ParameterDefinition** possui uma sintaxe similar às definições de classe e possui apenas uma palavra reservada:

a) **DEFAULT**: define o nome do atributo cujo valor é padrão para esse parâmetro ou o valor do parâmetro propriamente dito.

**Sintaxe**: **DEFAULT** = **AttributeName** onde:

- **AttributeName**: é o nome do atributo de valor padrão ou um valor propriamente dito (um *String* ou *Integer*, por exemplo).

ATTRIBUTE

**Sintaxe**: ATTRIBUTE **AtributeName** {**AtributeDefinition**}

Define as propriedades/atributos da classe.

**AtributeName** define o nome do atributo/propriedade.

**AtributeDefinition** define as características do atributo. Também possui uma sintaxe muito similar às definições de classe e pode receber (entre várias) as seguintes palavras reservadas:

a) **CALCULATED**: define que este é um atributo calculado.

**Sintaxe**: CALCULATED;

b) **FINAL**: define que o atributo é final e não pode ser sobreposto nas subclasses.

**Sintaxe**: FINAL;

c) **PRIVATE**: define que o atributo é privado.

**Sintaxe**: PRIVATE;

d) **PUBLIC**: define que o atributo é público.

**Sintaxe**: PUBLIC;

e) **REQUIRED**: define que o atributo é requerido.

**Sintaxe**: REQUIRED;

f) **TYPE**: define o tipo de dado do atributo. Entre os tipos suportados pelo Caché estão:

**- %Boolean**;

**- %Date**;

- **%Float**;

- **%Satus**;

- **%Time**;

- **%String;**

- **%Name.**

**Sintaxe**: TYPE = **TYPEDefinition**;

g) **INITIAL**: define uma expressão em **COS** ou um valor literal que é assumido como valor do atributo.

**Sintaxe**: INITIAL = **Expression**;

METHOD

**Sintaxe**: METHOD **MethodName** (**Pars**) {**MedthodDefinition**} ou

METHOD **MethodName FormalSpec** (**Pars**) {**MedthodDefinition**}

Define os métodos da classe.

**MethodName** define o nome do método.

**FormalSpec** define o separador padrão para os parâmetros formais desse método. O separador default é a "," (vírgula).

**Pars** define os parâmetros formais do método. Esses parâmetros são passados ao método quando de sua chamada. **Pars** consiste de uma cadeia de caracteres no formato:

**FormaParameter** : **FormalType** = **DefaultValue**, separadas pelo **FormalSpec**.

**MethodDefinition** consiste de uma série de palavras reservadas para definição das características do método e também possui uma sintaxe similar as de definições de classe. Dentre suas palavras reservadas estão:

a) **CALL**: define o nome do programa que deve ser executado quando da chamada deste método.

**Sintaxe**: CALL = **ProgramName**;

b) **CLASSMETHOD**: define que o método é um método de classe.

**Sintaxe**: CLASSMETHOD;

c) **CODE**: define o código COS do método.

**Sintaxe**: CODE = {**Bloco de Comandos**}

d) **FINAL**: define que o método é final e não pode ser sobreposto na subclasse.

**Sintaxe**: FINAL;

e) **PRIVATE**: define que o método é privado.

**Sintaxe**: PRIVATE;

f) **PUBLIC**: define que o método é publico.

**Sintaxe**: PUBLIC;

g) **RETURNTYPE**: define o tipo de dado do valor retornado pelo método. Quando RETURNTYPE não está presente na declaração do método ou recebe o valor nulo (“”) o método não retorna nenhum valor.

**Sintaxe**: RETURNTYPE = **DataType**;

h) **DESCRIPTION**: define uma descrição para o método.

**Sintaxe**: DESCRIPTION = “**Descrição do Método**”;