Transformação de Esquema Relacional para Esquema Orientado a Objeto em Sistemas de Bancos de Dados Heterogêneos

Aqueo Kamada

Este exemplar corresponde à redação final da tese defendida por Aqueo Kamada e aprovada pela Comissão Julgadora em 17 / 01 / 1996

Orientador

Unicamp - FEE - DCA
Campinas, Janeiro 1996

UNICAMP
BIBLIOTECA CENTRAL

Transformação de Esquema Relacional para Esquema Orientado a Objeto em Sistemas de Bancos de Dados Heterogêneos

Aqueo Kamada

Dissertação apresentada à FEE-UNICAMP para obtenção do grau de mestre em Engenharia Elétrica

Orientadora: **Beatriz Daltrini**
Unicamp/Campinas/SP/Brasil

Co-Orientador: Weiyi Meng
SUNY/Binghamton/NY/U.S.A.

Unicamp - FEE - DCA
Campinas, Janeiro 1996

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

K127t

Kamada, Aqueo
Transformação de esquema relacional para esquema orientado a objeto em sistemas de banco de dados heterogêneos / Aqueo Kamada.--Campinas, SP: [s.n.], 1996.

Orientadores: Beatriz Daltrini, Weiyi Meng.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica.

1. Banco de dados distribuidos. 2. Banco de dados orientado a objetos. 3. QUERY (Sistema de recuperação da informação). I. Daltrini, Beatriz. II. Meng, Weiyi. III.Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica. IV. Título.

À Querida Fátima
e aos Lindinhos Marcela, Diogo, Frederico e Hugo,
pelo apoio e compreensão
A.K.

Agradecimentos

Ao **Weiyi**, pelo seu sempre pronto apoio e orientação durante minha passagem pela SUNY, e depois via "e-mail".

À **Beatriz**, pela ajuda e orientação a esse trabalho.

Ao **CTI**, como centro de pesquisa e aos **amigos e colegas do IC** pelo apoio técnico. Em especial ao **Romildo**, ao **Miguel** e ao **Marcos**, pelo incentivo e apoio, fundamentais na realização desse trabalho.

Ao **CNPq**, pelo suporte financeiro parcial.

À **Fátima**, pelo apoio especial e compreensão, e por desempenhar o papel de pai em meu lugar, em muitas ocasiões. Obrigado!

À **Marcela**, ao **Diogo**, ao **Frederico** e ao **Hugo**, pela paciência quando eu estava ausente, quando não pude consertar um brinquedo, ou jogar bola, ou nadar, ou ajudar nas lições, ou instalar um software ou game, ou andar de mini-bugue, ou ....

Às minhas cunhadas **Odete** e **Elza** (ou Zelda ?) e à minha sobrinha **Camila**, pelo sempre carinhoso incentivo e apoio incondicional na realização desse trabalho, pelo que fazem por mim e por serem tão especiais.

À minha amiga **Ana Luisa**, pelas sugestões e ajuda na tradução desse trabalho (originalmente em inglês).

À **Todos** que participaram das minhas alegrias e tristezas, nesse período.

Obrigado!
A.K.

RESUMO

Esse trabalho apresenta um algoritmo que transforma esquemas relacionais em esquemas orientados a objetos num sistema de múltiplos bancos de dados heterogêneos. Nesse contexto, transformação de esquemas é um passo necessário antes de se fazer processamento de comandos de consulta. A implementação do algoritmo tem as seguintes características especiais: transforma esquemas entre dois sistemas de bancos de dados comerciais, especificamente de esquemas relacionais Sybase para esquemas orientados a objetos O2; inicia sessões interativas limitadas com o usuário, quando o transformador de esquemas necessita de informações adicionais, embora a maioria das transformações seja automática; 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.

ABSTRACT

This work presents an algorithm that transforms relational schemas to object-oriented schemas in a heterogeneous multidatabase system. In this context, schema transformation is a necessary step before query processing. The implementation of the algorithm has the following special features: it transforms schemas between two commercial database systems, specifically from Sybase relational schemas to O2 object-oriented schemas; it initiates limited interactive sessions between the schema transformer and its user when additional input to the schema transformer is necessary, although most of the transformation is automated; the transformed schemas are typical object-oriented schemas that may contain class hierarchies, composition hierarchies, set attributes and subtuple constructs; and the number of classes in the transformed object-oriented schema is relatively small, compared with the number of relations in the equivalent relational schema.

Índice

CAPÍTULO 1 *Introdução 1*

1.1 Motivação 1
1.2 Benefícios 3
1.3 Objetivo 3
1.4 Organização 4

CAPÍTULO 2 *Arquitetura de Sistemas de Múltiplos Bancos de Dados 6*

2.1 Sistemas de Múltiplos Bancos de Dados 6
2.1.1 API de Banco de Dados 7
2.1.2 Generalidade de uma API 7
2.1.3 Abordagem 1: Esquema e Linguagem Globais 8
2.1.4 Abordagem 2: Criação de Visão Temporária 9
2.2 Processamento de Comando de Consulta Global 9
2.2.1 Decomposição de Comando de Consulta Global 10
2.2.2 Tradução de Comando de Consulta Global 11
2.3 Obtenção de Esquema Global 12
2.4 Resumo do Capítulo 2 13

CAPÍTULO 3 *Modelos de Dados 14*

3.1 Modelo Entidade-Relacionamento 15
3.1.1 Entidades 16
3.1.2 Tipos de Atributos e Domínios 18
3.1.3 Chaves 19
3.1.4 Relacionamentos 19
3.1.5 Restrições Estruturais 23
3.2 Modelo Entidade-Relacionamento Estendido 24
3.2.1 Relacionamento Subtipo/Supertipo 26
3.2.2 Herança de Atributos 26
3.2.3 Especialização e Generalização 27
3.3 Modelo de Dados Relacional 28
3.3.1 Relações, Atributos, Tuplas e Domínios 28

3.3.2 Esquemas de Bancos de Dados Relacionais 30
3.3.3 Restrições do Modelo Relacional 30
3.3.3.1 Restrições de chave 30
3.3.3.2 Restrições de Integridade Referencial e Chaves estrangeiras 32
3.3.4 Transformação de Esquema ERE para Esquema Relacional 33
3.3.4.1 Regra 1: Transformar Entidade em Relação 33
3.3.4.2 Regra 2: Transformar Atributo set 33
3.3.4.3 Regra 3: Transformar Atributo Composto 33
3.3.4.4 Regra 4: Identificar Relacionamentos Um-para-Um e Um-para-muitos 34
3.3.4.5 Regra 5: Identificar Relacionamentos muitos-para-muitos 34
3.3.4.6 Regra 6: Identificar Relacionamentos Subtipo/Supertipo 35
3.4 Modelo de Dados Orientado a Objetos 35
3.4.1 Objeto, Identidade de Objeto e Objeto Complexo 35
3.4.2 Encapsulamento 38
3.4.3 Classe, Hierarquia de Classes, Herança e Conceitos Relacionados 40
3.4.4 Overriding, Polimorfismo e Late Binding 42
3.5 Resumo do Capítulo 3 44

CAPÍTULO 4 *Transformação de Esquema Relacional para Esquema Orientado a Objetos 45*

4.1 Convenções e Restrições 46
4.2 O Algoritmo de Transformação de Esquemas 47
4.2.1 Transformando relações para classes 48
4.2.1.1 REGRA 1: Transformar cada relação em classe 48
4.2.1.2 Exemplo para REGRA 1 48
4.2.2 Identificando relacionamento subclasse/superclasse 49
4.2.2.1 REGRA 2 : Identificar relacionamento subclasse/superclasse 49
4.2.2.2 Exemplos para REGRA 2 50
4.2.3 Identificando e transformando relações transformadas de atributos compostos 52
4.2.3.1 REGRA 3 : Identificar atributos compostos 52
4.2.3.2 Exemplos para REGRA 3 53
4.2.4 Identificando e transformando relações transformadas de atributo conjunto 55

4.2.4.1 REGRA 4 : Identificar e transformar atributo conjunto 55
4.2.4.2 Exemplo para REGRA 4 56
4.2.5 Identificando e transformando relações transformadas de relacionamentos unários muitos-para-muitos 57
4.2.5.1 REGRA 5 : Identificar e transformar relacionamentos unários muitos-para-muitos 57
4.2.5.2 Exemplos para REGRA 5 57
4.2.6 Identificando e transformando relação transformada de relacionamento binário muitos-para-muitos 60
4.2.6.1 REGRA 6 : Identificar e transformar relacionamento binário muitos-para-muitos 60
4.2.6.2 Exemplos para REGRA 6 61
4.2.7 Expandindo hierarquia de composição 63
4.2.7.1 REGRA 7: Expandir hierarquia de composição 63
4.2.7.2 Exemplo para REGRA 7 64
4.3 Adicionando métodos ao esquema OO transformado 65
4.4 Resumo do Capítulo 4 67

CAPÍTULO 5 *Implementação de um Transformador de Esquemas 69*

5.1 Preâmbulo para a Estratégia de Projeto de Implementação 70
5.2 Estratégia do Projeto de Implementação 70
5.3 Especificação dos programas 73
5.3.1 Programa 1: squeeze.lex 73
5.3.2 Programa 2: tsyo2.c 75
5.3.2.1 Passo 1: Regra 1- Transformar todas as relações em classes 77
5.3.2.2 Passo 2: Regra 2 - Identificar relacionamentos ISA e relações como parte de outras
Regra 3 - Identificar Atributo Composto
Regra 4 - Identificar Atributo "set" 79
5.3.2.3 Passo 3: Regra 5 - Transformar relacionamento unário muito-pra-muito.
Regra 6 - Transformar relacionamento binário muito-pra-muito.
Regra 7 - Expandir hierarquia de composição. 87

5.4 O programa "script" makefile 94
5.5 O arquivo readme 96
5.6 Resumo do Capítulo 5 98

CAPÍTULO 6 *Conclusões & Pesquisas Futuras 101*

6.1 Conclusão relacionada com a API Orientada a Objetos 101
6.2 Conclusão relacionada com o Transformador de esquemas 103
6.3 Pesquisas Futuras 105
6.3.1 Criação de Métodos 105
6.3.2 Usabilidade das Regras de transformação 106

Referências 107

Apêndice A - Arquivo squeeze.lex
Filtra informações do dicionário de dados, tal como se encontra no Apêndice B, para obter somente as informações necessárias. 112

Apêndice B - Arquivo sp_help.out
Informações do Dicionário de Dados do Sybase. 115

Apêndice C - Arquivo syrelati
Esquema Relacional reconstituído a partir do banco de dados pubs2 modificado. 122

Apêndice D - Arquivo o2class
Esquema Orientado a Objetos O2 - transformado a partir do esquema relacional Sybase do pubs2 modificado.. 126

Lista de Figuras

Figura 2.1 API de Banco de Dados 7

Figura 2.2.a Processamento de comando de consulta global 10

Figura 2.2.b Obtenção de Esquema Global 12

Figura 3.1.a Notação para esquemas ER 16

Figura 3.1.b Esquema ER para o banco de dados pubs2 modificado 17

Figura 3.1.4.a Exemplo de Relacionamento unário e relacionamento binário 20

Figura 3.1.4.b Exemplo de Relacionamento Unário 1:1. 21

Figura 3.1.4.c Exemplo de Relacionamento Binário 1:1 - participação parcial e total. 21

Figura 3.1.4.d Exemplo de Relacionamento Unário 1:m 22

Figura 3.1.4.e Exemplo de Relacionamento Binário 1:m 22

Figura 3.1.4.f Exemplo de Relacionamento Unário (m:n) 23

Figura 3.1.4.g Exemplo de Relacionamento Binário (m:n). 23

Figura 3.2.b Exemplo de generalização/especialização. 27

Figura 3.3.a O esquema SGBD relacional pubs2 31

Figura 3.4.b Hierarquia de composição - Objetos definidos em termos de objetos. 38

Figura 3.4.c Encapsulamento de dados e do corpo do método 39

Figura 3.4.d Relacionamento subclasse/superclasse, modelando o relacionamento IS_A 41

Figura 3.4.e Hierarquia de Composição - Recursividade com a própria classe 42

Figura 3.4.f Exemplo de "overriding" e polimorfismo 43

Figura 5.1.1 Informações associadas a relação publishers no Sybase. 71

Figura 5.2.1 Saída do programa 1 - Informações necessárias para reconstituir a relação publishers. 71

Figura 5.2.2 Esquema Relacional reconstituído, para a relação publishers 72

Figura 5.2.3 Classe publishers no O2 - transformada do equivalente esquema relacional Sybase 72

Lista de Figuras

Figura 5.3.1.a Especificação do programa 1: squeeze.lex 74

Figura 5.3.2.a Esquema Relacional Pubs2 reconstituído 76

Figura 5.3.2.b Esquema OO O2 para pubs2, logo após somente a regra 1 ser executada 78

Figura 5.3.2.c Herança de classes modelando relacionamentos IS_A. 81

Figura 5.3.2.d Atributo copy como parte da classe authors 82

Figura 5.3.2.e Classe au_pix como parte da classe authors 82

Figura 5.3.2.f Relação publication modificada. 83

Figura 5.3.2.g Classe roysched como parte da classe publication. 83

Figura 5.3.2.h Relação roysched com somente um atributo além da PK title_id. 84

Figura 5.3.2.i Atributo lorange da classe roysched como parte da classe publication 84

Figura 5.3.2.j Atributo do tipo "set" 85

Figura 5.3.2.k Relação discounts com uma PK composta. 85

Figura 5.3.2.l Atributo do tipo set criado de um atributo PK composto. 86

Figura 5.3.2.m Esquema relacional course 88

Figura 5.3.2.n Classe course transformada das relações course e prerequisite 89

Figura 5.3.2.o Relação prerequisite com mais um atributo, além da PK. 89

Figura 5.3.2.p Classe course transformada das relações course e prerequisite modificada. 89

Figura 5.3.2.q1 Classe authors com relacionamento binário muito-pra-muito 90

Figura 5.3.2.q2 Classe publication com relacionamento binário muito-pra-muito. 91

Figura 5.3.2.r Classe publication composta pela classe publishers. 92

Figura 5.3.2.s Relação publication com uma FK composta para publishers. 92

Figura 5.3.2.t Classe publication com atributo publishers
substituindo pub_id e pub_name 93

Figura 5.3.2.u Esquema O2 para pubs2 94

Lista de Tabelas

Tabela 3.1 PUBLICATION 28

Tabela 3.3 Visão esquemática do objeto computador. 36

Tabela 3.2 Visão Simplificada de partes de um objeto 36

Tabela 5.3.2 Convenções para descrição do transformador de esquema. 77

CAPÍTULO 1 *Introdução*

1.1 Motivação

Atualmente muitas organizações utilizam diferentes tipos de sistemas de gerência de bancos de dados (SGBDs), por razões históricas e econômicas, e também porque um SGBD específico pode ser mais adequado para uma determinada aplicação. Entretanto, usuários de um SGBD num ambiente heterogêneo têm dificuldades de fazer acesso a diferentes SGBDs, de maneira uniforme, principalmente devido à incompatibilidade dos modelos de dados e à falta de uma linguagem de acesso padronizada, estável e eficiente. Estas dificuldades refletem diretamente na produtividade de desenvolvimento e na qualidade da aplicação, pois muitas redundâncias e replicações devem ser feitas para várias plataformas de banco de dados. Sistemas de gerência de bancos de dados relacionais (SGBDR), que são largamente utilizados na indústria e na pesquisa, são bem conhecidos, fáceis de usar, têm uma boa base teórica e parecem ser bons para a maioria das aplicações tradicionais em sistemas comerciais e administrativos. Entretanto, eles são insuficientes em termos de expressividade da linguagem de consulta para especificar consultas complexas, em termos de modelo de dados e funcionalidade, para aplicações avançadas que necessitem recursos gráficos, de multimídia e de CAD/CAM, por exemplo.

Idealmente, uma linguagem de consulta deve permitir que usuários novatos e inexperientes, com pouco conhecimento ou somente com pouco entendimento básico do esquema de dados, possam consultar o banco de dados. A linguagem relacional SQL (Structured Query Language), com sua sintaxe linear foi desenvolvida há mais de vinte anos atrás e desde então não foi substancialmente mudada. Apesar de terem acontecido alguns avanços, pouco foi feito para melhorar a funcionalidade e a expressividade do SQL. No padrão comumente chamado de SQL2 [20] algumas deficiências e inconsistências do SQL foram removidas, mas não houve uma melhora substancial nos aspectos funcionalidade e expressividade do SQL. O emergente SQL3 [29] que provavelmente será concluído em 1996 ou 1997, pretende melhorar as capacidades de modelamento e a linguagem de consulta através da introdução de características orientadas a objetos.

Os sistemas de gerência de bancos de dados orientados a objetos (SGBDOO), que estão surgindo como a nova geração dos SGBDs, podem alcançar esses requisitos, fornecendo facilidades para o gerenciamento de objetos em constante evolução e conectados por relações semânticas complexas [11, 23]. Eles também melhoram o entendimento das consultas e as tornam mais próximas da visão que o usuário tem do problema [1, 2, 3, 24]. Mas, apesar de se terem obtido alguns avanços conceituais nos modelos de dados orientados a objetos, na capacidade para modelar a semântica relacional estendida e nas linguagens de consulta, esses conceitos não estão bem firmados. Além disso, a tecnologia de banco de dados orientado a objetos não está, ainda, amplamente difundida, em parte devido à predominância dos SGBDs relacionais. A migração de sistemas relacionais existentes para sistemas orientados a objetos e o treinamento de usuários nessa nova tecnologia, apresentam investimentos muito altos.

Com uma visão mais pragmática, considerando que as organizações, por razões históricas, econômicas e práticas, continuarão a utilizar os SGBDs relacionais, pode-se melhorar o processo de especificação de consultas e também preparar "suavemente" a migração para uma abordagem totalmente orientada a objetos. Então, parece ser viável e oportuno se ter um **"front-end" de bancos de dados, orientado a objetos**, baseado em um único modelo de dados orientado a objetos e em uma única linguagem de consulta global orientada a objetos para se ter acesso a diferentes SGBDs locais com diferentes modelos de dados (relacional, orientado a objetos, etc.). Este conjunto, englobando o sistema front-end de banco de dados e os SGBDs locais é chamado de sistema de múltiplos bancos de dados

(SMBD). O "front-end" é também chamado de API - Application Program Interface. Doravante, denominaremos o "front-end" de banco de dados de **API de banco de dados**.

1.2 Benefícios

Os benefícios imediatos que podem ser obtidos com a construção de tal API de banco de dados orientado a objetos são que:

1) ela **diminui a "distância" entre a maneira com que o usuário expressa suas consultas e a linguagem de consulta de banco de dados**, como o SQL e
2) ela **provê uma interface padrão e uniforme para todas as plataformas de bancos de dados** compatíveis com esse padrão.

Consequentemente as organizações mencionadas anteriormente e os desenvolvedores de software terão a produtividade no desenvolvimento aumentada e a qualidade das aplicações melhorada, pois não terão que replicar esforços para várias plataformas de bancos de dados. Isso é importante, pois vai melhorar grandemente a aplicabilidade dessa abordagem, tornando-a útil para a maioria das áreas onde os sistemas de bancos de dados atuais estão em uso.

1.3 Objetivo

Em um sistema de múltiplos bancos de dados , quando um programa de aplicação usa um comando para recuperar uma informação (como *select* em SQL), a API de banco de dados decompõe e traduz esse comando para comandos de recuperação para o banco de dados local específico onde aquelas informações estão disponíveis. Essas tarefas de decomposição e tradução da consulta podem ser efetuadas em tempo de execução, dinamicamente, mas para fazer isso, antes o esquema do banco de dados local precisa ser transformado e integrado ao esquema global, tratado a nível da API de banco de dados. O objetivo desse trabalho é apresentar um **algoritmo que transforma um esquema relacional em um esquema orientado a objetos**. A implementação desse algoritmo tem 4 características especiais:

1) transforma esquemas entre dois sistemas de bancos de dados comerciais, isto é, de esquemas relacionais Sybase para esquemas orientados a objetos O2;

2) inicia sessões interativas limitadas entre o transformador de esquema e o usuário, quando são necessárias informações adicionais para o transformador de esquemas, embora a maior parte da transformação seja automatizada;
3) os esquemas transformados são esquemas orientados a objetos típicos que podem conter hierarquia de classes, hierarquia de composição, hierarquia de tuplas e conjuntos de atributos; e
4) o número de classes no esquema transformado orientado a objetos é relativamente pequeno se comparado ao número de relações no esquema relacional equivalente.

Apesar da implementação, apresentada no Capítulo 5, considerar dois sistemas de banco de dados comerciais específicos, o algoritmo, apresentado no Capítulo 4, é suficientemente genérico para ser aplicado em outros sistemas de banco de dados relacionais e orientados a objetos, pois a definição de esquema em banco de dados relacional está bastante padronizada e o Sybase é um dos líderes no mercado de SGBDRs, e o O2 suporta ODMG-93 - Object Database Management Group [32], que é o primeiro padrão para banco de dados orientado a objetos proposto por representantes de várias companhias líderes no mercado de SGBDOOs.

Além disso, adotando-se a mesma estratégia de projeto de implementação apresentada no Capítulo 5, isolando a tarefa de como buscar a informação relacional do dicionário de dados específico das tarefas do processo de transformação de esquema, a modificação necessária num transformador para outros SGBDs comerciais é reduzida aos ajustes nos formatos dos esquemas. Nesta primeira fase do projeto, considera-se o modelo de dados O2 como o modelo de dados global. Isso significa que pode-se ter um banco de dados relacional (Sybase) e um banco de dados orientado a objetos (O2) como os sistemas de banco de dados locais e o modelo de dados O2 como o modelo de dados global. Esta estratégia facilita o trabalho e diminui o tempo para se obter os primeiros resultados reais.

1.4 Organização

No Capítulo 2, apresenta-se a arquitetura de um sistema de múltiplos bancos de dados. Caracteriza-se API como uma interface única para acesso a vários sistemas de bancos de dados existentes e dá-se uma visão geral de como um sistema de múltiplos

bancos de dados processa uma consulta global e como ele obtém um esquema global.

No Capítulo 3 são revisados alguns conceitos importantes dos modelos de dados mais comumente utilizados. São revistos o Modelo Entidade-Relacionamento (modelo ER ou MER), o Modelo Entidade-Relaciomento Estendido (modelo ERE ou MERE), Modelo de Dados Relacional (modelo relacional ou MDR) e o Modelo de Dados Orientado a Objetos (modelo OO ou MDOO). Como o algoritmo de transformação do esquema relacional para o esquema orientado a objetos considera a existência do esquema ERE correspondente e o modelo ERE é o mais popular para projeto de banco de dados relacional, descrevem-se brevemente as regras que transformam o esquema ERE em esquema relacional.

O Capítulo 4 apresenta o algoritmo para transformar um esquema de banco de dados relacional para um esquema de banco de dados orientado a objetos, expondo algumas restrições, que definem os limites do transformador de esquema.

No Capítulo 5 apresenta-se uma implementação específica do transformador de esquemas. Expõe-se a estratégia do projeto de implementação e a especificação dos programas para transformar um esquema Sybase (relacional) em um esquema O2 (orientado a objetos). Também é apresentado o programa "script" para executar o ciclo completo do processo de transformação do esquema, começando com os procedimentos para se extrair a informação do dicionário de dados Sybase e terminando com o esquema O2 transformado. Além disso é apresentado um arquivo leia-me.

O Capítulo 6 apresenta as conclusões e as futuras pesquisas relacionadas com este trabalho.

CAPÍTULO 2

Arquitetura de Sistemas de Múltiplos Bancos de Dados

Nesse capítulo discutem-se as principais características da arquitetura de sistemas de múltiplos bancos de dados (SMBD). Primeiro, na Seção 2.1, estende-se a discussão sobre os SMBDs iniciada no Capítulo 1 e caracteriza-se uma API de banco de dados como uma interface única para acesso a vários sistemas de bancos de dados (SGBD) locais existentes. Cada SGBD tem sua própria linguagem de consulta (query) e seu próprio esquema local que descreve como os dados estão organizados nesse SGBD. Na Seção 2.2, apresentam-se os passos necessários para processar um comando de consulta global (global query) e os passos para obter o esquema global num SMBD. Nesse contexto, decomposição de comandos de consulta (query decomposition), tradução de comandos de consulta (query translation), transformação de esquemas (schema transformation) e integração de esquemas (schema integration) são passos muito importantes.

2.1 Sistemas de Múltiplos Bancos de Dados

Um dos desafios na pesquisa em banco de dados é possibilitar que usuários de um SGBD tenham acesso a dados em diferentes SGBDs, com facilidade.

2.1.1 API de Banco de Dados

Uma API de banco de dados é uma interface provendo um conjunto de serviços específicos de acesso a um SMBD. Os serviços aqui discutidos são aqueles utilizados por programas de aplicação e por programas utilitários de administração de banco de dados, isto é, serviços que possam ser ativados por qualquer linguagem de consulta a banco de dados (query language). Cada serviço está associado a uma primitiva de acesso. De uma forma simplificada, como mostrada na Figura 2.1, os SGBDs locais provêm seus serviços através da API e os programas de aplicação e os programas utilitários de administração de banco de dados, por sua vez, fazem chamada a esses serviços através das primitivas de acesso.

A definição das características de uma API deve considerar as tendências de mercado na utilização de SGBDs, iniciativas de padronização de SGBDs e tecnologias emergentes na área de banco de dados.

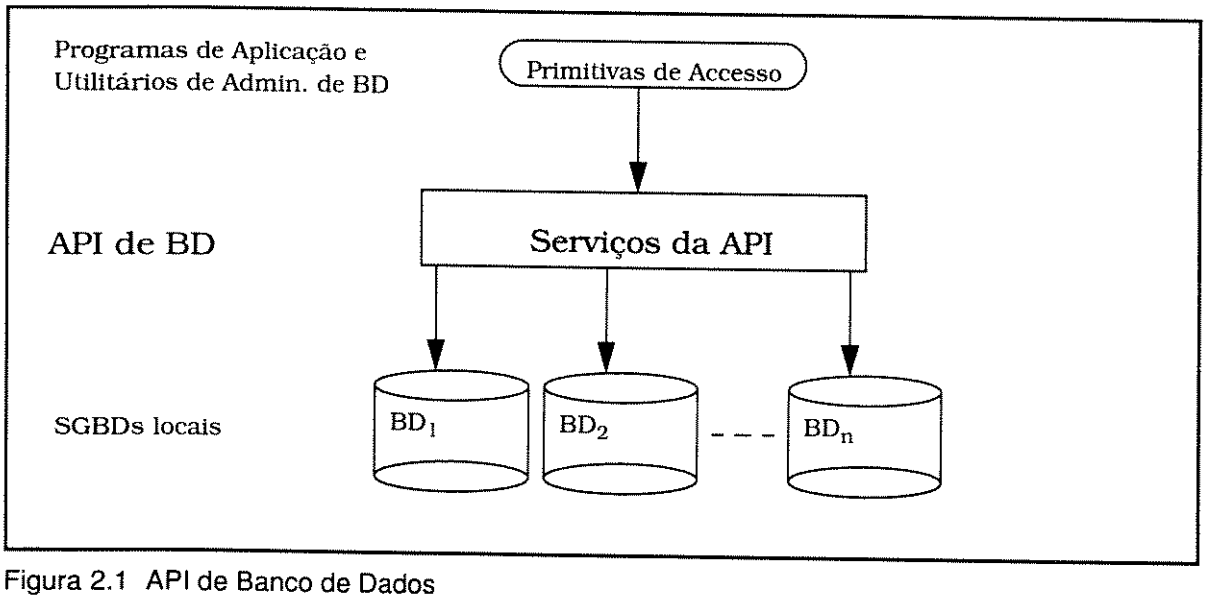

Figura 2.1 API de Banco de Dados

2.1.2 Generalidade de uma API

A generalidade de uma API pode variar, no sentido de que ela pode ser específica para um modelo de dados (SMBD Homogêneos), "genérica" para alguns modelos

de dados e genérica para "todos" os modelos de dados (SMBD Heterogêneos). Pode-se ter, por exemplo, uma API SQL específica para o modelo relacional e uma API SQL genérica para os modelos hierárquico, em redes, relacional e orientado a objetos.

2.1.3 Abordagem 1: Esquema e Linguagem Globais

Uma solução, vislumbrada por algumas pesquisas [22,30,44], é construir uma API de banco de dados, baseado num único modelo de dados (com um esquema global) e uma única linguagem de consulta global, como uma interface única para acesso a diferentes SGBDs locais existentes. Esses SGBDs locais podem suportar diferentes modelos de dados. Conforme dito anteriormente, esse conjunto, incluindo a API e os SGBDs locais é chamado de um Sistema de Múltiplos Bancos de Dados (SMBD). Um SMBD pode ser homogêneo ou heterogêneo significando, respectivamente, que os SGBDs locais, todos suportam o mesmo modelo de dados (relacional, por exemplo) ou alguns deles suportam diferentes modelos de dados (relacional, hierárquico e orientado a objetos, por exemplo). Um SMBD é também conhecido na literatura em inglês, como "federated database system". Assim, as técnicas de construção de sistemas de múltiplos bancos de dados heterogêneos (SMBD Heterogêneos) [7, 18, 30,33,34,42] devem ser usadas para permitir que usuários de um SGBD tenham acesso a dados de outros SGBDs, diferentes em termos de modelo de dados.

Conforme visto no Capítulo 1, quando um programa de aplicação usa um comando de consulta (tal como select do SQL) para recuperar informações, a API decompõe e traduz esse comando em comandos de consultas para os específicos SGBDs locais onde aquelas informações estão armazenadas. Essas tarefas de decomposição e tradução de comandos de consulta podem ser levadas avante em tempo de execução, dinamicamente, mas para isso, antes, os esquemas dos SGBDs locais precisam ser transformados e integrados num esquema global, tratado no nível da API de banco de dados. Nesse contexto, decomposição de comandos de consulta [5, 10], tradução de comandos de consulta [25, 44], transformação de esquemas [19, 22, 31, 43, 44, 45], integração de esquemas [4, 21] e também, otimização de comando global [5, 9, 28] são componentes críticos e de fundamental importância em SMBD Heterogêneos. Todos esses componentes, na realidade, devem trabalhar de forma cooperativa para processar os comandos de consulta nos SMBD Heterogêneos.

2.1.4 Abordagem 2: Criação de Visão Temporária

Uma outra abordagem para extrair dados de um SMBD, diferente da abordagem do esquema global, é criar uma visão temporária pertinente ao comando de consulta do usuário [26, 27]. Nessa abordagem, a linguagem de consulta do SMBD deve prover os meios para a definição da visão temporária bem como prover as funções para recuperar dados em vários SGBDs. É responsabilidade do usuário criar e manter as visões.

2.2 Processamento de Comando de Consulta Global

Nosso enfoque será sobre a abordagem que usa o esquema global e a linguagem de consulta global, por a considerarmos bastante viável em termos de processamento de comandos de consulta em SMBD Heterogêneos. Em SMBD Heterogêneos há vários SGBDs locais diferentes. O usuários interagem com a API de banco de dados através da linguagem de consulta global para especificar comandos de consultas baseadas no esquema global. Quando um comando de consulta é submetido, ele é processado em três passos. Primeiro, ele é decomposto em subcomandos (subqueries), de tal forma que os dados necessários para cada subcomando estejam disponíveis em um único SGBD local. Segundo, cada subcomando é traduzido para um comando (ou vários comandos, se necessário) atuando sobre o esquema local e enviado(s) para o correspondente SGBD para execução. Terceiro, os resultados retornados pelos comandos atuando sobre os SGBDs locais são combinados para formar a resposta. A Figura 2.2.a ilustra os passos do processamento de comandos de consulta global.

Conceitualmente um comando de consulta global é decomposto em dois tipos de sub-comandos: (1) sub-comandos de consulta global, ou simplesmente sub-comandos e (2) comandos de pós-processamento. Como visto anteriormente, os sub-comandos, embora ainda no esquema global, estão direcionados para atuar sobre um único SGBD local. Os comandos de pós-processamento combinam os resultados retornados pelos sub-comandos para formar a resposta. Eles podem não ser necessários. Um comando de pós-processamento pode ser processado pela API ou por algum SGBD local designado pela API.

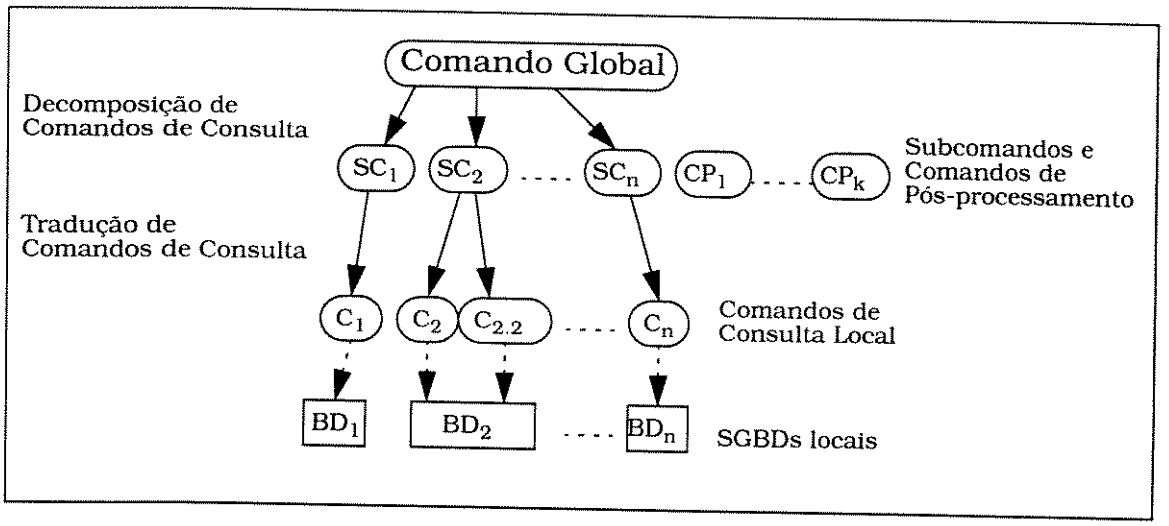

Figura 2.2.a Processamento de comando de consulta global

2.2.1 Decomposição de Comando de Consulta Global

O processo de decomposição de comandos de consulta global normalmente é realizado em dois passos:

1) O comando de consulta global usando nomes globais é modificado para comandos usando somente nomes contidos nos esquemas dos SGBDs locais [5, 10]. Os comandos de consulta modificados, embora utilizando somente nomes dos esquemas locais, podem ainda estar referenciando dados de mais que um SGBD.
2) Esses comandos modificados são decompostos. Note que, eventualmente, dados devam ser transmitidos de um SGBD para outro, para facilitar a decomposição. Por exemplo, considere a situação onde o comando de consulta resultante da modificação seja aninhado. Tal comando pode ser decomposto em dois sub-comandos: um para a consulta no primeiro nível e outro para a consulta no segundo nível de profundidade. O resultado do segundo subcomando (segundo nível) pode ser necessário ser transmitido para o SGBD onde o primeiro subcomando (primeiro nível) será executado, com todos os dados disponíveis.

Uma vez que as questões de como os dados devam ser transmitidos e que processamento local deveria ser executado antes da transmissão são relacionados com o

desempenho geral do sistema, esse segundo passo na decomposição de comandos de consulta global é de maneira geral, fortemente relacionado com os aspectos de otimização de comandos de consulta global [5].

2.2.2 Tradução de Comando de Consulta Global

Após a decomposição, cada subcomando necessita dados de um único SGBD e todos os sub-comandos estão ainda expressos na linguagem de consulta global. Se a linguagem de consulta global é diferente da linguagem de consulta de um SGBD local, o correspondente subcomando deve ser traduzido para o comando de consulta local. Essa tradução é feita pelo tradutor de comandos de consulta. Por exemplo, se a linguagem de consulta global é uma linguagem orientada a objetos e um subcomando é para um SGBD relacional usando SQL, então o subcomando precisa ser traduzido para um comando SQL. Assim, para traduzir diferentes subcomandos para diferentes SGBDs locais são necessários diferentes tradutores.

Um dado subcomando pode ser traduzido para diferentes conjuntos de comandos para os SGBDs locais e cada um desses conjuntos de comandos podem ser avaliados pelos SGBDs locais com diferentes custos. Portanto, cada tradutor de comandos de consulta precisa considerar não somente a correção da tradução, garantindo a equivalência com o subcomando original, mas também o desempenho dos comandos traduzidos.

Algumas vezes não é possível traduzir um comando de uma linguagem para um equivalente conjunto de comandos em outra (linguagem), porque algumas operações na primeira linguagem podem não ser suportadas na segunda. Por exemplo, um comando relacional com um predicado "join" pode não ter um equivalente comando hierárquico. Nesse caso, o comando relacional é traduzido em comandos que, juntos retornem um superconjunto do resultado desejado, de tal forma que a restrição ("join", no exemplo) seja tratada a nível de pós-processamento de comandos.

2.3 Obtenção de Esquema Global

Como visto anteriormente, a linguagem de consulta global pressupõe um esquema global, que será usada pela API. O esquema global é obtido de forma inversa em que os comandos de consulta são processados. Por exemplo, numa API orientada a objetos, para apresentar aos usuários uma visão orientada a objetos, os esquemas de cada SGBD local precisam ser transformados em equivalentes esquemas orientados a objetos, na verdade, subesquemas. Esses subesquemas, já no modelo de dados global, são integrados no esquema global único. A Figura 2.2.b ilustra os passos para obtenção do esquema global.

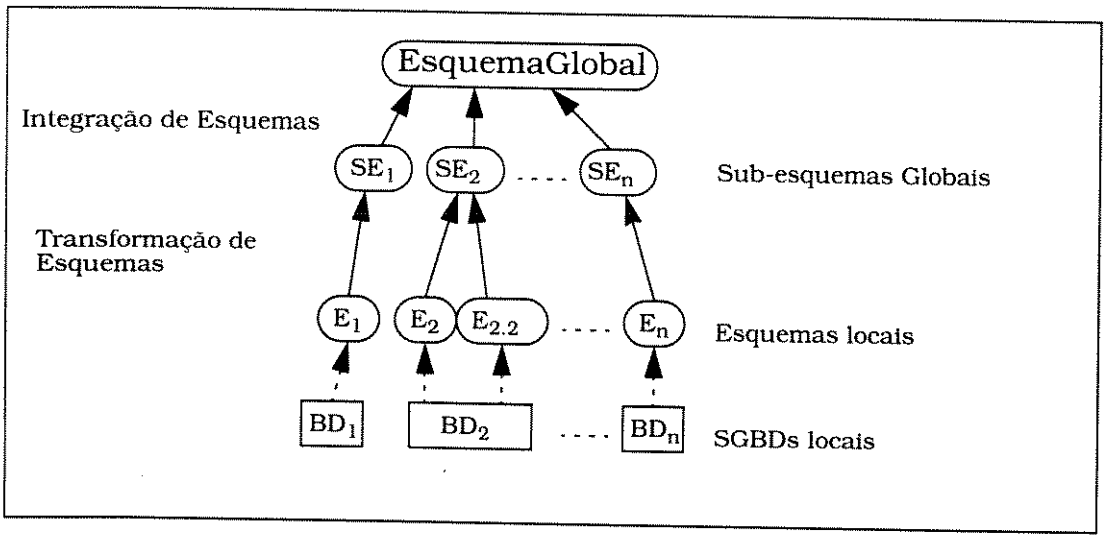

Figura 2.2.b Obtenção de Esquema Global

Deve-se esclarecer que o esquema global é um esquema virtual, sem instâncias. Os dados permanecem completamente nos SGBDs locais. Tal processo de transformação de esquemas é discutido em [22, 31, 43, 44]. Exceto pelo algoritmo apresentado por Meng, Kamada e Chang [31], que é parte desse trabalho, todos os outros apresentam limitações. O algoritmo apresentado por Keim, Kriegel e Miethsam [22] não considera a construção de hierarquia de classes no esquema orientado a objetos resultante, enquanto o algoritmo de Yan e Ling [43] não considera a criação de conjuntos de atributos (set) para simplificar o esquema orientado a objetos resultante. Muito embora o algoritmo de

Yu e outros [44] possa gerar esquemas tipicamente orientados a objetos, contendo hierarquia de classes, hierarquia de composições e de conjuntos atributos, ele gera um número desnecessariamente grande de classes. Além do mais, nenhum desses algoritmos foi implementado. Em Meng, Kamada e Chang [31] é apresentado um algoritmo de transformação de esquemas detalhado para transformar esquemas relacionais para esquemas orientados a objetos. Nesse trabalho, o algoritmo foi implementado para transformar esquemas de um SGBD relacional comercial (Sybase) para esquemas de um SGBD orientado a objetos, também comercial (O2). O transformador de esquemas implementado inicia uma sessão interativa com o usuário para solicitar mais informações quando não há informação no esquema relacional original, suficiente para que o transformador de esquemas tome as decisões de transformação. O número de classes no esquema O2 transformado é relativamente pequeno porque o algoritmo se beneficia da utilização de construtores de objetos complexos, tais como hierarquia de classes, hierarquia de composições, hierarquia de tuplas e conjuntos de atributos.

2.4 Resumo do Capítulo 2

Nesse capítulo, foram discutidas as principais características de sistemas de múltiplos bancos de dados e caracterizou-se uma API de banco de dados como uma interface única para diferentes SGBDs. Apresentou-se uma visão geral de como sistemas de múltiplos bancos de dados processam comandos de consulta global e como obtêm o esquema global, mostrando a importância dos componentes de transformação de esquemas, integração de esquemas, tradução de comandos de consulta e decomposição de comandos de consulta na construção de tais sistemas. No próximo capítulo, apresentam-se algumas das mais importantes características de quatro modelos de dados.

CAPÍTULO 3 *Modelos de Dados*

Um **modelo de dados** é uma norma que pode ser usada para representar dados do mundo real em forma abstrata. Ele é composto por um conjunto de entidades ou objetos, um conjunto de operadores e um conjunto de regras de integridade. Conceitualmente, um modelo de dados é a combinação de entidades de um domínio, com operadores, obedecendo as regras de integridade do modelo. Todas as técnicas de modelagem de dados tentam criar uma representação do domínio do problema do mundo real para o domínio da solução, que é uma abstração do domínio do problema. Nesse capítulo, são revistos alguns conceitos de quatro modelos de dados: Modelo Entidade-Relacionamento (modelo ER ou MER), Modelo Entidade-Relacionamento Estendido (modelo ERE ou MERE), Modelo de Dados Relacional (modelo relacional ou MDR) e Modelo de Dados Orientado a Objetos (modelo OO ou MDOO). Como o modelo ERE é um dos mais populares em projetos de bancos de dados relacionais, resumidamente apresenta-se um algoritmo para transformar esquemas ERE para esquemas relacionais (ver [41] para mais detalhes). É importante entender esse passo porque o algoritmo de transformação de esquema relacional em esquema OO, apresentado no Capítulo 4, considera o esquema relacional associado com o correspondente esquema ERE. Os conceitos estendidos do modelo ERE possibilitam

transformar um esquema ERE diretamente para um esquema OO, uma vez que esses esquemas apresentam muitas similaridades conceituais. O restante desse capítulo está estruturado da seguinte forma. Na Seção 3.1, apresentam-se alguns conceitos do MER, tais como entidades, atributos, domínios, chaves e restrições de integridade. Na Seção 3.2, são revistos os conceitos de supertipo e subtipo e os conceitos relacionados de especialização, generalização e herança de atributos, que estão definidos no contexto do MERE e que não estão presentes no MER. Na Seção 3.3, apresentam-se alguns conceitos do MDR, tais como relações, atributos, chaves primárias, chaves estrangeiras e restrições de integridade. Nessa seção, apresenta-se também um algoritmo para transformar esquemas ERE para esquemas relacionais. Finalmente, na Seção 3.4 apresentam-se alguns conceitos do MDOO, tais como objetos, classes, encapsulamento, herança, etc.

3.1 Modelo Entidade-Relacionamento

O modelo Entidade-Relacionamento (ER), introduzido por Chen [6] em 1976, é um modelo de dados conceitual de alto nível. Os conceitos desse modelo e suas variações são largamente usados para projeto conceitual de aplicações de banco de dados. Muitas ferramentas CASE (Computer-Aided Software Engineering) empregam esses conceitos e permitem que projetistas de bancos de dados desenvolvam esquemas ER interativamente. Uma ferramenta CASE gera automaticamente, a partir de um esquema ERE, os esquemas relacionais para SGBDs específicos.

O modelo ER descreve dados como entidades, relacionamentos e atributos. Nessa seção consideram-se somente os conceitos mais relevantes para o objetivo do projeto. Mais detalhes poderão ser encontrados em [6, 13]. Para facilitar a exposição, apresenta-se na Figura 3.1.b o esquema ER reconstituído da base de dados relacional pubs2, com algumas modificações, retirado do exemplo contido no Sybase SQL Server Transact-SQL User's Guide [37]. A convenção utilizada é mostrada na Figura 3.1.a.

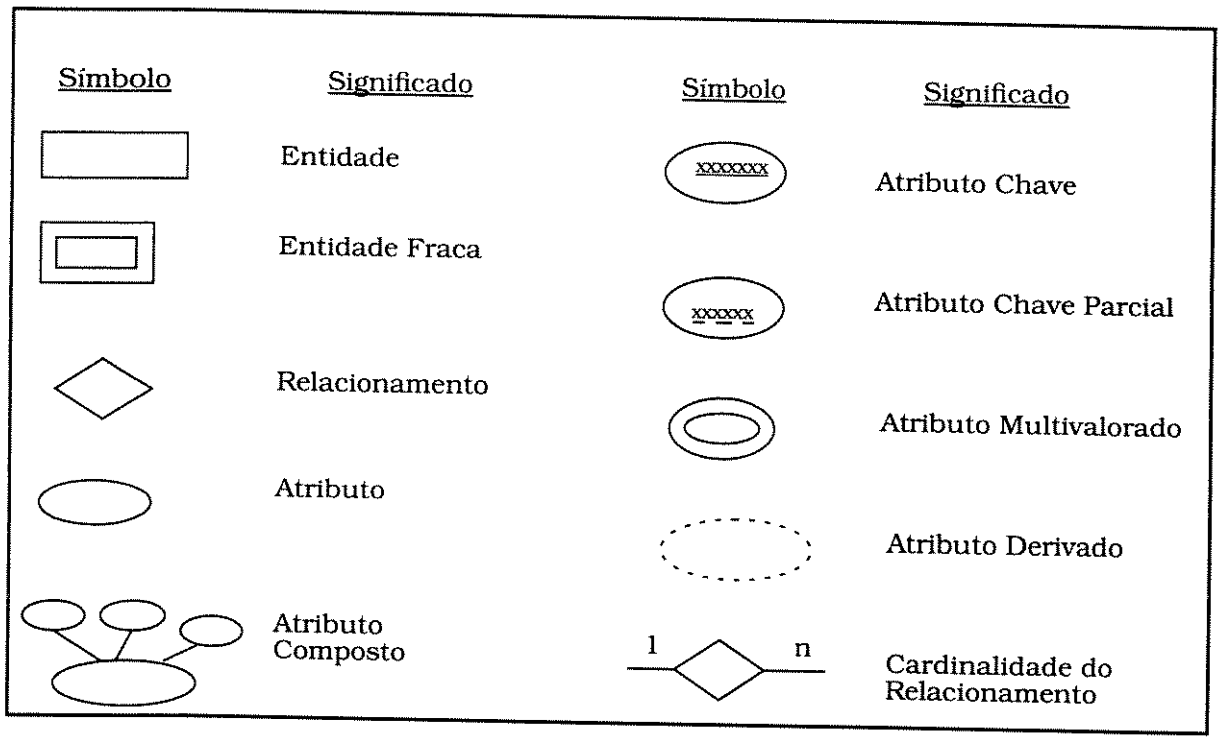

Figura 3.1.a Notação para esquemas ER

3.1.1 Entidades

Todo objeto do mundo real é representado como uma entidade no modelo ER. Uma entidade pode ser um objeto com uma existência física, tal como uma pessoa específica, um carro, um edifício ou um funcionário, ou ele pode ser um objeto com uma existência conceitual, tal como uma organização, um projeto ou uma profissão. Um conjunto de entidades do mesmo tipo, tal como o conjunto de autores de uma editora é chamada de um tipo de entidade ou conjunto de entidades (entity set). Uma entidade tem duas propriedades: ela existe e ela é distinguível. Além disso uma entidade pode ser classificada em **entidade forte (strong entity)**, que pode existir independentemente ou **entidade fraca (weak entity)**, cuja existência depende de outra entidade.

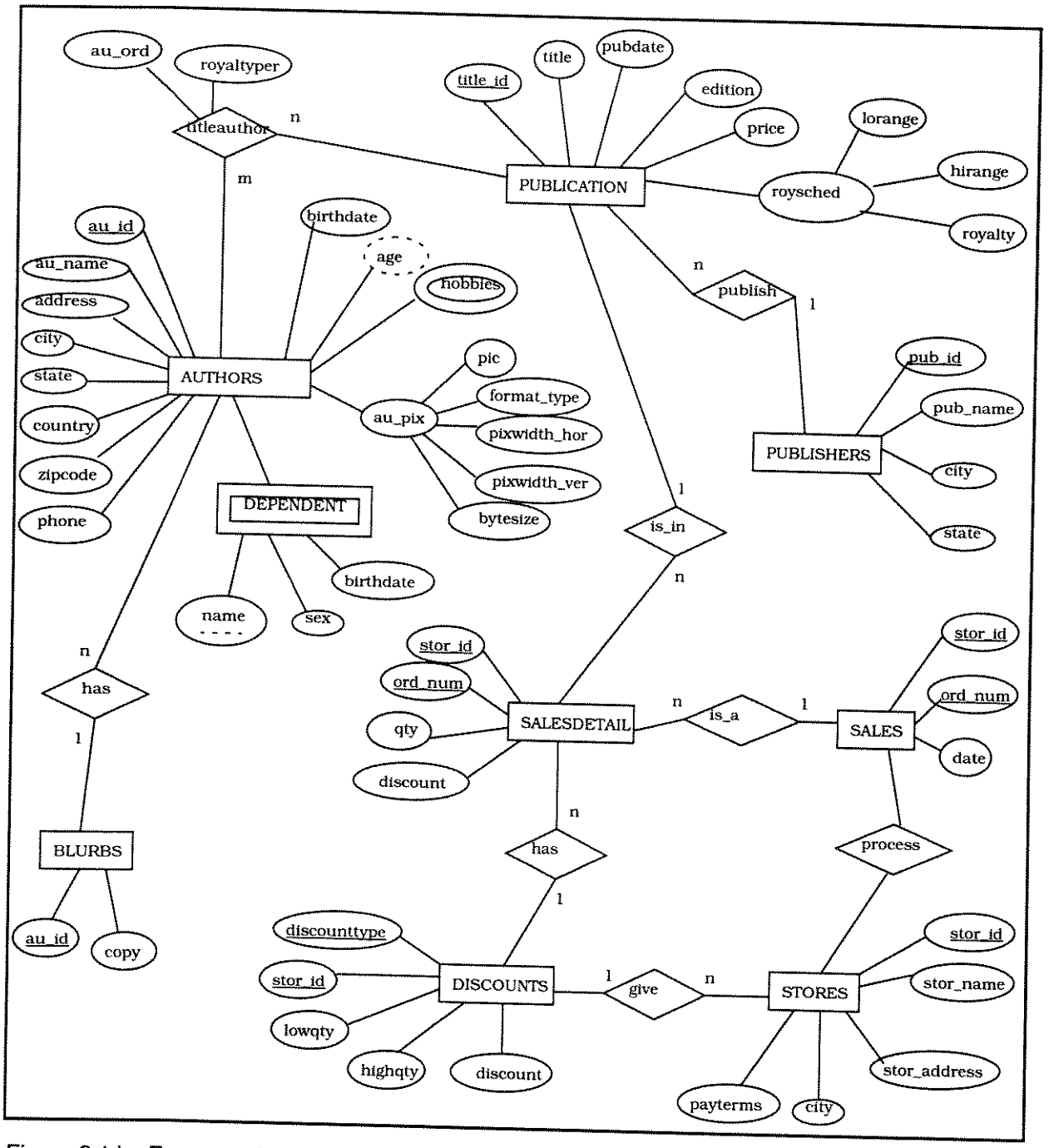

Figura 3.1.b Esquema ER para o banco de dados pubs2 modificado

Cada entidade fraca tem uma única entidade forte correspondente. Na realidade, a existência e a própria identificação de uma entidade fraca depende da existência da sua correspondente entidade forte.

3.1.2 Tipos de Atributos e Domínios

Propriedades que descrevem uma entidade são chamadas de **atributos**. Por exemplo, a entidade AUTHORS pode ser descrita pelos atributos nome do autor, telefone, endereço, etc. Cada entidade particular tem um valor para cada um de seus atributos. Um atributo pode ser simples ou composto (simple or composite), univalorado ou multivalorado (single-valued or multi-valued) e armazenado ou derivado (stored or derived).

Quando um atributo assume um valor único e indivisível para cada entidade ele é chamado de **atributo simples** (simple) e quando ele assume um valor único, mas o valor pode ser dividido em valores significativos menores, ele é chamado de **atributo composto** (composite). Na entidade AUTHORS, *phone* e *city* são atributos simples e *au_pix* um atributo composto. Um atributo é chamado **univalorado** (single valued) se ele assume um valor único para uma entidade e **multivalorado** (multivalued), caso contrário. Assim, na Figura 3.1.b, *au_name*, *phone* e *city* são atributos univalorados e *hobbies* é um atributo multivalorado, também chamado atributo conjunto (set attribute), porque cada autor pode ter vários hobbies. Um atributo é do tipo **armazenado** (stored) quando seu valor está realmente armazenado no banco de dados e ele é do tipo **derivado** (derived) quando seu valor pode ser computado a partir de outros valores. Todos os atributos de AUTHORS são do tipo armazenado, exceto o atributo *age* que pode ser do tipo derivado (seu valor pode ser computado a partir da data de nascimento e da data corrente do sistema).

Domínio de um atributo especifica que tipo e escopo de valores o atributo pode assumir. Por exemplo, o domínio do atributo *au_name* poderia ser "char(25)" e o domínio do atributo *age* poderia ser "18, 19, ...,70".

3.1.3 Chaves

Como visto anteriormente, um conjunto de entidades é uma coleção de entidades do mesmo tipo. Uma restrição importante sobre entidades de um conjunto é a restrição da chave. Um conjunto de entidades tem usualmente um atributo cujos valores são distintos para cada entidade individual. Tal atributo é chamado de **atributo chave** (attribute key) e seus valores podem ser usados para identificar cada entidade de forma única. Como pode ser visto no diagrama ER de pubs2 (Figura 3.1.b), cada atributo chave tem seu nome sublinhado dentro da elipse. Na entidade PUBLICATION, *title_id* é um atributo chave porque não é admitido nenhum par de publicações com a mesma identificação de título. Da mesma forma, *au_id* é um atributo chave identificando entidades do conjunto AUTHORS.

Algumas vezes, vários atributos juntos formam uma chave, no sentido de que a combinação dos valores dos atributos deve ser distinta para cada entidade individual. Um conjunto de atributos que possuem essa propriedade podem ser agrupados num atributo composto, que se torna um **atributo chave composto**, do conjunto de entidades. No diagrama ER de pubs2, a entidade SALES tem um atributo chave composto, formado pelos atributos *stor_id* e *stor_num*.

O conceito de atributo chave é válido para todas as entidades no conjunto de entidades. Uma entidade forte tem pelo menos um atributo chave e uma entidade fraca não tem atributo chave. Cada entidade fraca tem a chamada **chave parcial**. Para uma dada tupla na correspondente entidade forte suas tuplas dependentes podem ser identificadas pelos valores contidos na chave parcial. Na entidade fraca DEPENDENT o atributo *name* é uma chave parcial (sublinhada com uma linha pontilhada).

3.1.4 Relacionamentos

Um **relacionamento** (relationship) é uma associação entre entidades. No banco de dados pubs2, *has*, *publish*, *is_in* e *process* são exemplos de relacionamentos. Todos os relacionamentos com o mesmo significado entre conjuntos de entidades são coletivamente chamados de **conjunto de relacionamentos (relationship set)**. Formalmente, um conjunto

de relacionamentos R entre n conjuntos de entidades E_1, E_2, ... , E_n define um conjunto de associações entre entidades desses conjuntos. R é um conjunto de instâncias de relacionamentos r_i, onde cada r_i associa n entidades (e_1, e_1, ..., e_n) e cada entidade e_j em r_i é um membro do conjunto E_j, $0 < j < n$. Em outras palavras, cada instância de relacionamento r_i de R é uma associação de entidades, onde a associação inclui exatamente uma entidade de cada conjunto de entidades participante. O fato de um conjunto de entidades ser participante num relacionamento não necessariamente significa que todas as entidades do conjunto estão envolvidas no relacionamento. Como nosso exemplo pubs2 não pode mostrar claramente esse aspecto, tomaremos um outro exemplo. Na Figura 3.1.4.a, o relacionamento *ta_of* (teacher assistant of) não significa que todos os *estudantes* são professores assistentes do *curso*.

O **grau de um relacionamento** é o número de conjuntos de entidades participantes no relacionamento. Quando somente um conjunto de entidades é envolvido no relacionamento o grau é 1 (um), chamado de **relacionamento unário (unary relationship) ou auto-relacionamento**. Na Figura 3.1.4.a, o relacionamento *prerequisite* é um relacionamento unário sobre a entidade COURSE. Um **relacionamento binário** ocorre quando o grau é 2, isto é, há dois conjuntos de entidades participantes no relacionamento. *Take* e *ta_of* são relacionamentos binários sobre os conjuntos de entidades STUDENT e COURSE.

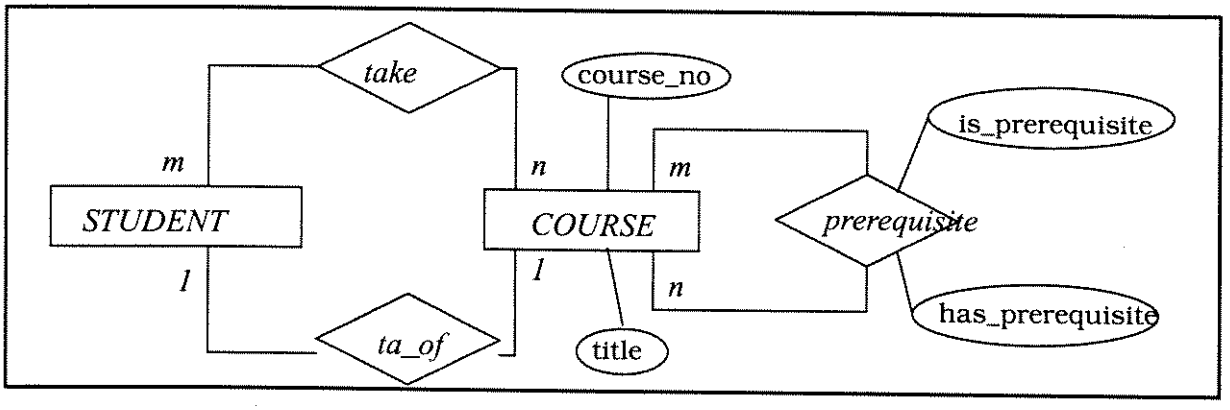

Figura 3.1.4.a Exemplo de Relacionamento unário e relacionamento binário

Relacionamento um-para-um, denotado por "(1:1)", significa que para cada entidade em um conjunto de entidades E_1, há no máximo uma entidade associada em outro conjunto de entidades E_2. E_1 e E_2 poderiam ser o mesmo conjunto de entidades. Na Figura 3.1.4.b, o relacionamento *married_to* é um **relacionamento unário (1:1)**, se considerarmos que uma pessoa se casa com somente uma outra pessoa. Na Figura 3.1.4.c, o relacionamento *manager_of* é um **relacionamento binário (1:1)** porque somente uma pessoa pode ser gerente de um departamento.

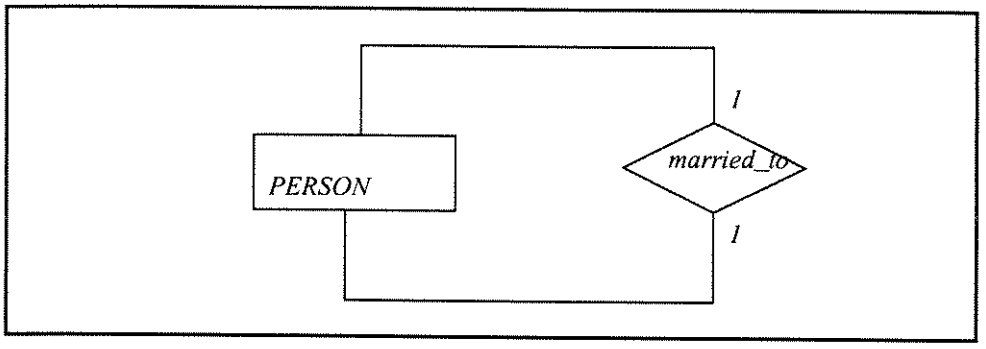

Figura 3.1.4.b Exemplo de Relacionamento Unário 1:1.

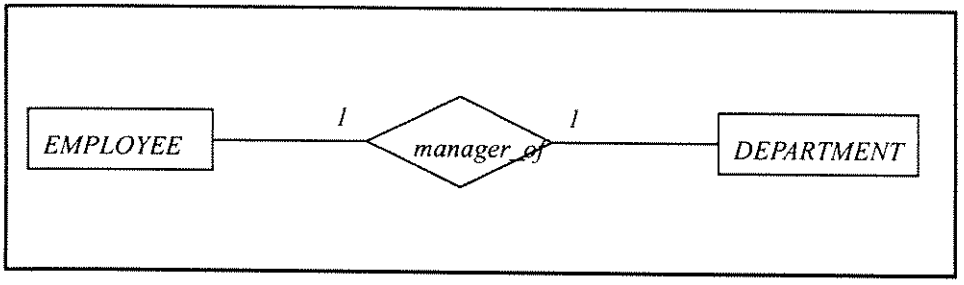

Figura 3.1.4.c Exemplo de Relacionamento Binário 1:1 - participação parcial e total.

Relacionamento um-para-muitos, denotado por "(1:m)", significa que entre o conjunto de entidades E_1 e o conjunto de entidades E_2, para cada entidade de E_1 há zero ou mais entidades associadas de E_2, e para cada entidade de E_2 há no máximo uma entidade

associada de E_1. E_1 e E_2 poderiam ser o mesmo conjunto de entidades. Na Figura 3.1.4.d, o relacionamento *mother_of* é um **relacionamento unário (1:m)**, porque uma pessoa pode ser mãe de várias pessoas. Esse relacionamento é também conhecido como **relacionamento recursivo** porque uma entidade PERSON pode desempenhar dois papéis: mãe e filha. Na Figura 3.1.4.e, o relacionamento *advisor_of* é um **relacionamento binário (1:m)** porque um professor pode ser orientador de muitos alunos.

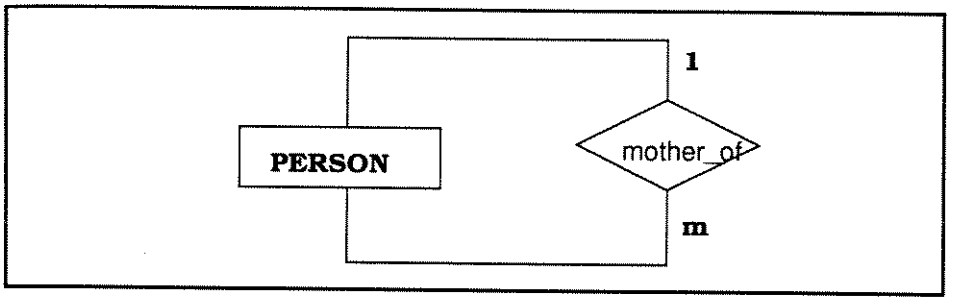

Figura 3.1.4.d Exemplo de Relacionamento Unário 1:m

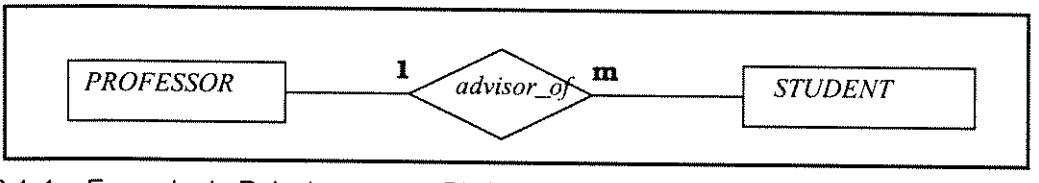

Figura 3.1.4.e Exemplo de Relacionamento Binário 1:m

Relacionamento muitos-para-muitos, denotado por "(m:n)", significa que para cada entidade de um conjunto de entidades E_1 há zero ou mais entidades associadas de outro conjunto de entidades E_2. E_1 e E_2 poderiam ser o mesmo conjunto de entidades. Na Figura 3.1.4.f, o relacionamento *brother_of* é um relacionamento unário (m:n), porque cada pessoa pode ser irmão de várias outras pessoas. O relacionamento *take* na Figura 3.1.4.g é um relacionamento binário (m:n), porque cada aluno pode cursar vários cursos e cada curso pode ser cursado por vários alunos.

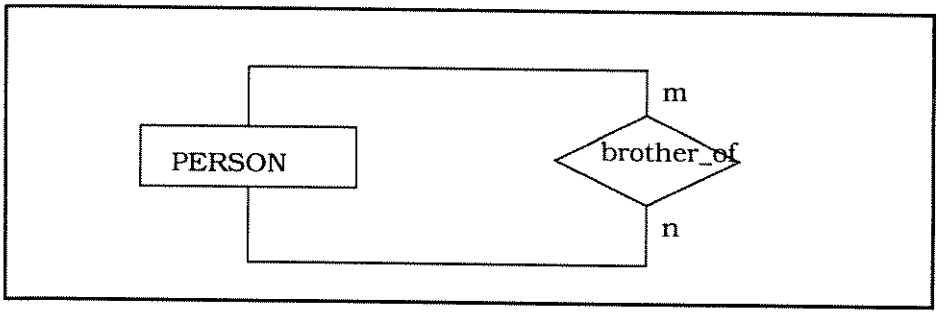

Figura 3.1.4.f Exemplo de Relacionamento Unário (m:n)

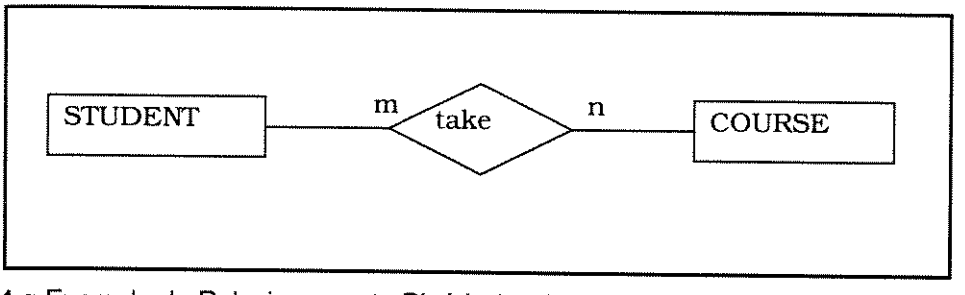

Figura 3.1.4.g Exemplo de Relacionamento Binário (m:n).

3.1.5 Restrições Estruturais

Conjuntos de relacionamentos normalmente têm certas restrições que limitam as combinações possíveis entre entidades que podem participar numa instância de relacionamento. Por exemplo, na Figura 3.1.4.e, se uma universidade tem uma regra estabelecendo que cada aluno deve ser orientado por exatamente um professor, essa restrição precisa ser descrita no esquema. Chamamos de **cardinalidade** do relacionamento a restrição que especifica o número de instâncias de relacionamento em que uma entidade pode participar. O relacionamento binário *advisor_of* entre PROFESSOR e STUDENT é de cardinalidade (1:m), significando que cada professor pode estar relacionado com vários alunos, mas cada aluno pode estar relacionado com somente um professor. As cardinalidades de relacionamento mais comuns são 1:1 (Figuras 3.1.4.b e 3.1.4.c), 1:m (Figuras 3.1.4.d e 3.1.4.e) e m:n (Figuras 3.1.4.f e 3.1.4.g).

Uma outra restrição é a chamada **restrição de participação**, que especifica que a existência de uma entidade depende de como ela está relacionada com outra entidade. A restrição de participação pode ser total ou parcial. Se uma política da universidade determina que todo aluno deve ser orientado por um professor, então uma entidade aluno pode existir somente se ele participa em uma instância de relacionamento chamada *advisor_of* (Figura 3.1.4.e). A participação de STUDENT na instância de relacionamento *advisor_of* é chamada de total, significando que toda entidade no conjunto total de entidades alunos deve ser relacionada com uma entidade professor via relacionamento *advisor_of*. A restrição de participação total é algumas vezes chamada de **dependência de existência**. Na Figura 3.1.4.c, nem todo funcionário é gerente de um departamento. Assim, a participação de EMPLOYEE no relacionamento *manager_of* é parcial, significando que algumas ou parte do conjunto de entidades funcionário são relacionados com uma entidade DEPARTMENT via relacionamento *manager_of*, mas não necessariamente todas.

3.2 Modelo Entidade-Relacionamento Estendido

Os conceitos do modelo ER apresentados na Seção 3.1 são suficientes para representar a maioria dos esquemas para aplicações tradicionais de bancos de dados, em sistemas administrativos. Entretanto, o aumento da demanda por novas aplicações utilizando dados muito mais complexos exige, como consequência, uma tecnologia de banco de dados muito mais complexa. Essas novas aplicações incluem bancos de dados de imagens, de gráficos, cartográficos, multimídia, de projetos de engenharia (CAD/CAM) e de conhecimentos para aplicações de inteligência artificial. Para representar esses requisitos, devem-se usar os chamados conceitos de modelagem semântica. Muitos desses conceitos estão incorporados ao modelo ERE [12, 35, 41]. O modelo ERE é uma extensão do modelo ER [6] e incorpora conceitos, tais como os de supertipo e subtipo e os conceitos relacionados de especialização e generalização. Associados a esses conceitos há um outro, que é o de herança de atributos. Para ilustrar esses conceitos, uma versão estendida do esquema depubs2 na notação ERE é apresentada na Figura 3.2.a. Setas são usadas para representar relacionamentos subtipo/supertipo.

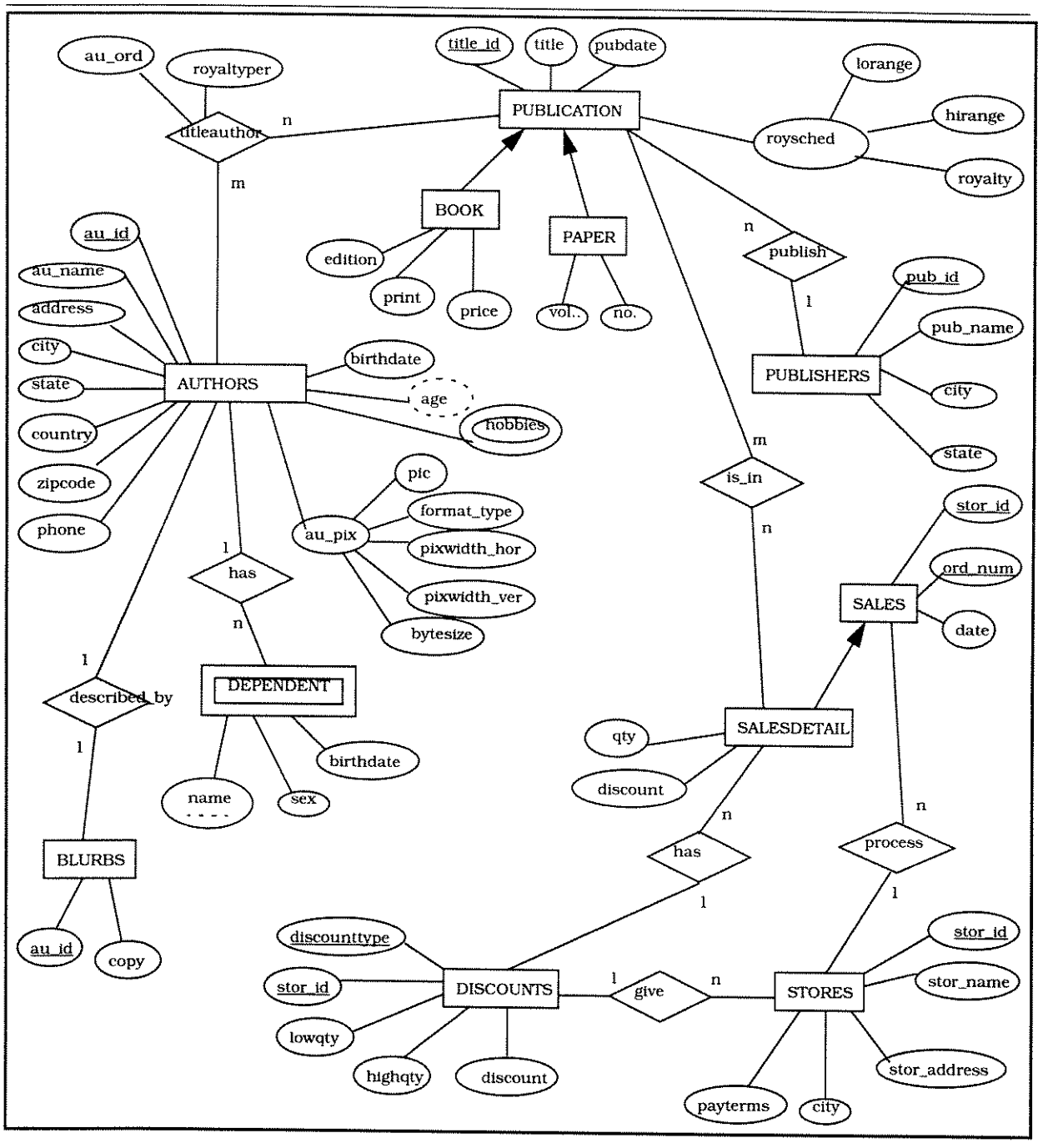

Figura 3.2.a Diagrama ERE do banco de dados pubs2.

3.2.1 Relacionamento Subtipo/Supertipo

A inclusão do conceito de subtipo e supertipo é, provavelmente a extensão mais importante para capturar o chamado **relacionamento is_a** entre entidades. Como visto na seção anterior, um tipo de entidade é um conjunto de entidades do mesmo tipo, tal como o conjunto de entidades PUBLICATION num banco de dados de uma companhia de publicidade. Em muitos casos é interessante dividir um conjunto de entidades em alguns subconjuntos de entidades tais que cada um desses subconjuntos tenham um significado mais apropriado para a aplicação. Por exemplo, na Figura 3.2.a, as entidades PUBLICATION podem ser agrupadas em entidades BOOK, PAPER, etc.

Cada um desses conjuntos de entidades, BOOK e PAPER, é um subconjunto do conjunto de entidades PUBLICATION, significando que toda entidade que é um livro ou um artigo é também uma publicação. Cada um desses subgrupos, BOOK e PAPER, é chamado de subtipo do conjunto de entidades PUBLICATION, e PUBLICATION é chamado de supertipo de BOOK e PAPER. O relacionamento entre um supertipo e qualquer um de seus subtipos é chamado de **relacionamento supertipo/subtipo**. Um relacionamento supertipo/subtipo é freqüentemente chamado de relacionamento **is_a** ou **is_an**. Isto se deve ao fato de que podemos dizer "a book **is_a** publication" ou "a paper **is_a** publication". Assim, se um conjunto de entidades E_2 é um subtipo do conjunto de entidades E_1, então toda entidade de E_2 é também uma entidade de E_1.

3.2.2 Herança de Atributos

Herança de atributos é o mecanismo associado com o conceito de relacionamento supertipo/subtipo, que significa que uma entidade no subtipo representa a mesma entidade do mundo real do supertipo. Cada entidade no subtipo herda todos os atributos da entidade como um membro do supertipo. A entidade também herda todas as instâncias de relacionamento do conjunto de relacionamentos em que o supertipo participa. BOOK e PAPER herdam todos os atributos e relacionamentos de PUBLICATION.

3.2.3 Especialização e Generalização

Especialização é o processo de definição de subtipos de um conjunto de entidades. Subtipos são definidos a partir de alguns aspectos que distinguem as entidades no supertipo. Por exemplo, BOOK e PAPER são especializações do supertipo PUBLICATION que se distinguem entre as entidades PUBLICATION baseados no tipo de publicação. Assim, o processo de especialização permite definir um conjunto de subtipos de um conjunto de entidades e associar atributos específicos adicionais para cada um dos subtipos. Generalização é o processo inverso da especialização, isto é, ao invés de se adicionar, suprimem-se as diferenças entre vários conjuntos de entidades, identificando seus aspectos comuns, para generalizá-los num único supertipo. Por exemplo, supondo que o conjunto de entidades PUBLICATION não esteja ainda definido e têm-se os conjuntos de entidades BOOK e PAPER, como definido na Figura 3.2.b. BOOK e PAPER podem ser generalizados num único conjunto de entidades PUBLICATION, como mostrado na Figura 3.2.a, através da supressão das diferenças (isto é, supressão dos atributos *edition*, *print* e *price* de BOOK e *vol.* e *no.* de PAPER) e mantendo os atributos comuns (isto é, atributos *title_id*, *title* e *pubdate*). O processo de definição de um conjunto de entidades generalizado a partir de outros conjuntos de entidades é chamado de generalização.

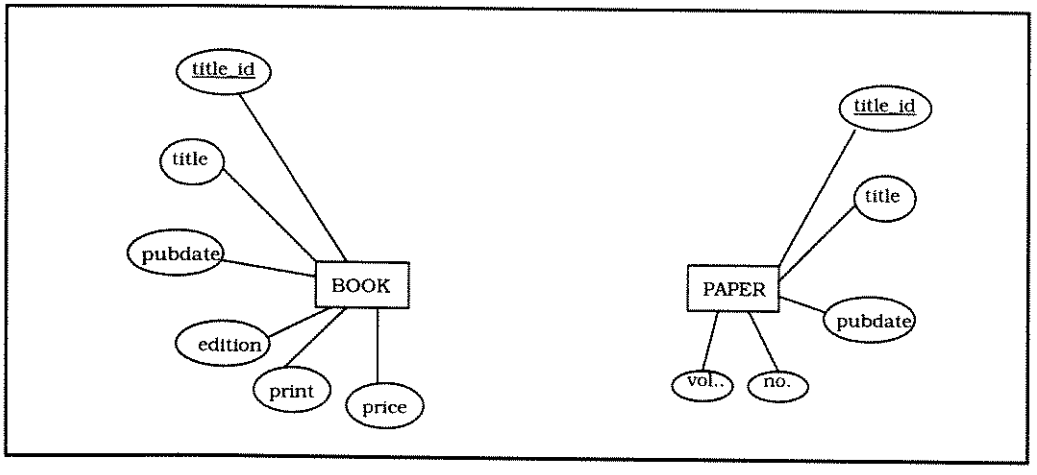

Figura 3.2.b Exemplo de generalização/especialização.

3.3 Modelo de Dados Relacional

O Modelo de Dados Relacional foi introduzido por Codd [8] em 1970. O modelo é baseado numa estrutura de dados simples e uniforme chamada relação e possui um sólido fundamento matemático. Todas as entidades e relacionamentos entre entidades do mundo real são uniformemente modeladas como relações num SGBD relacional. Nessa seção são revistos alguns conceitos, além do de relação, associados com o modelo de dados relacional, tais como os conceitos de atributos, domínios, tuplas, chaves e restrições de integridade.

3.3.1 Relações, Atributos, Tuplas e Domínios

Na realidade, uma relação pode ser vista, de uma forma simplificada, como uma tabela de valores. O nome da tabela equivale ao nome da relação, os nomes das colunas equivalem aos atributos da relação e o equivalente para as linhas de uma tabela são as tuplas de uma relação. Cada entidade ou cada relacionamento entre entidades é representado por uma tupla (um conjunto de valores de uma entidade específica) de uma relação. Na Tabela 3.1 é mostrada uma relação como um tabela composta por algumas tuplas para o conjunto de entidades PUBLICATION, do diagrama ER na Figura 3.1.b.

Tabela 3.1 PUBLICATION

title_id	title	pubdate	edition	price
0-1234-5678-9	Object-Oriented Database systems	07/07/95	third	49.00
9-8765-4321-0	My so called Life	10/05/95	second	55.00

PUBLICATION é o nome da relação (tabela), a primeira linha na tabela indica os nomes dos atributos (colunas) da relação. A segunda e a terceira linhas indicam as tuplas (conjunto de valores para os atributos de instâncias específicas da relação PUBLICATION) da relação. Domínio é um conjunto de valores associados com cada atributo de uma relação. Assim, o

domínio do atributo *title_id* poderia ser um conjunto de códigos ISBNs válidos, o domínio do atributo *pubdate* poderia ser um conjunto de datas válidas com 6 dígitos, etc. Um tipo de dado ou formato é também especificado para cada domínio. Por exemplo, o tipo de dado para o atributo *pubdate* pode ser declarado como uma cadeia de caracteres no formato dd/mm/aa, onde dd, mm e aa são números representando dias, meses e anos válidos respectivamente. É útil também especificar um nome para o domínio, para ajudar na interpretação de seus valores.

Formalmente, a estrutura de uma relação é descrita por um esquema relacional. Um esquema relacional R, denotado por $R(A_1:D_1, A_2:D_2, \ldots, A_n:D_n)$, constitui-se do nome da relação R e da lista de atributos $A_1, A_2, \ldots, A_n$. Cada atributo A_i está associado com algum domínio D_i, no esquema relacional R. O grau de uma relação é o número (n) de atributos de seu esquema relacional. No exemplo acima, o esquema relacional descrevendo publicação é como segue:

PUBLICATION(title_id:tid, title: varchar, pubdate: datetime, edition: varchar, price: int)

Para esse esquema, PUBLICATION é o nome da relação, que tem cinco atributos (o grau da relação é cinco).

Uma relação ou instância de relação r de um esquema relacional $R(A_1:D_1, A_2:D_2, \ldots, A_n:D_n)$, é um conjunto de n-uplas $r = \{t_1, t_2, \ldots, t_m\}$. Cada n-upla t_i é uma lista ordenada de n valores $t = \langle v_1, v_2, \ldots, v_n \rangle$, onde cada valor v_i é um elemento de D_i ou é um valor nulo especial.

A **relação** ou instância de relação r, representada na tabela 3.1, do esquema relacional PUBLICATION(title_id:tid, title: varchar, pubdate: datetime, edition: varchar, price: int) é o conjunto de **duas 5-uplas** r = {<0-1234-5678-9, Object-Oriented Database systems, 07/07/95, third, 49.00>, <9-8765-4321-0, My so called Life, 10/05/95, second, 55.00>}.

3.3.2 Esquemas de Bancos de Dados Relacionais

Um SGBD relacional normalmente contém muitas relações, com tuplas nas relações que estão associadas de várias formas. Apresentamos aqui, os conceitos de banco de dados relacional e esquema de banco de dados relacional como definido em [14]. Um **esquema de banco de dados relacional S** é um conjunto de esquemas relacionais S = {R_1, R_2, ..., Rm} e um conjunto de restrições de integridade IC. Uma **instância de banco de dados relacional BD** de S é um conjunto de instâncias de relações (ou simplesmente relações) BD = {r_1, r_2, ... , r_m} tal que cada r_i é uma instância de R_i e tal que as relações r_i satisfazem as restrições de integridade especificadas em IC. Na Figura 3.3.a, mostramos o esquema BD relacional pubs2 com as chaves primárias sublinhadas. Esse esquema relacional é equivalente ao esquema ERE da Figura 3.2.a. Na Subseção 3.3.4 mostramos como transformar um esquema ERE em um esquema BD relacional.

3.3.3 Restrições do Modelo Relacional

Essa seção apresenta alguns tipos de restrições que podem ser especificadas sobre o esquema de banco de dados relacional. Essas restrições incluem restrições de chave e restrições de integridade referencial. Outros tipos de restrições, tais como restrições de domínio e dependência de dados podem ser encontrados em [14, 15, 16].

3.3.3.1 Restrições de chave

Por definição todas as tuplas de uma relação são distintas. Isso significa que nenhum par de tuplas tem a mesma combinação de valores para todos os seus atributos. Normalmente, há alguns conjuntos de atributos de uma relação com a propriedade de assegurar que nenhum par de tuplas em qualquer relação, tenham a mesma combinação de valores para todos os seus atributos. Qualquer desses conjuntos de atributos é chamado de superchave (superkey). Assim, **superchave** é um atributo ou conjunto de atributos que pode ser usado para identificar cada relação de forma única.

AUTHORS(au_id: id, au_name: varchar, address: varchar, city: varchar, state: char, country: varchar, zipcode: char, phone: char, birthdate: datetime, age: smallint,)

PUBLICATION(title_id:tid, title: varchar, pubdate: datetime, pub_id: char) pub_id is FK to PUBLISHERS

BOOK(title_id:tid, edition: varchar, print: price: int) title_id is FK to PUBLICATION

PAPER(title_id: tid, vol.: char, no.: smallint) title_id is FK to PUBLICATION

TITLEAUTHOR(au_id:id, title_id: tid, au_ord: tinyint, royaltyper: int) au_id is FK to AUTHORS and title_id is FK to PUBLICATION

ROYSCHED(title_id: tid, lorange: int, hirange: int, royalty: int) title_id is FK to PUBLICATION

PUBLISHERS(pub_id: char, pub_name: varchar, city: varchar, state: char)

SALES(stor_id: char, ord_num: varchar, date: datetime) stor_id is FK to STORES

SALESDETAIL(stor_id: char, ord_num: varchar, title_id: tid, qty: smallint, discount: float) title_id is FK to PUBLICATION, stor_id and ord_num are FKs to SALES

STORES(stor_id: char, stor_name: varchar, stor_address: varchar, city: varchar, payterms: varchar)

DISCOUNTS(discounttype: varchar, stor_id: char, lowqty: smallint, highqty: smallint, discount: float) stor_id is FK to STORES

BLURBS(au_id: id, copy: text) au_id is FK to AUTHORS

AUTHORS-HOBBIES(au_id: id, hobbies: varchar) au_id is FK to AUTHORS

AU_PIX(au_id: id, pic: image, format_type: char, bytesize: int, pixwidth_hor: char,)pixwidth_ver: char) au_id is FK to AUTHORS

DEPENDENT(au_id: id, name: varchar, sex: char, birthdate: datetime) au_id is FK to AUTHORS

Figura 3.3.a O esquema SGBD relacional pubs2

Na relação PUBLICATION, *title_id* é uma superchave, mas *title_id* e *pubdate* juntos, também podem formar uma super chave. Uma superchave pode ter atributos redundantes. Um conceito mais útil é o de chave, que não tem redundância. Uma **chave** (key) é uma superchave em que nenhum atributo pode ser excluído sem que destrua suas propriedades de chave. Na realidade, uma chave é uma superchave mínima que uma relação deve ter. No nosso exemplo, *au_id* é um atributo chave, identificando entidades AUTHORS. Em geral, um esquema relacional pode ter mais que uma chave. Nesse caso, cada uma dessas chaves é chamada de **chave candidata** (candidate key). **Chave primária** (primary key) é uma chave candidata que é escolhida para uma aplicação particular. No nosso exemplo, *au_id* e *title_id* foram escolhidas como chaves primárias das relações AUTHORS e PUBLICATION respectivamente. O conceito de **chave estrangeira** (foreign key) é utilizado para fazer uma associação entre duas relações (entidades). Chave estrangeira de uma relação R_1 não é necessariamente uma chave primária de R_1, mas é a chave primária de outra relação R_2. Dadas duas relações, R_1 e R_2, um conjunto de atributos FK da relação R_1 é uma chave estrangeira de R_1 se os atributos em FK têm o mesmo domínio da chave primária PK da outra relação R_2. FK de R_1 pode ter diferentes nomes daqueles da PK de R_2. *Title_id* na relação ROYSCHED é uma chave estrangeira referenciando a chave primária da relação PUBLICATION.

3.3.3.2 Restrições de Integridade Referencial e Chaves estrangeiras

Uma das mais importantes restrições entre relações no modelo relacional é a de integridade referencial. Essa restrição é especificada entre dois esquemas relacionais e é usada para manter a consistência entre tuplas dos dois esquemas. Informalmente, a **restrição de integridade referencial** define que uma tupla numa relação que faz referência a uma outra relação deve se referir a uma tupla já existente naquela relação. Em outras palavras, a restrição de integridade referencial pode ser expressa como segue: se um atributo A de uma relação R_1 é declarada como uma chave estrangeira referenciando uma outra relação R_2, então todo valor contido em A, ou é um valor nulo (se admitido) ou é o valor da chave de R_2, ou seja, ou o valor em A não referencia nenhuma tupla, ou referencia uma tupla

já existente em R2. Assim, o valor contido em A não deve referenciar uma tupla que ainda vai ser criada em R2.

3.3.4 Transformação de Esquema ERE para Esquema Relacional

Nessa subseção, apresenta-se um algoritmo para mapear um esquema ERE para o equivalente esquema BD relacional [41]. O entendimento das regras desse algoritmo é muito importante, uma vez que nossas regras para transformar esquemas relacionais para esquemas OO, apresentado nos Capítulos 4 e 5, consideram o esquema ERE associado. Uma outra boa razão é que, da mesma forma que o modelo ER é hoje um modelo muito utilizado e útil para projetos conceituais de BD relacionais, o modelo ERE pode ser bastante útil para projetos conceituais de BD-OO, isto é, um esquema ERE poderia ser diretamente transformado para um esquema OO. Usando essas regras o esquema BD relacional mostrado na Figura 3.3.a é obtido a partir do esquema ERE da Figura 3.2.a.

3.3.4.1 Regra 1: Transformar Entidade em Relação

Toda entidade E é transformada em uma relação R(E). Cada atributo de E, que não seja do tipo set ou composto torna-se um atributo em R(E) e a chave de E torna-se a chave primária de R(E), denotada por chave(R(E)).

3.3.4.2 Regra 2: Transformar Atributo set

Cada atributo set A de E é transformado em uma relação separada R_A com dois atributos, K e A, onde K é igual à chave(R(E)). Chave(R_A) consiste de todos os atributos de R_A, isto é, é composta por K e A. Para assegurar que toda tupla em R_A corresponda a uma única tupla em R(E), K de R_A é definida como uma chave estrangeira referenciando chave(R(E)).

3.3.4.3 Regra 3: Transformar Atributo Composto

Há duas formas de transformar um atributo composto B de uma entidade E.

Regra 3.1: Usar só os atributos folhas e ignorar o atributo composto B. Por exemplo, se um atributo composto *Name* é depois descrito como *FisrtName* e *LastName*, então somente *FirstName* e *LastName* serão usados na relação transformada.

Regra 3.2: Transformar o atributo composto B numa relação separada B. Para assegurar que cada tupla em B corresponda a alguma tupla em R(E), a chave de R(E) é adicionada à relação B e é definida como a Chave(B) e como uma chave estrangeira de B referenciando a chave de R(E). Algumas vezes, é importante também assegurar a semântica de que cada tupla em R(E) tenha uma correspondente tupla em B. Por exemplo, isso acontece se quisermos que cada autor tenha uma fotografia (*au_pix* na Figura 3.2.a). Isto pode ser feito declarando-se a chave de R(E) também como uma chave estrangeira referenciando a chave de B.

3.3.4.4 Regra 4: Identificar Relacionamentos Um-para-Um e Um-para-muitos

Para cada relacionamento um-para-um ou um-para-muitos entre as entidades E_1 e E_2, a chave de $R(E_1)$ é adicionada em $R(E_2)$. Além disso, o atributo adicionado em $R(E_2)$ é definido como chave estrangeira referenciando a chave de $R(E_1)$. Se E_2 é uma entidade fraca e E_1 é sua correspondente entidade forte, então a chave de $R(E_2)$ consiste da chave de E_1 e da chave parcial de E_2. É possível que E_1 e E_2 sejam a mesma entidade, isto é, é um relacionamento unário.

3.3.4.5 Regra 5: Identificar Relacionamentos muitos-para-muitos

Transformar cada relacionamento muitos-para-muitos em uma relação separada. Tal relação consiste de todos os atributos chaves das entidades envolvidas e todos os atributos do próprio relacionamento. A chave da relação consiste de todas as chaves das entidades envolvidas e esses atributos são também definidos como chaves estrangeiras referenciando as respectivas relações.

3.3.4.6 Regra 6: Identificar Relacionamentos Subtipo/Supertipo

Se a entidade E_2 é um subtipo da entidade E_1, então $R(E_2)$ herda a chave de $R(E_1)$ e ela se torna também a chave de $R(E_2)$. Outros atributos de $R(E_1)$ não são explicitamente herdados de $R(E_2)$. Para assegurar a semântica de que toda instância descrita em $R(E_2)$ seja também descrita em $R(E_1)$, a chave de $R(E_2)$ é definida como chave estrangeira referenciando a chave de $R(E_1)$.

3.4 Modelo de Dados Orientado a Objetos

Em sistemas OO, tudo é considerado objeto. Objetos podem ser tão pequenos e simples como números e ícones e tão grandes e complexos quanto aviões e grandes sistemas de software. Nessa seção, reveremos os mais importantes conceitos do modelo de dados orientado a objetos [17,23], tais como objeto, classe, hierarquia de classes, herança, que serão usados em nosso algoritmo para transformar esquemas relacionais em esquemas OO, nos Capítulos 4 e 5.

3.4.1 Objeto, Identidade de Objeto e Objeto Complexo

Qualquer **objeto** do mundo real tem um **conjunto de atributos** genéricos. Por exemplo, um (objeto) computador pode ter atributos como marca, tipo, processador, velocidade, memória e HD entre outros vários possíveis atributos. Assim um computador com as seguintes especificações: Quantex PC Pentium 60mhz 8MB 512MB é um objeto (uma instância de computador). Da mesma forma, um Apple PowerMac PowerPC 100mhz 16MB 1GB é um outro objeto (uma outra instância de computador). Essencialmente, nesse caso, computador captura a definição daqueles objetos, isto é, computador é uma classe daqueles objetos. Voltaremos a discutir o conceito de classe depois. Como visto anteriormente, objetos têm diferentes níveis de grandeza e complexidade. Independentemente do tamanho e da complexidade dos objetos, para cada um deles é associado um estado e um comportamento.

O **estado**, na verdade a memória privada contendo a **estrutura de dados** do objeto, é representado pelos valores dos atributos dos objetos (também chamados de propriedades ou variáveis de instância). O **comportamento** é definido pelos **métodos atuando sobre o estado** do objeto. Um **método** consiste de dois componentes básicos: assinatura e corpo. A **assinatura** especifica o nome do método, os nomes e as classes dos parâmetros e a classe do resultado, se existir. O conjunto de assinaturas dos métodos de uma dada classe, representa a interface pública dos objetos daquela classe. O **corpo** representa a implementação do método e consiste de um conjunto de instruções expressas em qualquer linguagem de programação. Assim, as assinaturas de um dado objeto são as portas de entrada das mensagens que serão processadas pelos corpos dos respectivos métodos daquele objeto. Uma visão muito simplificada de um objeto é mostrada na Tabela 3.2 e uma visão esquemática do objeto computador acima descrito é mostrada na Tabela 3.3.

Tabela 3.2 Visão Simplificada de partes de um objeto

estado	estrutura de dados		memória privada
comportamento	método	assinatura	interface pública
		corpo	implementação do método

Tabela 3.3 Visão esquemática do objeto computador.

Estrutura de Dados	marca: string tipo: string processador: string velocidade: string memória: string preço: integer
Assinatura	inserir apagar mudar_memória(integer) mudar_preço(integer)
Corpo	implementação de inserir implementação de apagar implementação de mudar_memória implementação de mudar_preço

Cada objeto é identificado por um **identificador de objeto (OID)**. Os OIDs são gerados e mantidos pelo sistema. A identidade de um objeto tem uma existência independente dos valores dos atributos do objeto. Embora tanto o OID como a chave primária no modelo relacional tenham a função de identificar univocamente as entidades que representam, há algumas diferenças conceituais entre ambos, a saber:

- Uma chave é única dentro de uma relação, enquanto um OID é único dentro do sistema inteiro.
- A chave é implementada pelo programador de aplicações, enquanto um OID é implementado pelo sistema.
- Uma chave é muito mais significativa que um OID, para o usuário. Muito embora quando é necessário usar chaves longas, isso se torna extremamente ineficiente, principalmente quando usado como chave estrangeira. Em muitos casos, especialmente quando usados como chave estrangeira, os usuários tendem a colocar códigos artificiais que freqüentemente não têm nenhum significado semântico.
- Supondo a situação em que a chave precisa ser mudada: quando o número de funcionários e/ou o número de empresas crescem e a organização quer criar um identificador novo e mais significativo para os funcionários. Em SGBDs relacionais, o atributo chave contendo o identificador de um funcionário não pode ser mudado, porque seu valor identifica especificamente aquele funcionário. Conseguindo-se mudar esse valor, ele não estará mais identificando o mesmo funcionário e sim outro. Em SGBDs OO, um OID também não pode ser mudado, mas ele não carrega nenhuma informação semântica da aplicação, em oposição à chave em SGBDs relacionais. Em outras palavras, como um OID é criado e administrado pelo sistema, ele não contém informação tal como identificador de funcionário. Na realidade, em SGBDs OO, tal informação precisa ser um atributo como outro qualquer. E como um atributo, ele pode ser modificado.

Normalmente, os usuários não precisam ter acesso a um OID. Alguns sistemas admitem que os usuários tenham acesso a OIDs diretamente. Outros não admitem diretamente, mas permitem associação de nomes aos objetos (O2, por exemplo [11]).

Há dois tipos de objetos: simples e complexo. **Objetos simples** são todos os objetos primitivos, tais como inteiros ou caracteres. Em muitos modelos esses objetos não são representados como objetos e não possuem OIDs associados a eles. Eles são auto-identificados pelos seus próprios valores. O valor de um atributo de objeto pode ser outro objeto, primitivo ou não. **Objetos complexos** são compostos por outros objetos, usando várias estruturas, tais como **set, tuple, array, list, bag**, etc. Portanto, objetos (complexos) podem ser recursivamente definidos em termos de outros objetos. Graças ao OID, objetos podem compartilhar outros objetos e uma rede de objetos pode ser construída. A hierarquia que faz a conexão de atributos complexos com seus domínios é conhecida como hierarquia de composição. A Figura 3.4.b mostrando parte do esquema BD OO de pubs2 é um exemplo de uma rede de objetos, com objetos definidos em termos de outros. A classe publication tem a classe publishers definida como domínio do atributo complexo chamado *pub_id*.

```
class publishers
type struct( pub_id: string
             pub_name: string
             city: string
             state: string)


 class publication
 type struct(  title_id: string
              title: string
              pub_id: publishers
              price: float
             pubdate: date)
```

Figura 3.4.b Hierarquia de composição - Objetos definidos em termos de objetos.

3.4.2 Encapsulamento

Cada objeto consiste de uma interface bem definida e uma implementação. A **interface** é a especificação de um conjunto de operações (isto é, o conjunto de assinaturas dos métodos) que podem ser invocadas sobre o objeto e é sua única parte visível (interface

pública do objeto). A **implementação** contém dados, isto é, o estado ou a representação estática do objeto, e os corpos dos métodos que provêem o código para cada operação. O estado encapsula os dados e o corpo encapsula a implementação do comportamento de um objeto. Para cada mensagem entendida por um objeto, há uma correspondente assinatura de método que recebe a mensagem e um correspondente corpo de método que processa a mensagem. Nesse sentido, as duas mais importantes capacidades que a orientação a objetos pode prover através da idéia de encapsulamento são:

- fazer uma clara distinção entre especificação e implementação de uma operação
- modularidade

Como um exemplo, se tomarmos a classe book como mostrada na Figura 3.4.c, o método *increase_price* tem como assinatura seu próprio cabeçalho que é, junto com as outras assinaturas, a única parte visível de todos os objetos da classe book. A representação dos dados e o corpo do método são encapsulados (eles não são acessíveis exceto através da assinatura do método).

```
class book
type struct( au_id: string
             au_name: string
             pubdate: date
              edition: string
              price: real)

method increase_price(amount: real)

body increase_price(amount: real)
  { *self.price += amount}
```

Figura 3.4.c Encapsulamento de dados e do corpo do método

Embora na comunidade de linguagem de programação a representação de dados seja, claramente, parte da implementação do objeto e seja invisível ao mundo exterior a esse objeto, na comunidade de banco de dados, não é tão claro se a representação dos dados deva ser sempre parte da implementação ou se deva ser parte da interface. Comandos de consulta são freqüentemente expressos em termos de predicados sobre valores de atributos. Assim, tratamento de comandos de consulta é uma situação onde a violação de encapsulamento é quase obrigatória [1, 36]. Quando comandos de consulta são usados, muitos SGBDs OO, incluindo O2 [1] em alguns casos, não garantem que o encapsulamento seja observado.

3.4.3 Classe, Hierarquia de Classes, Herança e Conceitos Relacionados

Uma **classe** é um conjunto de objetos tendo exatamente a mesma estrutura interna e, portanto, os mesmos atributos e os mesmos métodos. Objetos pertencentes a uma classe são chamados de instâncias daquela classe. Uma **classe** descreve a forma (atributos) de suas instâncias e as operações (métodos) aplicáveis a essas instâncias. Todas as instâncias de uma dada classe têm um valor para cada atributo definido para aquela classe e elas respondem a todas as mensagens que invocam os métodos da classe. Os modelos de dados OO usam o conceito de classe como base para a instanciação de objetos. Nesse sentido, uma **classe** é um objeto que atua como um template. Usando classe e o mecanismo de instanciação, a mesma definição de classe pode ser reusada para gerar muitos objetos com a mesma estrutura e com o mesmo comportamento, evitando a especificação e a armazenagem de muita informação redundante. O conceito de hierarquia de classes pode estender essa capacidade de encapsulamento de informação. A **hierarquia de classes** modela a semântica do relacionamento IS_A entre classes, isto é, ela admite a definição de uma classe chamada de **subclasse**, como uma especialização de uma outra classe, chamada **superclasse**. Uma subclasse herda os atributos e os métodos de sua superclasse e, além disso, pode ter atributos e métodos adicionais, específicos da subclasse. Na Figura 3.4.d, as classes book e paper são subclasses da classe publication, isto é, as classes book e paper

herdam todos os atributos e métodos da classe publication. Em outras palavras, "book IS_A publication" e "paper IS_A publication". Alguns sistemas suportam herança múltipla (O2, por exemplo) e outros só suportam herança simples. Herança múltipla significa que uma classe pode ter várias superclasses e herança simples significa que uma classe pode ter somente uma superclasse. O mecanismo de subclasse/superclasse admite a propriedade chamada de polimorfismo de inclusão (inclusion polymorphism), que significa que uma instância de uma classe pode aparecer ou ser usada em qualquer situação onde uma instância de sua superclasse é esperada.

```
class publication
 type struct(  title_id: string
               title: string
              pubdate: date
               pub_id: publishers)

class book
inherit publication
type struct( edition: string
             print: string
             price: float)


 class paper
 inherit publication
 type struct( vol.: integer
              no.: integer)
```

Figura 3.4.d Relacionamento subclasse/superclasse, modelando o relacionamento IS_A

Uma outra dimensão na organização de classes, além da hierarquia de classes, é a chamada hierarquia de composição. A **hierarquia de composição** pode ser representada por um grafo, onde os nós são classes. Um exemplo de hierarquia de composição é mostrada na Figura 3.4.b. Um arco pode conectar duas classes (publication para publishers) quando a classe publishers é o domínio de um atributo (*pub_id*) da classe publication. Assim, uma hierarquia de composição é um conjunto de classes, composta por

classes definidas em termos de outras classes que, por sua vez, podem ser definidas em termos de outras classes, de forma aninhada. Na realidade, essa hierarquia não é uma hierarquia no sentido amplo, uma vez que classes podem ser definidas recursivamente. Por exemplo, a classe employee tem um atributo *manager* cujo domínio é a própria classe employee, como mostrada na Figura 3.4.e.

```
class employee
type struct(e_name: string
            status: string
            salary: real
            hire_year: integer
            manager: employee)
```

Figura 3.4.e Hierarquia de Composição - Recursividade com a própria classe

Alguns modelos de dados OO fazem distinção entre tipos e classes. Em O2, por exemplo, **classes**, cujas instâncias são objetos, **encapsulam dados e comportamento**, isto é, os dados e o comportamento não são diretamente manipuláveis por outros objetos; e **tipos**, cujas instâncias são valores, **não encapsulam nada**, isto é, suas estruturas de dados são diretamente manipuláveis por outros objetos.

3.4.4 Overriding, Polimorfismo e Late Binding

A herança pressupõe que uma subclasse herda todas as propriedades (atributos e métodos) exatamente como elas estão definidas na sua superclasse. Por **"overriding"** entende-se que uma subclasse herda somente o significado semântico de algumas das propriedades da sua superclasse. Especificamente, "overriding" ocorre freqüentemente, quando se usa uma implementação de método na subclasse, diferente da implementação de método de sua superclasse, embora as assinaturas sejam as mesmas. Supondo a operação show, como mostrada na Figura 3.4.f, num sistema multimídia de

cinema hipotético. A operação show pode mostrar desde biografias de atores e diretores até temas musicais de filmes, passando por melhores desempenhos e prêmios. A operação tem um único nome (show) e pode ser invocada indiscriminadamente para vários objetos (biografia, fotografia, melhor performance, etc). Assim, "overriding" é a redefinição da implementação de operações. O resultado é um único nome de operação (mesma assinatura) para diferentes corpos de método. A capacidade de poder utilizar a mesma propriedade (ou método) em diferentes contextos é chamada de **polimorfismo**.

Para prover a capacidade de "overriding", algumas vezes o sistema não consegue fazer a ligação do nome da operação com o correspondente corpo de método em tempo de compilação, mas deve fazê-la em tempo de execução. Essa ligação atrasada é chamada de **"late binding"** (ligação tardia).

```
class star
type struct(star_id: string
            star_name: string
            star_ sex: string
            star_ age: integer)
method show()
body show
{ print(star_id)
  print(star_name)
  print(star_sex)
  print(star_age) }

class best_performance
inherit star
type struct(clip: avi)

method show()
body show
{ player(clip) }
```

```
class biography
inherit star
type struct(h_ text: html)

method show()
body show(h_text)
{ navigator(h_text) }

class picture
inherit star
type struct(pic: ps)

method show()
body show
{ xview pic }
```

Figura 3.4.f Exemplo de "overriding" e polimorfismo

3.5 Resumo do Capítulo 3

Nesse capítulo, foram revistos alguns dos mais importantes conceitos de quatro modelos de dados. O entendimento de tais conceitos foram fundamentais para afinar o algoritmo de transformação de esquemas apresentado no próximo capítulo. Os modelos de dados revistos foram: Modelo Entidade-Relacionamento (MER), Modelo Entidade-Relacionamento Estendido (MERE), Modelo de Dados Relacional (RDM) e Modelo de Dados Orientado a Objetos (MDOO). Além disso, apresentou-se resumidamente um algoritmo para transformar esquemas ERE para esquemas relacionais, uma vez que o modelo ERE é um dos mais populares para projeto conceitual de banco de dados relacional.

CAPÍTULO 4

Transformação de Esquema Relacional para Esquema Orientado a Objetos

Nesse capítulo apresenta-se o algoritmo para transformar esquemas de bancos de dados relacionais para esquemas de bancos de dados orientados a objetos. Considera-se que para cada relação há uma entidade ou relacionamento correspondente no esquema conceitual representado no modelo Entidade-Relacionamento Estendido (ERE). Em outras palavras, o esquema de banco de dados relacional foi transformado de um único esquema ERE (no sentido de que cada entidade está relacionada a alguma outra (ou outras) entidade(s) de alguma forma) através das regras apresentadas no Capítulo 3. Na Seção 4.1, expõem-se algumas convenções adotadas para facilitar a apresentação do algoritmo e algumas restrições que definem os limites do transformador de esquemas. Na Seção 4.2, mostra-se o algoritmo do transformador de esquemas, explicando cada uma de suas 7 regras através de exemplos extraídos do banco de dados pubs2, com algumas modificações para facilitar a explanação. A Seção 4.3 provê uma discussão de como métodos poderiam ser adicionados ao esquema orientado a objetos, muito embora o objetivo, nesse momento, não seja automatizar o processo de criação de métodos.

4.1 Convenções e Restrições

Uma vez que o modelo de dados relacional não tem o conceito equivalente a métodos do modelo de dados orientado a objetos, o algoritmo de transformação trata somente de aspectos de relações, atributos, chaves e relacionamentos. Além disso, como em um sistema de múltiplos bancos de dados o enfoque principal é sobre recuperação de informação e os SGBDs locais devem ter plena autonomia em termos de novas aplicações e administração de banco de dados, a criação de métodos não é tão importante nesse momento. Claro que, quando um sistema de múltiplos bancos de dados admite transações de inserção e atualização de informações, as tarefas de criação de métodos são extremamente necessárias. A Seção 4.3 apresenta uma discussão sobre adição de métodos ao esquema orientado a objetos transformado. Como visto antes, esse algoritmo é genérico o suficiente para ser aplicado em transformação de esquemas para a maioria de SGBDs relacionais e orientados a objetos, com pequenos ajustes. Para que a apresentação do algoritmo seja genérica, usa-se a notação de esquema relacional, tal como aquela apresentada na Seção 3.3. Da mesma forma, usa-se uma notação de esquema orientado a objetos num formato muito próximo daquele utilizado na definição de classes em C++.

Para cada relação R, se ela é transformada de uma entidade E, denota-se o conjunto de entidades como E(R). Também para denotar a classe transformada da relação R usa-se C(R). Durante o processo de transformação de um esquema de banco de dados relacional para um esquema de banco de dados orientado a objetos, é algumas vezes criticamente importante decifrar (compreender) como cada relação foi transformada a partir de um esquema ERE. Por exemplo, se a relação R_1 e a relação R_2 foram transformadas das entidades E_1 e E_2, respectivamente, e E_2 é um subtipo de E_1, então pode-se determinar que a classe $C(R_2)$ é uma subclasse de $C(R_1)$. Note que o esquema ERE original já não está mais disponível após ter sido transformado em um esquema relacional. Geralmente não é possível reconstruir completa e automaticamente um esquema ERE de um esquema relacional (que foi transformado de um esquema ERE). Por exemplo, considere os dois casos seguintes:

Caso 1- R é transformada de um atributo composto de uma entidade E_1, a chave de R é definida como chave estrangeira referenciando a chave de $R(E_1)$, mas a chave de $R(E_1)$ não é definida como chave estrangeira referenciando a chave de R.

Caso 2 - R é transformada de uma entidade E que é um subtipo da entidade E_1 e E_1 não participa de outros relacionamentos no esquema ERE original.

Os dois Rs nos dois casos acima estão associados com exatamente as mesmas informações, isto é, a chave estrangeira de R é também sua chave, ela referencia a chave de outra relação e ela não tem nenhum outro relacionamento chave/chave-estrangeira. Claramente, não é possível saber se R foi transformada de um atributo composto ou de um subtipo. Portanto, algumas vezes, é necessário que o usuário do transformador de esquemas forneça informações adicionais. Quando o transformador de esquemas precisa de informações do usuário, ele faz uma pergunta simples e de fácil resposta. Somente um "sim" ou "não" é necessário. Assume-se que o usuário tem conhecimento suficiente sobre a semântica dos dados no SGBD local. Por exemplo, o usuário poderia ser o administrador de banco de dados (DBA) do SGBD relacional.

4.2 O Algoritmo de Transformação de Esquemas

Após estudos e análises dos algoritmos apresentados em [22, 43, 44], concluiu-se que todos eles apresentavam limitações, ou no processo de transformação ou no esquema OO transformado. Conforme visto no Capítulo 2, o algoritmo apresentado por Keim, Kriegel e Miethsam em [22], não considera a construção de hierarquia de classes no esquema orientado a objetos resultante, enquanto o algoritmo de Yan e Ling em [43], não considera criação de conjuntos de atributos (set) para simplificar o esquema orientado a objetos resultante. Muito embora o algoritmo de Yu e outros [44] possa gerar esquemas tipicamente orientados a objetos, contendo hierarquia de classes, hierarquia de composições e de conjuntos atributos, ele gera um número desnecessariamente grande de classes.

O algoritmo a ser apresentado foi criado como uma alternativa a esses algoritmos que têm o mesmo objetivo, ou seja, transformar esquemas relacionais em esquemas orientados a objetos. O algoritmo constitui-se de 7 regras de transformação. Para facilitar o entendimento dessas regras de transformação, exemplos são providos após a apresentação de cada uma das regras. Conforme dito antes, os conceitos revistos no Capítulo 3 sobre MER, MERE, MDR e MDOO são bastante importantes para se entender as regras de transformação que serão descritas agora.

4.2.1 Transformando relações para classes

4.2.1.1 REGRA 1: Transformar cada relação em classe

Cada relação R é transformada em uma classe C(R). Cada atributo de R torna-se um atributo de C(R). O tipo de dado de um atributo da relação é mapeado para o domínio do correspondente atributo da classe, baseado numa tabela preestabelecida de mapeamento. Por exemplo, smallint é mapeada para integer.

4.2.1.2 Exemplo para REGRA 1

Os atributos com os respectivos domínios no **esquema relacional DISCOUNTS** são transformados para os atributos e domínios definidos na **classe discounts**, mostrados a seguir:

DISCOUNTS(discounttype: varchar, stor_id: char, lowqty: smallint, highqty: smallint, discount: float)

```
class discounts {
        discounttype: string
        stor_id: string
        lowqty: integer
        highqty: integer
        discount: float
        }
```

4.2.2 Identificando relacionamento subclasse/superclasse

4.2.2.1 REGRA 2 : Identificar relacionamento subclasse/superclasse

Para que $C(R_0)$ seja uma subclasse de $C(R_1)$, $E(R_0)$ deve ser um subtipo de $E(R_1)$. Se $E(R_0)$ é um subtipo de $E(R_1)$, então R_0 e R_1 têm a mesma chave e a chave de R_0 é também definida como uma chave estrangeira referenciando a chave de R_1, mas a chave de R_1 não é definida como uma chave estrangeira referenciando a chave de R_0. Nessa situação, se R_0 também tem relacionamentos chave/chave-estrangeira com outras relações, então ***R_0 é transformada de um subtipo de $E(R_1)$***, senão R_0 poderia ser transformada de *um subtipo de $E(R_1)$* (caso a) ou de *um atributo composto de $E(R_1)$* (caso b). Como não há uma forma definitiva para que o transformador de esquemas consiga distinguir os dois casos sem informação adicional, uma sessão interativa é iniciada pelo transformador de esquemas, sempre que a situação acima é detectada. Se $C(R_0)$ é uma subclasse de $C(R_1)$, então deve ser verdade que todo objeto do mundo real descrito em $C(R_0)$ seja também descrito em $C(R_1)$. Por exemplo, todo livro é também descrito como uma publicação. Portanto, a questão feita pelo transformador de esquemas poderia ser simplesmente "Todo objeto em $C(R_0)$ é também um objeto em $C(R_1)$? ". Uma resposta **sim** indica que ***R0 foi transformado de um subtipo de E(R1)* (caso a)**. Uma resposta **não** indica que *R0 foi transformado de um atributo composto de E(R1)* (caso b). Esse segundo caso será tratado pela Regra 3.

Se for decidido que $C(R_0)$ se tornará uma subclasse de $C(R_1)$, então

1) Adicione ": $C(R_1)$" após "$C(R_0)$" na definição de classe de $C(R_0)$ e

2) Remova de $C(R_0)$ todos os atributos que aparecem em $C(R_1)$.

4.2.2.2 Exemplos para REGRA 2

Considerando somente os **esquemas relacionais SALES e SALESDETAIL**, eles seriam transformados automaticamente para as **classes sales e salesdetail**, onde a **classe salesdetail herda a classe sales**.

SALES(stor_id: char, ord_num: varchar, date: datetime)

SALESDETAIL(stor_id: char, ord_num: char, title_id: tid, qty: smallint, discount: float)
title_id is FK to PUBLICATION, **stor_id and ord_num are FKs to SALES**

```
class sales {
        stor_id: string
        ord_num: string
        date: date
        }

class salesdetail: sales {
        title_id: string
        qty: integer
        discount: float
        }
```

Agora, tendo-se os **esquemas relacionais PUBLICATION, BOOK e PAPER**, se as respostas para as questões "Todo BOOK é também uma PUBLICATION ?" e "Todo PAPER é também uma PUBLICATION ?" forem **sim** esses esquemas relacionais serão transformados para as **classes publication, book e paper**, com as **classes book e paper herdando a classe publication.**

.

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime),

BOOK(title_id: tid, edition: varchar, print: varchar, price: smallint) title_id is FK to PUBLICATION

PAPER(title_id: tid, vol.: smallint, no.: smallint) title_id is FK to PUBLICATION

```
class publication {
        title_id: string
        title: string
        pubdate: date
        }
class book : publication {
        edition: string
        print: string
        price: integer
        }
class paper : publication {
        vol.: integer
        no.: integer
        }
```

4.2.3 Identificando e transformando relações transformadas de atributos compostos

A regra número 3 identifica e transforma cada relação R que foi transformada a partir de um atributo composto A de uma entidade E.

4.2.3.1 REGRA 3 : Identificar atributos compostos

O caso (b) da Regra 2 identifica uma situação onde uma relação é transformada de um atributo composto. Repetindo a situação: se R_0 e R_1 têm a mesma definição de chave, se a chave de R_0 é também definida como uma chave estrangeira referenciando a chave de R_1, se a chave de R_1 não é definida como uma chave estrangeira referenciando a chave de R_0 e se R_0 não tem relacionamentos chave/chave-estrangeira com outras relações, então **R_0 poderia ser transformada** de um subtipo de E(R1) (caso a) ou **de *um atributo composto de $E(R_1)$* (caso b da Regra 2)**. Uma outra situação é descrita a seguir.

Se foi detectado que R_0 e R_1 têm a mesma definição de chave e a chave de R_0 é uma chave estrangeira referenciando a chave de R_1 e a chave de R_1 é uma chave estrangeira referenciando a chave de R_0, então sabe-se que não há relacionamento subclasse/superclasse entre $C(R_0)$ e $C(R_1)$. Na realidade, esse é o caso em que ou R_0 é transformado de um atributo composto de $E(R_1)$ ou R_1 é transformado de um atributo composto de $E(R_0)$. Nessa situação, se um deles tem relacionamento chave/chave-estrangeira com outras relações, então o **outro é transformado de um atributo composto (caso c)**, desde que relações transformadas de atributos compostos podem ter relacionamentos chave/chave-estrangeira com somente uma outra relação. Entretanto, se nenhum deles tem relacionamento chave/chave-estrangeira com outras relações, então o transformador de esquemas não pode identificar qual dos dois foi transformado de um atributo composto do outro, sem informação adicional. Nessa nova situação, uma pergunta será feita ao usuário. Uma vez que, R_0 sendo transformado de um atributo composto de $E(R_1)$, tem a semântica de que R_0 é parte da

descrição de R_1, a pergunta feita poderia ser "R_0 é parte da descrição de R_1 ?". Uma resposta **sim** indica que ***R_0 foi transformado de um atributo composto de $E(R_1)$*** **(caso d)**. Uma resposta **não** indica que ***R_1 foi transformado de um atributo composto de $E(R_0)$*** **(caso e)**.

Se R_0 é transformado de um atributo composto de $E(R_1)$, então transforme $C(R_0)$ para uma estrutura (com a semântica de "struct" na linguagem C) em $C(R_1)$ como segue:

1) Criar um atributo com nome R_0 em $C(R_1)$ cujo domínio é do tipo struct formado por todos os atributos de R_0, exceto pela chave de R_0.
2) Descartar $C(R_0)$.

4.2.3.2 Exemplos para REGRA 3

Considerando os **esquemas relacionais PUBLICATION, BOOK e PAPER** no exemplo da Regra 2, se as respostas para as questões "Todo BOOK é também uma PUBLICATION ?" e "Todo PAPER é também uma PUBLICATION ?" fossem **não (caso b)** aqueles esquemas relacionais seriam transformados para a seguinte **classe publication**, com **book e paper como "struct" (atributo composto) dentro da classe publication**.

```
class publication {
        title_id: string
        title: string
        pubdate: date
        book: struct(edition: string
                     print: string
                     price: integer)
        paper: struct(vol.: integer
                      no.: integer)
        }
```

Por outro lado, os **esquemas relacionais PUBLICATION e ROYSCHED** como definidos abaixo, serão transformados na **classe publication** que terá um atributo composto constituído dos atributos de ROYSCHED **(caso c)**, como mostrado após os esquemas relacionais.

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime, pub_id: char) title_id is FK to ROYSCHED, **pub_id is FK to PUBLISHERS**

ROYSCHED(title_id: tid, lorange: integer, hirange: integer, royalty: integer) title_id is FK to PUBLICATION

```
class publication {
        title_id: string
        title: string
        pubdate: date
        pub_id: string
        roysched: struct(lorange: integer
                         hirange: integer
                         royalty: integer)
        }
```

Substituindo-se o esquema relacional PUBLICATION acima pelo mostrado abaixo, ambos os esquemas, PUBLICATION e ROYSCHED serão completamente simétricos, no sentido de que suas chaves estrangeiras serão iguais às respectivas chaves primárias e suas chaves estrangeiras referenciarão a chave primária do outro. Dessa forma, se a resposta para a pergunta "ROYSCHED é parte da descrição de PUBLICATION ?" fosse **sim (caso d)** o transformador de esquemas geraria a mesma definição de classe mostrada acima.

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime, pub_id: char) title_id is FK to ROYSCHED

E se a resposta para a questão acima fosse **não (caso e)**, o transformador de esquemas geraria a seguinte definição de classe.

```
class roysched {
        title_id: string
        lorange: integer
        hirange: integer
        royalty: integer
        publication: struct(title: string
                            pubdate: date)
        }
```

4.2.4 Identificando e transformando relações transformadas de atributo conjunto

4.2.4.1 REGRA 4 : Identificar e transformar atributo conjunto

Se a relação R_0 é transformada de um atributo A (do tipo conjunto) da entidade E_1, então tem-se

(1) Chave(R_0) consiste de todos os atributos de R_0,

(2) Atr(R_0) - {A} = Chave(R_1), onde Atr(R_0) é o conjunto de todos os atributos de R_0 e R_1 = R(E_1),

(3) Atr(R_0) - {A} é a chave estrangeira referenciando Chave(R_1) e

(4) R_0 não tem relacionamentos chave/chave-estrangeira com outras relações

As condições acima são necessárias, mas não suficientes. Por exemplo, se R_0 é transformado de uma entidade fraca E_0 cuja correspondente entidade forte é E_1 e E_0 não tem relacionamentos chave/chave-estrangeira com outras relações, então R_0 e R_1 = R(E_1) também satisfazem as condições acima, quando E_0 tem somente um atributo (isto é, a chave parcial). Portanto, nesse caso, uma outra sessão interativa é iniciada pelo transformador de esquemas. Se as relações R_0 e R_1 satisfazem as condições

acima, então faz-se a pergunta "R_0 é parte da descrição de R_1 ?". Se a resposta for **sim** (isto é, **R_0 é transformado de um atributo do tipo conjunto**), então

1) Criar um atributo "set" em $C(R_1)$ adicionando "A: set(D)" na definição de classe de $C(R_1)$, onde D é o tipo de dado de A em R_1, e
2) Descartar $C(R_0)$.

Se a resposta para a pergunta acima for **não**, não faça nada nesse momento (**a Regra 7 tratará disso depois**).

4.2.4.2 Exemplo para REGRA 4

Considerando os esquemas relacionais HOBBIES e AUTHORS (mostrados abaixo) como R_0 e R_1, respectivamente, todas as condições da Regra 4 são satisfeitas. Se a resposta para a pergunta "HOBBIES é parte da descrição de AUTHORS ?" for **sim**, então um atributo de nome hobbies, tendo como domínio um "set" do domínio de HOBBIES, será criado na classe authors.

HOBBIES(au_id: id, hobbies: varchar) au_id is FK to AUTHORS e

AUTHORS(au_id: id, au_name: varchar, phone: char)

```
class authors {
        au_id: string
        au_name: string
        phone: string
        hobbies: set(string)
        }
```

4.2.5 Identificando e transformando relações transformadas de relacionamentos unários muitos-para-muitos

4.2.5.1 REGRA 5 : Identificar e transformar relacionamentos unários muitos-para-muitos

Uma relação R_0 é transformada de um relacionamento unário muitos-para-muitos sobre a entidade E_1 se e somente se Chave(R_0) consiste de exatamente duas chaves estrangeiras, ambas referenciando Chave(R_1), onde R_1 = R(E_1). Considerando a relação R_0 transformada de um relacionamento unário muitos-para-muitos sobre a entidade E_1 e R_1 transformado de uma entidade E_1, se R_0 não contém outros atributos além das chaves estrangeiras (isso significa que o relacionamento unário muitos-para-muitos não tem seus próprios atributos), então

(1) criar dois atributos "set" em C(R_1), sendo um "R_0: set(C(R_1))" e o outro "Used_as_R_0: set(C(R_1))", e

(2) Descartar C(R_0).

Agora, considerando a relação R_0 transformada de um relacionamento unário muitos-para-muitos sobre a entidade E_1 e R_1 transformado de uma entidade E_1, se R_0 contém outros atributos, vamos dizer A, B, ... além das chaves estrangeiras (isso significa que o relacionamento unário muitos-para-muitos tem seus próprios atributos), então

1) Criar dois atributos "set" em C(R_1), sendo um "R_0: set(struct(R_0: C(R_1), A: D(A), B: D(B), ...)" e o outro "used_as_R_0: set(struct(R_0: C(R_1), A: D(A), B: D(B), ...)" onde D(A), D(B), ... são os tipos de dados de A, B, ... em R_1, respectivamente, e
2) Descartar C(R_0).

4.2.5.2 Exemplos para REGRA 5

Por essa regra, a partir dos **esquemas relacionais PREREQUISITE e COURSE** (mostrados abaixo) transformados do relacionamento unário muitos-para-muitos prerequisite sobre a entidade course (conforme Figura 3.1.4.a), **dois atributos "set" serão criados na classe course**, sendo um "prerequisite: set(course)" que especifica, para cada

curso, o conjunto dos cursos prerequisitos, e o outro "used_as_prerequisite: set(course)" que especifica, para cada curso **c**, o conjunto de cursos que tem **c** como seus prerequisitos.

PREREQUISITE (is_prerequisite: char, has_prerequisite: char) **is_prerequisite is FK to COURSE** and **has_prerequisite is FK to COURSE**

COURSE(course_no: char, title: varchar)

class course {
course_no: string
title: string
prerequisite: set(course)
used_as_prerequisite: set(course)
}

Agora, considerando o **esquema relacional PREREQUISITE** acima **com mais um atributo** (por exemplo, semester_to_take), além da chave primária composta, **dois atributos serão criados, na classe course**, sendo um "prerequisite: set(struct(course: course, semester_to_take: string))" que especifica, para cada curso, o conjunto de cursos prerequisitos e os semestres em que esses cursos devem ser cursados, e o outro "used_as_prerequisite: set(struct(course: course, semester_to_take: string))" que especifica, para cada curso **c**, o conjunto de cursos que têm **c** como seus prerequisitos e os semestres em que esses cursos devem ser cursados. A definição da classe course transformada das relações COURSE e PREREQUISITE modificada fica como mostrada abaixo.

PREREQUISITE (is_prerequisite: char, has_prerequisite: char, semester_to_take: char) is_prerequisite is FK to COURSE and has_prerequisite is FK to COURSE

```
class course {
        course_no: string
          title: string
          prerequisite: set(struct(
                    course: course
                    semester_to_take: string))
          used_as_prerequisite: set(struct(
                    course: course
                    semester_to_take: string))
        }
```

Note que alguém poderia pensar que o atributo "used_as_prerequisite" não é realmente necessário uma vez que tal informação pode ser encontrada sem ele, e que ele pode também introduzir alguns dados redundantes. Nosso argumento a favor de se criar o atributo "used_as_prerequisite" são os seguintes. Primeiro, enquanto linguagens de consulta orientadas a objetos podem ser usadas para "navegar" de um atributo complexo para sua classe de domínio (isto é, através da expressão de um caminho com pontos) com facilidade, freqüentemente torna-se difícil especificar o caminho inverso. Como resultado, sem o atributo "used_as_prerequisite" será menos fácil para um usuário BDOO especificar o comando de consulta para buscar os cursos que usam um dado curso como seus prerequisitos. Segundo, lembrar que estamos criando um esquema (ou visão) virtual orientado a objetos sobre SGBDs relacionais. Na medida em que o esquema OO é virtual e todos os dados estão na realidade armazenados nos SGBDs locais, nunca ocorrerá redundância de dados no nível da API.

4.2.6 Identificando e transformando relação transformada de relacionamento binário muitos-para-muitos

4.2.6.1 REGRA 6 : Identificar e transformar relacionamento binário muitos-para-muitos

Uma relação R_0 é transformada de um relacionamento binário muitos-para-muitos entre as entidades E_1 e E_2 se e somente se Chave(R_0) consiste de exatamente duas chaves estrangeiras, uma referenciando Chave(R_1) e a outra referenciando Chave(R_2).

Considerando que uma relação R_0 é transformada de um relacionamento binário muitos-para-muitos entre E_1 e E_2, $R_1 = R(E_1)$, $R_2 = R(E_2)$ e se R_0 não contém nenhum outro atributo além das duas chaves estrangeiras (isso significa que o relacionamento muitos-para-muitos não tem seus próprios atributos) então

1) Criar um atributo "set", adicionando-se "R_2: set($C(R_2)$)" na definição de $C(R_1)$,
2) Criar um atributo "set", adicionando-se "R_1: set($C(R_1)$)" na definição de $C(R_2)$, e
3) Descartar $C(R_0)$.

Agora, considerando que uma relação R_0 é transformada de um relacionamento binário muitos-para-muitos entre E_1 e E_2, $R_1 = R(E_1)$, $R_2 = R(E_2)$ e se R_0 contém outros atributos, por exemplo A, B, ..., além das duas chaves estrangeiras (isso significa que o relacionamento muitos-para-muitos tem seus próprios atributos) então

1) Criar um atributo "set", adicionando-se "R_0: set(tuple(R_2: $C(R_2)$, A: D(A), B: D(B), ...)" na definição de $C(R_1)$,
2) Criar um atributo "set", adicionando-se "R_0: set(tuple(R_1: $C(R_1)$, A: D(A), B: D(B), ...)" na definição de $C(R_2)$, e
3) Descartar $C(R_0)$.

4.2.6.2 Exemplos para REGRA 6

Considerando um relacionamento binário muitos-para-muitos sem nenhum outro atributo além das chaves, se tivermos os **esquemas relacionais TAKE, STUDENT e COURSE** (mostrados abaixo), como R_0, R_1 and R_2 respectivamente, o transformador de esquemas gerará as definições das **classes student e course** abaixo.

TAKE(student_id: char, course_no: char) student_id is FK to STUDENT and course_no is FK to COURSE

STUDENT(student_id: char, name: varchar, age: smallint, gpa: char)

COURSE(course_no: char, title: varchar)

```
class student {
        student_id: string
        name: string
        age: integer
        gpa: string
        course: set(course)
        }

class course {
        course_no: string
        title: string
        student: set(student)
        }
```

Agora, considerando um relacionamento binário muitos-para-muitos com outros atributos além das chaves, se tivermos os seguintes **esquemas relacionais TITLEAUTHOR, AUTHORS e PUBLICATION** (mostrados abaixo), como R_0, R_1 and R_2 respectivamente, o transformador de esquemas gerará as definições das **classes authors e publication** como se fossem transformadas de um relacionamento binário muitos-para-muitos titleauthor entre as entidades authors e publication.

TITLEAUTHOR(au_id: id, title_id: tid, au_ord: tinyint, royaltyper: int) **au_id is FK to AUTHORS** and **title_id is FK to PUBLICATION**,

AUTHORS(au_id: id, au_name: varchar, phone: char)

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime)

```
class authors {
        au_id: string
        au_name: string
        phone: string
        titleauthor: set(struct(
                        publication: publication
                        au_ord: integer
                        royaltyper: integer))
        }
```

```
class publication {
        title_id: string
        title: string
        pubdate: date
        titleauthor: set(struct(
                        authors: authors
                        au_ord: integer
                        royaltyper: integer))
        }
```

4.2.7 Expandindo hierarquia de composição

4.2.7.1 REGRA 7: Expandir hierarquia de composição

Se R_0 ainda tem um correspondente $C(R_0)$, isto é, se $C(R_0)$ ainda não foi descartado por alguma das regras acima (significando que R_0 não é transformado de um atributo "set", ou de um atributo composto, ou de um relacionamento muitos-para-muitos), e R_0 tem uma chave estrangeira FK referenciando Chave(R_1) e FK não é igual Chave(R_0) (significando que essa FK não torna $C(R_0)$ uma subclasse de $C(R_1)$, mas é possível que R_0 seja transformado de uma entidade fraca), então

Caso 1: Se FK contém somente um atributo, então

mudar o domínio de FK para $C(R_1)$.

Caso 2: Se FK tem mais que um atributo, então

1) Remover todos os atributos em FK de $C(R_0)$ que não sejam utilizados para outros propósitos (isto é, como chave estrangeira para outra relação) e
2) Criar um novo atributo R_1 (em R_0) com domínio $C(R_1)$.

4.2.7.2 Exemplo para REGRA 7

Os **esquemas relacionais PUBLICATION e PUBLISHERS** (mostrados abaixo), serão transformados para uma **classe publication composta pela classe publishers**.

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime, **pub_id**: char) pub_id is FK to PUBLISHERS,

PUBLISHERS(pub_id: char, pub_name: varchar, city: varchar, state: char)

```
class publication {
        title_id: string
        title: string
        pubdate: date
        pub_id: publishers
        }
class publishers {
        pub_id: string
        pub_name: string
        city: string
        state: string
        }
```

Por outro lado, os **esquemas relacionais PUBLICATION e PUBLISHERS** modificados (como mostrados abaixo) serão transformados para as definições de **classes publication** abaixo **e publishers** acima. A classe publication terá um **atributo publishers substituindo pub_id e pub_name**, cujo domínio será a própria classe publishers.

PUBLICATION(title_id: tid, title: varchar, pubdate: datetime, **pub_id**: char, **pub_name**: varchar) **pub_id and pub_name are FK to PUBLISHERS**,

PUBLISHERS(pub_id: char, pub_name: varchar, city: varchar, state: char)

class publication {
title_id: string
title: string
pubdate: date
publishers: publishers
}

Resumindo, nessa seção vimos que se uma relação é transformada de um atributo "set", de um atributo composto ou de um relacionamento muitos-para-muitos de um esquema ERE, então ela não será transformada em uma classe. Ao invés disso, ela será transformada para um atributo "set" ou "struct" de uma outra classe. A hierarquia de classe num esquema OO é obtida pela identificação de relacionamento subtipo/supertipo (isto é, hierarquia de especialização/generalização) no esquema ERE. A hierarquia de composição no esquema OO é obtida pela exploração de relacionamentos chave/chave-estrangeira no esquema relacional.

4.3 Adicionando métodos ao esquema OO transformado

As regras de transformação apresentadas na Seção 4.2 consideraram somente aspectos estruturais de um esquema OO. Isso pode ser suficiente se os usuários da API estão limitados a submeter somente comandos de consulta. Se for admitido que os usuários da API submetam comandos de atualização (isto é, inserção, remoção e atualização de dados) no SGBD relacional, então a adição de métodos ao esquema OO transformado pode

se tornar necessário. Nessa seção, discutem-se os métodos que poderiam ser adicionados ao esquema OO transformado.

Em SGBD-OO, cada método consiste de duas partes. A primeira parte é a sua assinatura (signature), ou seja, o nome do método, o tipo ou classe dos argumentos e tipo ou classe do resultado, se existir. A segunda parte é o corpo do método (body), ou seja, o código da implementação do método. Um exemplo de assinatura de um método para a classe book poderia ser "**method increase_price(amount: float)**".

Se o privilégio de atualizar deve ser garantido a usuários da API, métodos apropriados devem ser definidos para viabilizar que usuários especifiquem pedidos de atualização. Por exemplo, se for permitido que novos autores sejam inseridos na classe authors, um método para inserção tal como "**method add_author(a:authors): boolean**" deve ser definido na classe authors. Muito provavelmente, somente a assinatura de tal método precise ser especificada e o corpo será vazio, na medida em que a efetiva atualização não ocorre no nível da API (lembrar que ela tem somente o esquema virtual, sem instância), mas no SGBD local. Em geral um comando do usuário para acesso a um SMBD, é traduzido pela API, para um conjunto funcionalmente equivalente de comandos para os SGBDs locais, por outra peça de software chamada de tradutor de comandos [44].

Algumas vezes, pode ser benéfico para o desempenho geral implementar no nível da API-OO parte de um método. Por exemplo, considere a situação em que um atributo A de uma relação R, tenha sido declarado como "**not null**", significando que toda tupla em R deve ter realmente um valor em A. Pela regra de transformação, A se tornará um atributo da classe C(R). Agora, assuma que um usuário da API-OO submeta um comando para inserir um objeto em C(R) com um valor nulo para A. Uma forma de tratar esse comando é traduzi-lo para um comando relacional equivalente que tentaria inserir a tupla na relação R com o valor nulo para A. Como o valor nulo para A não é permitido, o SGBD relacional retornará uma mensagem dizendo que tal inserção não é permitida. Essa mensagem será então repassada ao usuário da API-OO. Uma outra alternativa para tratar o mesmo comando OO é implementar um corpo simples para o correspondente método, tratando a restrição de valor não nulo no nível da API-OO. Uma vez que essa abordagem não implica em tradução de comandos e não requer interação com os SGBDs locais, espera-se um desempenho melhor

da API-OO. Assim, em geral, se um comando OO pode ser respondido/tratado no nível da API-OO, sem necessidade de acesso aos dados reais nos SGBDs locais, implementando um corpo simples para o correspondente método, pode-se melhorar o desempenho geral da API-OO.

Métodos podem também ser adicionados no esquema OO transformado para mudar visões do esquema. Por exemplo, se em um objeto da classe dependents, como mostrado abaixo, o atributo *age* dos dependentes é freqüentemente necessário, podemos adicionar um método tal como **method age(): integer** na classe dependents. O corpo de age() contém o código que computa a idade a partir da data corrente mantida pelo sistema e da data de nascimento trazida do SGBD local.

```
class dependent {
        au_id: string
        name: string
        sex: string
        birthdate: date
        age: integer
method age(): integer
body age
{ age= idade(current_date, birthdate) }
}
```

4.4 Resumo do Capítulo 4

Nesse capítulo foi apresentado um algoritmo para fazer a transformação de esquema relacional para esquema orientado a objetos. Primeiramente, foram definidas algumas convenções adotadas para facilitar a apresentação do algoritmo e algumas restrições que definem os limites do transformador de esquemas. Mostrou-se o algoritmo do transformador de esquemas, explicando cada uma de suas 7 regras através de exemplos

extraídos do banco de dados pubs2, com algumas modificações para simplificar a explanação. Finalmente discutiu-se como poderiam ser adicionados métodos ao esquema orientado a objetos, muito embora o objetivo, nesse momento, não seja tratar de criação de métodos. No próximo capítulo, apresentaremos uma implementação específica para o transformador de esquemas visto nesse capítulo. Ele mostra a estratégia do projeto e as especificações dos programas para transformar esquemas relacionais do Sybase para esquemas OO do O2.

CAPÍTULO 5 *Implementação de um Transformador de Esquemas*

Esse capítulo apresenta a implementação de um transformador de esquemas para transformar esquemas Sybase (relacional) para esquemas O2 (orientado a objetos). Expõe-se a estratégia do projeto de implementação e a especificação dos programas. Primeiro, na Seção 5.1, faz-se um preâmbulo para a estratégia acima citada, definindo algumas condições e suposições antes de iniciar essa implementação específica. Na Seção 5.2, apresenta-se a estratégia do projeto dando uma visão geral de como ela foi desenvolvida. As especificações dos programas são mostradas na Seção 5.3, descritas numa linguagem do tipo PDL (Program Definition Language). O programa 1 extrai do Sybase somente as informações necessárias para o propósito da transformação, usando um analisador léxico gerado pelo LEX (um utilitário do UNIX). Na realidade, esse programa é um filtro que obtém informações, tais como nomes das relações, atributos, domínio e chaves do dicionário de dados do Sybase. O programa 2 transforma as informações "relacionais" (obtidas do programa 1) para um esquema relacional e, então, transforma esse esquema relacional em um esquema orientado a objetos, usando as regras de transformação discutidas no Capítulo 4. A Seção 5.4 mostra o programa "script" makefile do UNIX para executar o ciclo completo de transformação, começando com informações do dicionário de dados do Sybase e terminando nas classes do esquema do O2.

5.1 Preâmbulo para a Estratégia de Projeto de Implementação

Quando um projetista de banco de dados cria um esquema relacional (isto é, um conjunto de relações associadas), ele primeiro usa a linguagem de definição de dados (DDL) do SQL para definir cada relação (isto é, tabela) no esquema e então compila os comandos. Após a compilação, a definição original não estará mais disponível. Ao invés disso, o SGBD armazena as informações compiladas no dicionário de dados num formato diferente. Por exemplo, no Sybase, para buscar as informações sobre a relação (ou tabela) publishers, pode-se usar o comando "sp_help publishers". O sistema Sybase retornará as informações mostradas na Figura 5.1.1.

Para o propósito desse trabalho, essas informações contêm muitas "impurezas". E essa situação não é muito diferente para outros SGBDs. Assim, iniciam-se as tarefas de transformação de esquemas assumindo que esse tipo de informação pode se tornar disponível e que é necessário "filtrar" essas informações para se obter somente as desejadas.

5.2 Estratégia do Projeto de Implementação

O objetivo da estratégia do projeto de implementação do transformador de esquemas foi ser tão geral e independente quanto possível de qualquer SGBD. Muito embora isso possa parecer contraditório, uma vez que o transformador de esquema comporta-se como um "driver" (e como "driver" ele é necessariamente dependente de um SGBD específico), procurou-se separar, tanto quanto possível, os aspectos que são dependentes de um SGBD local específico daqueles que podem ser generalizados para qualquer SGBD local. Assim, a transformação de esquemas é feita em duas fases por dois diferentes programas. Na primeira fase, fortemente dependente do SGBD relacional local, o programa 1, tendo como entrada um arquivo contendo informações tais como as mostradas na Figura 5.1.1, extrai só as informações necessárias (tais como nomes de relações, atributos, domínios e chaves) e

armazena-os num arquivo intermediário. Por exemplo, o arquivo intermediário para a relação publishers conterá informações tais como as mostradas na Figura 5.2.1.

```
Name                 Owner                Type
-------------------- -------------------- ----------
publishers           dbo                  user table

Data_located_on_segment     When_created
--------------------------- ------------------------
default                     Jun 27 1994  4:04PM

Column_name Type    Length Prec  Scale Nulls Default_name  Rule_name     Identity
----------- ------- ------ ----- ----- ----- ------------ ------------- --------
pub_id      char         4 NULL  NULL     0 NULL         pub_idrule           0
pub_name    varchar     40 NULL  NULL     1 NULL         NULL                 0
city        varchar     20 NULL  NULL     1 NULL         NULL                 0
state       char         2 NULL  NULL     1 NULL         NULL                 0

index_name          index_description                                  index_keys
------------------- -------------------------------------------------- ----------
pubind              clustered, unique located on default               pub_id

(1 row affected)
keytype  object       related_object  object_keys                    related_keys
-------- ------------ --------------- ------------------------------ ------------------------------
primary  publishers   -- none --      pub_id, *, *, *, *, *, *, *    *, *, *, *, *, *, *, *
foreign  publication  publishers      pub_id, *, *, *, *, *, *, *    pub_id, *, *, *, *, *, *, *

(return status = 0)
```

Figura 5.1.1 Informações associadas a relação publishers no Sybase.

```
primary publishers none pub_id
foreign publication publishers pub_id related pub_id
rel publishers
pub_id char NULL NULL NULL pub_idrule
pub_name varchar NULL NULL NULL NULL
city varchar NULL NULL NULL NULL
state char NULL NULL NULL NULL
```

Figura 5.2.1 Saída do programa 1 - Informações necessárias para reconstituir a relação publishers.

Na segunda fase, independente do SGBD local, o programa 2, tendo como entrada o arquivo intermediário acima citado, reconstitui o esquema relacional num formato bem próximo daquele da definição SQL original do banco de dados relacional. Por exemplo, o esquema relacional para a relação publishers poderia ser algo próximo ao mostrado na Figura 5.2.2.

```
Relation[0]: publishers
  pub_id: char
  pub_name: varchar
  city: varchar
  state: char
    primary key[0]: pub_id
```

Figura 5.2.2 Esquema Relacional reconstituído, para a relação publishers

Após isso, o programa 2 efetivamente transforma o esquema relacional obtido em um esquema OO, usando as regras apresentadas no Capítulo 4. A equivalente classe publishers no O2, para a relação publishers é mostrada na Figura 5.2.3.

```
class publishers
type tuple(pub_id: string
           pub_name: string
           city: string
           state: string)
```

Figura 5.2.3 Classe publishers no O2 - transformada do equivalente esquema relacional Sybase

5.3 Especificação dos programas

5.3.1 Programa 1: squeeze.lex

O program 1, da primeira fase acima mencionada, é um programa "script" LEX. O utilitário LEX é um gerador de analisadores léxicos do ambiente UNIX, que gera programas C conforme os comandos no programa "script". Ele é especialmente útil e fácil de usar para gerar programas para pesquisar palavras e/ou reformatar um arquivo texto. Ele possui algumas variáveis predefinidas, que são muito úteis. Duas dessas variáveis são yytext e yyleng, que armazenam, respectivamente, a última palavra pesquisada e o tamanho dessa palavra. Nessa subseção, apresentaremos somente a descrição em termos de PDL (Figura 5.3.1.a), para facilitar o entendimento. O correspondente código fonte LEX para esse programa é mostrado no Apêndice A.

Função **scan** : lê um arquivo texto e retorna a próxima palavra na variável **yytext** e o tamanho da palavra na variável **yyleng**.

```
While is not end of file do
   scan
   case yytext is

      a number:        ignore it

      "index_name" or
      "Object does not":
                       mark end of attributes

      a sequence of "-":
                       if yyleng = 8 then mark begin of attributes
                       else if yyleng = 22 then mark begin of relation'
                       else ignore it

      "primary":       print it,
                        mark begin of primary key

      "foreign":       print it,
                       mark begin of foreign key

      "(return status":
                      mark end of everything

      a sequence of "*":
                       if number of "*" = 0 and if is foreign key then print "related",
                       increment number of "*"

      "NULL" or
      any other sequence of char:
                       if is relation then print "rel ",
                                                print it
                       else if is attributes then print it
                       else if is primary key then print it
                       else if is foreign key then print it
```

Figura 5.3.1.a Especificação do programa 1: squeeze.lex

5.3.2 Programa 2: tsyo2.c

O programa 2 é um programa C (compilado pelo compilador C++) que lê o arquivo intermediário gerado pelo programa 1, reconstitui o esquema relacional num formato bastante próximo da definição SQL original do banco de dados relacional e a partir desse ponto, efetivamente aplica as 7 regras de transformação do algoritmo. A Figura 5.3.2.a e o Apêndice C mostram o esquema relacional pubs2 reconstituído. A dinâmica do processo de transformação de esquemas não é aleatória, em termos de sequência de execução das regras. As regras devem ser executadas obedecendo a uma ordenação e ao mesmo tempo a um agrupamento. Os agrupamentos se devem, ora por questões de similaridade das condições, mas com ações diferentes, ora devido ao caráter eliminatório das condições contidas nas regras.

Passo 1 - A regra número 1 deve ser executada para todas as relações do esquema, porque as regras subsequentes pressupõem que todas as relações têm sua classe correspondente criada por essa regra. Assim, no momento do término da execução da regra 1, o número de classes criadas deve ser igual ao número de relações existentes no esquema relacional.

Passo 2 - As regras número 2, 3 e 4 podem ser executadas nessa ordem e num passo único para cada relação R_i no esquema, isto é, cada relação R_i pode ser verificada se ela satisfaz as condições da regra 2, da regra 3 ou da regra 4, antes de se começar a verificar a relação R_{i+1}.

Passo 3 - As regras número 5, 6 e 7 devem ser executadas nessa ordem e num passo único para cada relação R_i no esquema, isto é, cada relação R_i deve ser verificada se ela satisfaz as condições da regra 5, da regra 6 ou da regra 7, antes de se começar a verificar a relação R_{i+1}.

Pode ser possível reduzir o número de passos necessários. Entretanto, a eficiência do transformador de esquema não é a principal preocupação no estágio atual porque o número de relações num SGBD é tipicamente pequeno e a transformação de um esquema é feita uma única vez.

```
Relation[0]: publishers
 pub_id: char
 pub_name: varchar
 city: varchar
 state: char
  primary key[0]: pub_id

Relation[1]: publication
 title_id: tid
 title: varchar
 pub_id: char
 pubdate: datetime
  primary key[0]: title_id
  foreign name[0]: publishers
      foreign key[0]: pub_id
      related key[0]: pub_id

Relation[11]: book
 title_id: tid
 edition: varchar
 print: varchar
 price: int
  primary key[0]: title_id
  foreign name[0]: publication
      foreign key[0]: title_id
      related key[0]: title_id

Relation[12]: paper
 title_id: tid
 vol.: varcahr
 no.: smallint
  primary key[0]: title_id
  foreign name[0]: publication
      foreign key[0]: title_id
      related key[0]: title_id

Relation[7]: stores
 stor_id: char
 stor_name: varchar
 stor_address: varchar
 city: varchar
 payterms: varchar
  primary key[0]: stor_id

Relation[8]: discounts
 discounttype: varchar
 stor_id: char
 lowqty: smallint
 highqty: smallint
 discount: float
  primary key[0]: discounttype
  primary key[1]: stor_id
  foreign name[0]: stores
      foreign key[0]: stor_id
      related key[0]: stor_id

Relation[3]: titleauthor
 au_id: id
 title_id: tid
 au_ord: tinyint
 royaltyper: int
  primary key[0]: au_id
  primary key[1]: title_id
  foreign name[0]: publication
      foreign key[0]: title_id
      related key[0]: title_id
  foreign name[1]: authors
      foreign key[0]: au_id
      related key[0]: au_id

Relation[4]: roysched
 title_id: tid
 lorange: int
 hirange: int
 royalty: int
  primary key[0]: title_id
  foreign name[0]: publication
      foreign key[0]: title_id
      related key[0]: title_id

Relation[5]: sales
 stor_id: char
 ord_num: varchar
 date: datetime
  primary key[0]: stor_id
  primary key[1]: ord_num
  foreign name[0]: stores
      foreign key[0]: stor_id
      related key[0]: stor_id

Relation[6]: salesdetail
 stor_id: char
 ord_num: varchar
 title_id: tid
 qty: smallint
 discount: float
  primary key[0]: stor_id
  primary key[1]: ord_num
  foreign name[0]: publication
      foreign key[0]: title_id
      related key[0]: title_id
  foreign name[1]: sales
      foreign key[0]: stor_id
      related key[0]: stor_id
      foreign key[1]: ord_num
      related key[1]: ord_num

Relation[2]: authors
 au_id: id
 au_name: varchar
 address: varchar
 city: varchar
 state: char
 country: varchar
 zipcode: char
 phone: char
 birthdate: datetime
 age: int
  primary key[0]: au_id

Relation[9]: au_pix
 au_id: char
 pic: image
 format_type: char
 bytesize: int
 pixwidth_hor: char
 pixwidth_vert: char
  primary key[0]: au_id
  foreign name[0]: authors
      foreign key[0]: au_id
      related key[0]: au_id

Relation[10]: blurbs
 au_id: id
 copy: text
  primary key[0]: au_id
  foreign name[0]: authors
      foreign key[0]: au_id
      related key[0]: au_id

Relation[13]: authors-hobbies
 au_id: id
 hobbies: varcahr
  primary key[0]: au_id
  primary key[1]: hobbies
  foreign name[0]: authors
      foreign key[0]: au_id
      related key[0]: au_id

Relation dependent
 au_id: id
 name: varchar
 sex: char
 birthdate: datetime
  primary key[0]: au_id
  primary key[1]]: name
  foreign name[0]: authors
      foreign key[0]: au_id
      related key[0]: au_id
```

Figura 5.3.2.a Esquema Relacional Pubs2 reconstituído

Novamente, descreve-se esse programa numa linguagem do tipo PDL, principalmente devido à complexidade para se descrever detalhes dos procedimentos em linguagem natural. A Tabela 5.3.2 mostra as convenções utilizadas na descrição desse programa. O fluxo de execução do programa obedece a ordem dos passos acima. Após cada passo, procura-se esclarecer com alguns exemplos extraídos do banco de dados pubs2. Nos exemplos os atributos apresentam-se no estilo *itálico.*

Tabela 5.3.2 Convenções para descrição do transformador de esquema.

Expressão	Indica
R	Relação
R_i	Relação i
$C(R_i)$	Classe transformada a partir da relação R_i
PK	Primary Key (Chave primária)
FK	Foreign Key (Chave estrangeira)
$PK(R_i)$	Primary Key da relação R_i
$FK(R_i)$	Foreign Key da relação R_i
$\#PK(R_i)$	Número de atributos da PK de R_i
$\#att(R_i)$	Número de atributos de R_i

5.3.2.1 Passo 1: Regra 1- Transformar todas as relações em classes

Para cada relação R_i no esquema relacional

Criar uma classe $C(R_i)$ no esquema OO, com todos os atributos de R_i. O tipo de dado de cada atributo de R_i será o domínio do correspondente atributo da classe criada.

No exemplo pubs2, criamos as classes publishers, publication, book, paper, authors, dependent, authors-hobbies, titleauthors, roysched, sales, salesdetail, stores, discounts, au_pix and blurbs, mostradas na Figura 5.3.2.b, correspondentes às

relações da Figura 5.3.2.a. Nesse momento não trataremos das chaves e dos relacionamentos entre relações. Eles serão tratados nas regras subsequentes.

```
class publishers
type tuple(pub_id: string
      pub_name: string
      city: string
      state: string)

class titleauthor
type tuple(au_id: string
      title_id: string
      au_ord: integer
      royaltyper: integer)

class roysched
type tuple(title_id: string
      lorange: integer
      hirange: integer
      royalty: integer)

class sales
type tuple(stor_id: string
      ord_num: string
      date: date)

class salesdetail
type tuple(stor_id: string
      ord_num: string
      title_id: string
      qty: integer
      discount: float)

class publication
type tuple(title_id: string
      title: string
      pub_id: string
      pubdate: date)

class book
type tuple(title_id: string
      edition: string
      print: string
      price: integer)

class paper
type tuple(title_id: string
      vol.: string
      no.: integer)

class stores
type tuple(stor_id: string
      stor_name: string
      stor_address: string
      city: string
      payterms: string)

class discounts
type tuple(discounttype: string
      stor_id: string
      lowqty: integer
      highqty: integer
      discount: float)

class authors
type tuple(au_id: string
      au_name: string
      address: string
      city: string
      state: string
      country: string
      zipcode: string
      phone: string
      birthdate: date
      age: integer)

class au_pix
type tuple(au_id: string
      pic: bitmap
      format_type: string
      bytesize: integer
      pixwidth_hor: string
      pixwidth_vert: string)

class blurbs
type tuple(au_id: string
      copy: string)

class authors-hobbies
type tuple(au_id: string
      hobbies: string)

class dependent
type tuple(au_id: string
      name: string
      sex: string
      birthdate: date)
```

Figura 5.3.2.b Esquema OO O2 para pubs2, logo após somente a regra 1 ser executada

5.3.2.2 Passo 2: Regra 2 - Identificar relacionamentos ISA e relações como parte de outras
Regra 3 - Identificar Atributo Composto
Regra 4 - Identificar Atributo "set"

Para cada relação (vamos dizer, R_0) no esquema relacional, verificar

Se **(1) {FK(R_0) = PK(R_0) e**

FK(R_0) referencia a PK(R_1)} [ver nota] então

se **(2) {FK(R_1) não é subconjunto de PK(R_1) ou**

. **FK(R_1) não referencia a PK(R_0)}** então

. se R_0 tem relacionamentos chave/chave-estrangeira com outras relações então

. . fazer C(R_1) ser uma superclasse de C(R_0) e remover de C(R_0) todos os

. . atributos que aparecem em C(R_1), exceto se o atributo é a PK(R_0).

. else se **R_0 IS_An R_1** então

. **(caso a)** fazer C(R_1) ser una superclasse de C(R_0) e remover de C(R_0) todos os

. atributos que aparecem em C(R_1), exceto se o atributo é a PK(R_0).

senão **(condição 2 é falsa)**

se R_0 tem relacionamentos chave/chave-estrangeira com outras relações então

criar um atributo em C(R_0) chamado de R_1, cujo domínio seja do tipo tupla formado por todos os atributos de R_1 que não estejam na PK(R_1)

Descartar C(R_1).

senão R_1 não tem relacionamentos com outras relações então

se **R_0 é parte da descrição de R_1** então

criar um atributo em C(R_1) chamado de R_0, cujo domínio seja do tipo tupla formado por todos os atributos de R_0 que não estejam na PK(R_0)

Descartar C(R_0).

senão se **(3) {PK(R_0) não consiste de exatamente duas FKs (FK1 e FK2, onde FK1 e FK2 referenciam R_1 e R_2 respectivamente)}** então

se #PK(R_0) = #att(R_0) e

#FK(R_0) +1 = #att(R_0) e

FK(R_0) referencia a PK(R_1) e

R_0 não tem relacionamentos chave/chave-estrangeira com outras relações então

se **R_0 é parte da descrição de** R_1 então

criar um atributo em C(R_1) chamado pelo atributo em R_0, que não esteja em FK(R_0), tendo como domínio um conjunto do domínio do atributo.

Descartar C(R_0).

Nota: A FK (Foreign Key) precisa ser exatamente como a PK (Primary Key) e não somente parte dela. Em outras palavras, se a PK é uma PK composta, a FK precisa ser uma FK composta referenciando a PK composta em R_1

No esquema de banco de dados relacional pubs2 mostrado na Figura 5.3.2.a, os pares R_0 e R_1 que satisfazem a condição (1) {FK(R_0) = PK(R_0) e FK(R_0) referencia a PK(R_1)} e a condição (2) {FK(R_1) não é subconjunto de PK(R_1) ou FK(R_1) não referencia a PK(R_0)} são:

1- relações book como R_0 e publication como R_1,

2- relações paper como R_0 e publication como R_1,

3- relações roysched como R_0 e publication como R_1,

4- relações salesdetail como R_0 e sales como R_1,

5- relações blurbs como R_0 e authors como R_1,

6- relações au_pix como R_0 e authors como R_1

Olhando o diagrama na Figura 3.2.a e os pares R_0 e R_1 acima identificados, os pares de relações book e publication, paper e publication, e salesdetail e sales mantêm um relacionamento ISA e todas as outras relações R_0's são partes de suas respectivas relações R_1's. Portanto, se considerarmos que "book is a publication" e/ou "paper is a

publication", as classes book e paper herdam todas as propriedades da classe publication (Figura 5.3.2.c). Da mesma forma, a classe salesdetail herda todas as propriedades da classe sales, se considerarmos que "salesdetail is a sales". Todos os atributos das subclasses (book, paper e salesdetail), que aparecem nas superclasses (publication e sales) serão removidos, exceto os atributos que sejam iguais às PKs das subclasses.

Como a relação blurbs tem somente um atributo além da PK, a classe blurbs torna-se parte da classe authors através da cópia do atributo *copy* da classe blurbs como um atributo da classe authors (Figura 5.3.2.d). Como a relação au_pix tem mais que um atributo além da PK, a classe au_pix torna-se parte da classe authors, criando-se na classe authors um atributo chamado *au_pix* que tem como domínio a tupla formada por todos os atributos da relação au_pix, exceto pela PK da relação au_pix (Figura 5.3.2.e). Da mesma forma, para a relação roysched como R_0 e a relação publication como R_i (Figura 5.3.2.g), ocorre a mesma situação acima descrita para as relações au_pix e authors.

```
class publication
type tuple(title_id: string
           title: string
           pub_id: string
           pubdate: date)

class book
inherit publication
type tuple(title_id: string
           edition: string
           print: string
           price: integer)

class paper
inherit publication
type tuple(title_id: string
           vol.: string
           no.: integer)

class sales
type tuple(stor_id: string
           ord_num: string
           date: date)

class salesdetail
inherit sales
type tuple(stor_id: string
           ord_num: string
           title_id: string
           qty: integer
           discount: float)
```

Figura 5.3.2.c Herança de classes modelando relacionamentos IS_A.

```
class authors
type tuple(au_id: string
           au_name: string
           address: string
           city: string
           state: string
           country: string
           zipcode: string
           phone: string
           birthdate: date
           age: integer
           copy: string)
```

Figura 5.3.2.d Atributo copy como parte da classe authors

```
class authors
type tuple(au_id: string
      au_name: string
      address: string
      city: string
      state: string
      country: string
      zipcode: string
      phone: string
      birthdate: date
      age: integer
      copy: string
      au_pix: tuple(
            pic: bitmap
            format_type: string
            bytesize: integer
            pixwidth_hor: string
            pixwidth_vert: string))
```

Figura 5.3.2.e Classe au_pix como parte da classe authors

Se inserirmos na relação publication uma FK chamada *title_id* referenciando a relação roysched, como na Figura 5.3.2.f abaixo,então a condição (2) {$FK(R_1)$ não é subconjunto de $PK(R_1)$ ou $FK(R_1)$ não referencia a $PK(R_0)$} do PASSO 2, torna-se falsa.

Assim os pares R_0 e R_1 que satisfazem a condição (1) e não satisfazem a condição (2) são:

1- relações publication como R_0 e roysched como R_1,

2- relações roysched como R_0 e publication como R_1

```
Relation[1]: publication
  title_id: tid
  title: varchar
  pubdate: datetime
  pub_id: char
    primary key[0]: title_id
    foreign name[0]: publishers
          foreign key[0]: pub_id
          related key[0]: pub_id
  foreign name[1]: roysched
       foreign key[0]: title_id
       related key[0]: title_id
```

Figura 5.3.2.f Relação publication modificada.

Nessas condições, parece razoável supor que roysched seja parte de publication. Assim, como a relação roysched tem mais que um atributo além da PK *title_id*, a classe roysched torna-se parte da classe publication, criando-se na classe publication um atributo chamado *roysched* que tem como domínio a tupla formada por todos os atributos da relação roysched, exceto pela PK *title_id* de roysched (Figura 5.3.2.g).

```
class publication
 type tuple(title_id: string
        title: string
        pubdate: date
        pub_id: string
        roysched: tuple(
              lorange: integer
              hirange: integer
              royalty: integer))
```

Figura 5.3.2.g Classe roysched como parte da classe publication.

Continuando nessas condições, se deixarmos na relação roysched somente a PK *title_id* e mais um atributo (*lorange*, por exemplo), removendo todos os outros atributos, como na Figura 5.3.2.h abaixo, então a classe roysched torna-se parte da classe publication copiando-se da classe roysched o atributo *lorange* (que não é a PK) como atributo da classe publication (Figura 5.3.2.i).

```
Relation[4]: roysched
  title_id: tid,
  lorange: int
    primary key: title_id
    foreign key: title_id references titles
```

Figura 5.3.2.h Relação roysched com somente um atributo além da PK *title_id*.

```
class publication
type tuple(title_id: string
           title: string
           pubdate: date
           pub_id: string
           lorange: integer)
```

Figura 5.3.2.i Atributo *lorange* da classe roysched como parte da classe publication

No esquema relacional pubs2 (Figura 5.3.2.a), considerando-se a relação authors-hobbies como R_0 e a relação authors como R_1 a condição (1) {FK(R0) = PK(R0) e FK(R0) referencia a PK(R1)} é satisfeita. Assim, um atributo chamado *hobbies*, do tipo set com domínio do atributo hobbies (string, no caso), será criado na classe authors (Figura 5.3.2.j).

```
class authors
type tuple(au_id: string
           au_name: string
           address: string
           city: string
           state: string
           country: string
           zipcode: string
           phone: string
           birthdate: date
           age: integer
           copy: string
           au_pix: tuple(pic: bitmap
                         format_type: string
                         bytesize: integer
                         pixwidth_hor: string
                         pixwidth_vert: string)
           hobbies: set(string))
```

Figura 5.3.2.j Atributo do tipo "set"

Um outro exemplo, agora com PK composta, pode ser visto se removermos todos os atributos, exceto pela PK, da relação discounts (Figura 5.3.2.k). Considerando-se a relação discounts como R_0 e a relação stores como R_1, cria-se um atributo na classe stores, chamado *discounttype*, do tipo **set** com domínio do atributo discounttype (string, no caso) (Figura 5.3.2.l).

```
Relation[8]: discounts
 discounttype: varchar
 stor_id: char
  primary key[0]: discounttype
  primary key[1]: stor_id
  foreign name[0]: stores
      foreign key[0]: stor_id
      related key[0]: stor_id
```

Figura 5.3.2.k Relação discounts com uma PK composta.

```
class stores
type tuple(stor_id: string
           stor_name: string
           stor_address: string
           city: string
           payterms: string
           discounttype: set(string))
```

Figura 5.3.2.I Atributo do tipo set criado de um atributo PK composto.

5.3.2.3 Passo 3: Regra 5 - Transformar relacionamento unário muito-pra-muito. Regra 6 - Transformar relacionamento binário muito-pra-muito. Regra 7 - Expandir hierarquia de composição.

Se **(1) {PK(R_0) consiste de exatamente duas FKs (FK1 e FK2) e FK1 e FK2 referenciam PK(R_1)}** então

se R_0 não tem nenhum outro atributo então

. criar dois atributos conjunto (set) em $C(R_1)$, tendo como domínio a própria $C(R_1)$.

. O nome do primeiro atributo será o nome de R_0 e o nome do segundo atributo

. será o used_as_R_0.

senão (R_0 tem outros atributos)

. criar dois atributos conjunto em $C(R_1)$, tendo como domínio a tupla formada

. por $C(R_1)$ e todos os atributos, exceto pela PK(R_0). O nome do primeiro

. atributo será o nome de R_0 e o nome do segundo atributo será used_as_R_0.

Descartar $C(R_0)$.

senão

se **(2) {PK(R_0) consiste de exatamente duas FKs (FK1 e FK2) e FK1 e FK2 referenciam R_1 e R_2 respectivamente}** então

se R_0 não tem nenhum outro atributo então

. criar um atributo conjunto em $C(R_1)$, chamado por R_2, com $C(R_2)$ como seu domínio.

. criar um atributo conjunto em $C(R_2)$, chamado por R_1, com $C(R_1)$ como seu domínio.

else (R_0 tem outros atributos)

. criar um atributo conjunto em $C(R_1)$, chamado por R_0, tendo como domínio a tupla

. formada por $C(R_2)$ e todos os atributos, exceto pela PK(R_0).

. criar um atributo conjunto em $C(R_2)$, chamado por R_0, tendo como domínio a tupla

. formada por $C(R_1)$ e todos os atributos , exceto pela PK(R_0).

Descartar $C(R_0)$.

senão

se **(3) {R_0 não satisfaz as condições da regra 5 nem da regra 6 e**

R_0 tem FKi referenciando R_1, onde FKi não é um subconjunto de PK(R_0)} então

se FKi tem mais que um atributo então

. Remover os atributos de C(R_0), que estejam também em FKi e

. não sejam usados por outras relações

. Criar um atributo em C(R_0), chamado por R_1, tendo como domínio C(R_1).

senão (FKi tem somente um atributo)

Mudar para C(R_1) o domínio do atributo relacionado com FKi.

Olhando o diagrama ERE do banco de dados **course** mostrado na Figura 3.1.4.a e o esquema relacional associado (na Figura 5.3.2.m), e considerando-se a relação prerequisite como R_0 e a relação course como R_1, a primeira condição acima é satisfeita.

```
Relation[0]: course
course_no: char
title: char
        primary key[0]: course_no

Relation[1]: prerequisite
is_prerequisite: char
has_prerequisite: char
        primary key[0]: is_prerequisite
        primary key[1]: has_prerequisite
        foreign name[0]: course
            foreign key[0]: is_prerequisite
            related key[0]: course_no
            foreign key[1]: has_prerequisite
            related key[1]: course_no
```

Figura 5.3.2.m Esquema relacional course

Assim, cria-se dois atributos **set** na classe course, tendo como domínio a própria classe course. O nome do primeiro atributo seria *prerequisite* e o nome do segundo seria *used_as_prerequisite* (Figura 5.3.2.n).

```
class course
type tuple(course_no: string
           title: string
           prerequisite: set(course)
           used_as_prerequisite: set(course))
```

Figura 5.3.2.n Classe course transformada das relações course e prerequisite

Inserindo-se pelo menos mais um atributo (*semester_to_take*, por exemplo), na relação prerequisite (Figura 5.3.2.o abaixo), então dois atributos **set** do tipo tupla como na Figura 5.3.2.p, serão criados.

```
Relation[1]: prerequisite
is_prerequisite: char
has_prerequisite: char
semester_to_take: char
      primary key[0]: is_prerequisite
      primary key[1]: has_prerequisite
      foreign name[0]: course
          foreign key[0]: is_prerequisite
          related key[0]: course_no
          foreign key[1]: has_prerequisite
          related key[1]: course_no
```

Figura 5.3.2.o Relação prerequisite com mais um atributo, além da PK.

```
class course
type tuple(course_no: string
           title: string
           prerequisite: set(tuple(
                      course: course
                      semester_to_take: string))
           used_as_prerequisite: set(tuple(
                      course: course
                      semester_to_take: string)))
```

Figura 5.3.2.p Classe course transformada das relações course e prerequisite modificada.

Voltando ao exemplo pubs2, se considerarmos a relação titleauthor como R_0, a relação authors como R_1 e a relação publication como R_2, a condição **(2) {PK(R_0) consiste de exatamente duas FKs (FK1 e FK2) e FK1 e FK2 referenciam R_1 e R_2 respectivamente}** será verdadeira. Dessa forma, como visto na Figura 5.3.2.q1, será criado um atributo **set** na classe authors, chamado *titleauthor*, tendo como domínio a tupla formada pela classe publication e todos os atributos, exceto pela PK (*au_id* e *title_id*), da relação titleauthor. Será criado também, um atributo **set** na classe publication, chamado *titleauthor*, tendo como domínio a tupla formada pela classe authors e todos os atributos, exceto pela PK (*au_id* e *title_id*), da relação titleauthor (Figura 5.3.2.q2).

```
class authors
type tuple(au_id: string
        au_name: string
        address: string
        city: string
        state: string
        country: string
        zipcode: string
        phone: string
        copy: string
        au_pix: tuple(
                pic: bitmap
                format_type: string
                bytesize: integer
                pixwidth_hor: string
                pixwidth_vert: string)
        hobbies: set(string)
        titleauthor: set(tuple(
                publication: publication
                au_ord: integer
                royaltyper: integer)))
```

Figura 5.3.2.q1 Classe authors com relacionamento binário muito-pra-muito

```
class publication
type tuple(title_id: string
        title: string
        pubdate: date
        pub_id: string
        roysched: tuple(
                lorange: integer
                hirange: integer
                royalty: integer)
        titleauthor: set(tuple(
                authors: authors
                au_ord: integer
                royaltyper: integer)))
```

Figura 5.3.2.q2 Classe publication com relacionamento binário muito-pra-muito.

Agora, removendo-se todos os atributos, exceto pela PK (*au_id* e *title_id*), da relação titleauthor, a condição (2) do PASSO 3 acima continua verdadeira. Porém a transformação é feita da seguinte forma: cria-se um atributo **set** na classe authors, chamado *publication*, tendo como domínio a própria classe publication e cria-se também, um atributo **set** na classe publication, chamado *authors*, tendo como domínio a própria classe authors.

Em pubs2 da Figura 5.3.2.a, considerando-se a relação publication como R_0 e a relação publishers como R_1, a condição (3) {**R_0 não satisfaz as condições da regra 5 nem da regra 6 e R_0 tem FKi referenciando R_1, onde FKi não é um subconjunto de PK(R_0)**} será verdadeira. Assim, muda-se o domínio atual de *pub_id* para publishers (Figura 5.3.2.r).

```
class publication
type tuple(title_id: string
        title: string
        pubdate: date
        pub_id: publishers
        roysched: tuple(
                lorange: integer
                hirange: integer
                royalty: integer)
        titleauthor: set(tuple(
                authors: authors
                au_ord: integer
                royaltyper: integer)))
```

Figura 5.3.2.r Classe publication composta pela classe publishers.

Se inserirmos um atributo (pub_name, por exemplo) na relação publication, para formar uma chave estrangeira composta FK, referenciando a PK de publishers (Figura 5.3.2.s), o procedimento para transformar é como segue: 1- remover os atributos *pub_id* e *pub_name* da classe publication, uma vez que eles não são usados por outras relações e 2- criar um atributo na classe publication, chamado *publishers*, tendo como domínio a própria classe publishers (Figura 5.3.2.t).

```
Relation[1]: publication
  title_id: tid
  title: varchar
  pubdate: datetime
  pub_id: char
  pub_name: varchar
    primary key[0]: title_id
    foreign name[0]: publishers
        foreign key[0]: pub_id
        related key[0]: pub_id
        foreign key[1]: pub_name
        related key[1]: pub_name
    foreign name[1]: roysched
        foreign key[0]: title_id
        related key[0]: title_id
```

Figura 5.3.2.s Relação publication com uma FK composta para publishers.

```
class publication
type tuple(title_id: string
        title: string
        pubdate: date
        publishers: publishers
        roysched: tuple(
                lorange: integer
                hirange: integer
                royalty: integer)
        titleauthor: set(tuple(
                authors: authors
                au_ord: integer
                royaltyper: integer)))
```

Figura 5.3.2.t Classe publication com atributo publishers substituindo pub_id e pub_name

A Figura 5.3.2.u e o Apêndice D mostram o esquema O2 como resultado da transformação de esquema para o banco de dados pubs2.

```
class publishers
  type tuple(pub_id: string
          pub_name: string
          city: string
          state: string)

class publication
type tuple(title_id: string
title: string
pubdate: date
pub_id: publishers
roysched: tuple(
lorange: integer
hirange: integer
royalty: integer)
titleauthor: set(tuple(
              authors: authors
              au_ord: integer
              royaltyper: integer)))

class book
inherit publication
type tuple(title_id: string
            edition: string
            print: string
            price: integer)

class paper
inherit publication
type tuple(title_id: string
           vol.: string
            no.: integer)

class authors
type tuple(au_id: string
au_name: string
address: string
city: string
state: string
country: string
zipcode: string
phone: string
copy: string
au_pix: tuple(
          pic: bitmap
          format_type: string
         bytesize: integer
           pixwidth_hor: string
           pixwidth_vert: string)
hobbies: set(string)
titleauthor: set(tuple(
              publication: publication
               au_ord: integer
               royaltyper: integer)))

class dependent
type tuple(au_id: string
       name: string
       sex: string
       birthdate: date)

class sales
  type tuple(stor_id: string
          ord_num: string
          date: date)

class salesdetail
  inherit sales
  type tuple(stor_id: string
          ord_num: string
          title_id: string
          qty: integer
          discount: float)

class stores
  type tuple(stor_id: string
          stor_name: string
          stor_address: string
          city: string
          payterms: string)

class discounts
  type tuple(discounttype: string
          stor_id: string
          lowqty: integer
          highqty: integer
          discount: float)
```

Figura 5.3.2.u Esquema O2 para pubs2

5.4 O programa "script" makefile

Esse é um programa makefile (do UNIX) para executar um ciclo completo de transformação de esquema, começando com a extração de informações do dicionário de dados do Sybase e terminando com o esquema orientado a objetos do O2.

```
#*************************************************************************
# Execute tsyo2.o having as input the file named inphase1 and generate the file named
# o2class.

o2cl:           ../tsyo2.o inphase1
                ../tsyo2.o
                more o2class

# Compile schema transformer program named tsyo2.c and generate the object program
# named tsyo2.o.
../tsyo2.o:     ../tsyo2.c
                gcc -g ../tsyo2.c -o ../tsyo2.o

# Squeeze.o "squeeze" the input file named sp_help_out and generate the intermediate
# file named inphase1.
inphase1:       ../squeeze.o sp_help_out
                ../squeeze.o < sp_help_out
                cat outpkey outfkey outclass > inphase1

# Execute the UNIX LEX (Lexical analyser generator) to generate lex.yy.c program and then
# compile it to generate the object program squeeze.o.
../squeeze.o:
                lex ../squeeze.lex
                gcc lex.yy.c -ll -o ../squeeze.o

# Execute a Sybase isql utility having as input file sp_help_in and as output sp_help_out.
sp_help_out:
                source sp_help.bat
#*************************************************************************
```

5.5 O arquivo readme

Esse arquivo-texto contém uma descrição sucinta do conteúdo e do propósito de cada um dos arquivos envolvidos na tarefa de executar a transformação de esquemas do Sybase para os correspondentes esquemas do O2. Ele descreve também os relacionamentos entre os arquivos.

Arquivo 1: **sp_help.bat**

Conteúdo:

```
isql -Uloginname -Ppassword < sp_help_in > sp_help_out
```

Esse é um arquivo de comando shell do UNIX. Ele executa o utilitário isql do Sybase, que tendo como entrada o arquivo chamado *sp_help_in*, gera um outro arquivo chamado *sp_help_out*. O objetivo desse arquivo de comando único, é colocar o isql como um servidor para executar os comandos contidos no arquivo de entrada. Mais detalhes sobre o utilitário isql ver SYBASE SQL Server - Utility Programs for UNIX [38].

Arquivo 2: **sp_help_in**

Conteúdo:

```
use pubs2
go
sp_help publishers
go
sp_help publication
go
.
.
.
```

Esse é o arquivo de entrada para sp_help_bat (que executa o utilitário isql). Ele contém uma lista de comandos isql e seu objetivo é capturar informações relacionais do dicionário de dados do Sybase. O primeiro comando é **use dbname**, onde dbname é o nome do banco de dados no qual se deseja trabalhar. O segundo comando é **go**, para executar a operação. Os comandos seguintes são uma seqüência de **sp_help relação** e **go**, onde relação é o nome da relação que se deseja extrair do banco de dados. Mais detalhes sobre os comandos use, go, sp_help e outros comandos relacionados ver Sybase SQL Server - Reference Manual [39].

Arquivo 3: **sp_help_out**

Esse arquivo é a saída gerada por sp_help_bat (que executa o utilitário isql). Ele contém informações do banco de dados especificado, extraídas do dicionário de dados do Sybase. Essas informações foram extraídas através dos comandos sp_help. O Apêndice B contém um exemplo de tais informações extraídas do dicionário de dados do Sybase.

Arquivo 4: **squeeze.lex**

Esse arquivo contém um programa fonte LEX (ver Apêndice A) para filtrar somente as informações que são relevantes para o propósito de transformação de esquemas. Ele gera um programa fonte C chamado squeeze.c. O correspondente programa executável é squeeze.o.

Arquivo de entrada: sp_help_out
Arquivo de saída: inphase1

Arquivo 5: **tsyo2.c**

Esse arquivo contém um programa fonte C para fazer a transformação de esquemas Sybase para esquemas O2. O correspondente programa executável é tsyo2.o.

Arquivo de entrada: Inphase1
Arquivos de saída: o2class, syrelations, syo2debug, syo2error.

Arquivo 6: **inphase1**

Gerado por squeeze.o e é a entrada para tsyo2.o. Contém as informações necessárias (ou possíveis) filtradas do dicionário de dados.

Arquivo 7: **syrelations**

Gerado por tsyo2.o. Contém todas as relações e informações de chaves, equivalentes ao esquema relacional original, recriado a partir das informações contidas em inphase1.

Arquivo 8: **o2class**

Gerado por tsyo2.o. Contém as classes do esquema O2, equivalentes às relações do esquema Sybase.

Arquivo 9: **syo2debug**

Gerado por tsyo2.o. Contém mensagens que podem ser úteis para propósitos de depuração.

Arquivo 10: **syo2error**

Gerado por tsyo2.o. Contém mensagens de erros geradas em condições de exceção.

Arquivo 11: **makefile**

Contém uma seqüência de comandos make para executar o transformador de esquemas e compilar (se necessário) os programas sp_help_bat, squeeze.lex e tsyo2.c.

5.6 Resumo do Capítulo 5

Esse capítulo apresentou a implementação de um algoritmo para fazer a transformação de esquemas Sybase (relacional) para esquemas O2 (orientado a objetos).

Foram apresentadas algumas condições e suposições limitando o escopo dessa implementação específica. Apresentou-se a estratégia do projeto dando uma visão geral de como ela foi desenvolvida. A especificação do programa 1 mostra como se extrai do Sybase somente as informações necessárias para o propósito da transformação, usando um analisador léxico gerado pelo LEX (um utilitário do UNIX). A especificação do programa 2 mostra como se transformam as informações "relacionais" (obtidas do programa 1) em um esquema relacional e como esse esquema relacional é transformado em um esquema orientado a objetos do O2, usando as regras de transformação discutidas no Capítulo 4. Foi apresentado também um programa "script" makefile do UNIX para executar o ciclo completo de transformação, começando com informações do dicionário de dados do Sybase e terminando nas classes do esquema do O2. No próximo capítulo são feitas algumas considerações sobre o trabalho e apresentadas algumas áreas que devem ser pesquisadas no futuro.

CAPÍTULO 6

Conclusões & Pesquisas Futuras

6.1 Conclusão relacionada com a API Orientada a Objetos

As dificuldades que caracterizam o desenvolvimento de software usando sistemas de bancos de dados, em ambientes heterogêneos, causam impactos diretamente na produtividade do desenvolvimento e na qualidade das aplicações. Essas dificuldades se devem principalmente à diversidade de modelos de dados, à falta de uma linguagem de consulta padrão, estável e eficiente e à consequente proliferação de linguagens e dialetos. A comunidade de banco de dados tem procurado apresentar soluções para as insuficiências dos SGBDs relacionais em termos de modelo de dados, funcionalidade e expressividade de suas linguagens de consulta para aplicações avançadas, enquanto uma cultura totalmente orientada a objetos não pode ser assimilada pelas organizações. Nesse sentido, considerando que as organizações continuarão a usar os SGBDs relacionais por razões históricas, econômicas e práticas, um enfoque mais pragmático seria melhorar o processo de especificação de consulta e suavemente preparar a migração para uma abordagem totalmente orientada a objetos.

Uma solução, vislumbrada por algumas pesquisas [22,30,44], é construir uma **API de banco de dados, baseado num único modelo de dados (com um esquema global) e numa única linguagem de consulta global**, como uma interface única para acesso a vários SGBDs locais existentes. Cada SGBD tem sua própria linguagem de consulta (query) e seu próprio modelo de dados que descreve como os dados estão organizados nesse SGBD. Esse conjunto incluindo a API e os SGBDs locais é chamado de um Sistema de Múltiplos Bancos de Dados (SMBD). Um SMBD pode ser homogêneo ou heterogêneo, significando respectivamente que os SGBDs locais, todos suportam o mesmo modelo de dados (relacional, por exemplo) ou alguns deles suportam diferentes modelos de dados (relacional, hierárquico e orientado a objetos, por exemplo). Conforme visto em capítulos anteriores, quando um programa de aplicação usa um comando de consulta (tal como select do SQL) para recuperar informações, a API decompõe e traduz esse comando em comandos de consultas para os específicos SGBDs locais onde aquelas informações estão armazenadas. Essas tarefas de decomposição e tradução de comandos de consulta podem ser levadas avante em tempo de execução, dinamicamente, mas para isso, antes, os esquemas dos SGBDs locais precisam ser transformados e integrados num esquema global, tratado no nível da API de banco de dados. Nesse contexto, os componentes que fazem **decomposição de comandos** de consulta [5, 10], **tradução de comandos** de consulta [25, 44], **transformação de esquemas** [19, 22, 31, 43, 44, 45], **integração de esquemas** [4, 21, 40] e também, **otimização de comando global** [5, 9, 28] são críticos e de fundamental importância em SMBD Heterogêneos. Todos esses componentes, na realidade, devem trabalhar de forma cooperativa para processar os comandos de consulta nos SMBD Heterogêneos. A definição das características de uma API, em termos de modelo de dados e de linguagem de consulta, deve considerar as tendências de mercado na utilização de SGBDs, iniciativas de padronização de SGBDs e tecnologias emergentes na área de banco de dados.

Esse trabalho está inserido no contexto de um projeto de pesquisa, que pretende construir uma **API orientada a objetos para acesso a múltiplos bancos de dados heterogêneos**. A API é baseada num único modelo de dados orientado a objetos e

numa única linguagem de consulta global, também orientada a objetos para acesso a diferentes SGBDs locais, os quais suportam diferentes modelos de dados. Essa iniciativa procura: (1) diminuir a "distância" entre a forma natural com que o usuário idealiza suas consultas e as linguagens de consultas, do tipo SQL e (2) criar uma interface padrão e uniforme para acesso a todos os SGBDs que sejam aderentes a essa interface padrão. Esses dois benefícios melhorarão a produtividade no desenvolvimento e a qualidade das aplicações no desenvolvimento de aplicações de banco de dados, uma vez que eles podem evitar a redundância de esforços para várias plataformas de bancos de dados.

6.2 Conclusão relacionada com o Transformador de esquemas

Apresentamos um **algoritmo para transformar esquemas relacionais para esquemas orientados a objetos** tal como proposto em [31], como um passo necessário antes de se executar as tarefas de processamento de comandos de consulta. O algoritmo é uma alternativa a várias outras propostas [22, 43, 44] que consideramos insuficientes ou ineficientes em alguns aspectos. A proposta de Keim, Kriegel e Miethsam [22], não considera a construção de hierarquia de classes no esquema orientado a objetos resultante, enquanto o algoritmo de Yan e Ling [43] não considera criação de conjuntos de atributos (set) para simplificar o esquema orientado a objetos resultante. Muito embora o algoritmo de Yu e outros [44] possa gerar esquemas tipicamente orientados a objetos, contendo hierarquia de classes, hierarquia de composições e de conjuntos atributos, ele gera um número desnecessariamente grande de classes. Além do mais, nenhum desses algoritmos foi implementado.

O algoritmo proposto por Meng, Kamada e Chang em [31] foi implementado para **transformar esquemas relacionais do Sybase para esquemas orientados a objetos do O2** e pode gerar esquemas tipicamente orientados a objetos.

Resumindo o algoritmo, vimos que se uma relação é transformada de um atributo "set", de um atributo composto ou de um relacionamento muitos-para-muitos de um esquema ERE, então ela não será transformada em uma classe. Ao invés disso, ela será transformada em um atributo complexo do tipo "set" ou "struct" de uma outra classe. A hierarquia de classe num esquema OO é obtida pela identificação de relacionamento subtipo/supertipo (isto é, hierarquia de especialização/generalização) no esquema ERE. A hierarquia de composição no esquema OO é obtida pela exploração de relacionamentos chave/chave-estrangeira no esquema relacional. Assim, os esquemas orientados a objetos resultantes podem conter hierarquias de classes, hierarquias de composições, hierarquias de tuplas e conjuntos de atributos. Como consequência dessa agregação de atributos complexos, o número de classes no esquema OO transformado é relativamente pequeno se comparado com o número de relações no esquema relacional correspondente. O algoritmo constitui-se de sete regras de transformação que são quase que totalmente automatizadas, necessitando de muito pouca interação com o usuário. Não foram efetuados testes exaustivos no transformador de esquemas, mas acredita-se que, em termos de qualidade, os testes foram bastante significativos. A gama de testes aplicada produziu resultados que estavam conforme com as nossas expectativas.

O projeto de implementação do algoritmo apoiou-se em princípios básicos de desenvolvimento, principalmente em termos de modularidade coesão e acoplamento, conforme preconizada pela engenharia de software. A implementação foi bastante útil no refinamento do algoritmo, uma vez que possibilitou a captura de aspectos que antes não haviam sido observados. Um exemplo de captura de tais aspectos foi a caracterização das diferenças entre os diversos casos da Regra 3 que trata de relações transformadas de atributos compostos. Embora nossa implementação, apresentada no Capítulo 5, considere dois SGBDs comerciais específicos, o algoritmo, apresentado no Capítulo 4, é genérico suficiente para ser aplicado em outros SGBDs relacionais e orientados a objetos. Além disso, ao adotar-se a mesma estratégia de projeto de implementação aqui utilizada, isolando-se as tarefas de obtenção de informações relacionais de um dicionário de dados específico das tarefas do processo de transformação de esquemas, as modificações

necessárias no transformador de esquemas (programa 2 no Capítulo 5) se reduzem a ajustes no formato dos esquemas. A estratégia de considerar o modelo de dados do O2 como nosso modelo de dados global, facilita nosso trabalho e reduz o tempo na obtenção dos primeiros resultados reais, na medida em que, tanto o Sybase como o O2 podem atuar como os SGBDs locais e é necessário somente transformar esquemas do Sybase para esquemas do O2.

6.3 Pesquisas Futuras

Há duas áreas que precisam ser melhor investigadas. Uma é a **criação de métodos no esquema orientado a objetos**, especialmente quando admite-se que o usuário efetue comandos de atualização no nível da API de banco de dados e a outra área é **melhorar a usabilidade das regras de transformação**, do ponto de vista do usuário do transformador de esquemas.

6.3.1 Criação de Métodos

Uma vez que o modelo de dados relacional não tem o conceito equivalente a métodos no modelo de dados orientado a objetos, o algoritmo de transformação preocupou-se somente com os aspectos de relações, atributos, chaves e relacionamentos. Além disso, como em um sistema de múltiplos bancos de dados o enfoque principal é recuperação de informação e os SGBDs locais devem ter plena autonomia em termos de novas aplicações e administração de banco de dados, a criação de métodos não é tão importante nesse momento. Claro que, quando um sistema de múltiplos bancos de dados **admite transações de inserção e atualização** de informações, as tarefas de criação de métodos são extremamente necessárias. Planejamos continuar a discussão iniciada no Capítulo 4, investigando como **automatizar ou semi-automatizar o processo de criação de métodos**.

6.3.2 Usabilidade das Regras de transformação

A outra área é **melhorar a usabilidade das regras de transformação**. Nossas regras assumem que as relações são bem definidas com informações completas sobre chaves estrangeiras e chaves primárias. Enquanto isso pode ser verdade para esquemas criados por especialistas em bancos de dados, pode não ser verdade para esquemas criados por não especialistas. No caso em que nem todas as informações esperadas do SGBD relacional estão disponíveis, o transformador de esquemas faz perguntas ao usuário. As perguntas são do tipo "Todo livro é também uma publicação?" (nesse caso, procurando detectar um relacionamento subclasse/superclasse entre a classe livro e a classe publicação) ou "Hobbies é parte da descrição de authors?" (nesse caso, procurando verificar se a classe hobbies pode ser transformada em um atributo da classe authors. Embora necessite apenas um "sim" ou um "não" para responder a perguntas desse tipo, até que ponto essas perguntas são inteligíveis ou não suscitam dúvidas no usuário? Uma resposta "sim" quando deveria ser "não" (ou vice-versa) gerará um esquema OO completamente diferente do esperado. Em situações como essa, em que nem todas as informações esperadas do SGBD relacional estão disponíveis, **como interagir com o usuário para criar um bom esquema orientado a objetos**, claramente necessita ser mais pesquisado.

Referências

(1) F. Bancilhon, S. Cluet and C. Delobel - A Query Language for O2. Em "Building an Object-Oriented Database System - The Story of O2", Morgan Kaufmann Publishers, San Mateo, California, pp. 234-255, 1994.

(2) J. Banerjee, W. Kim and K-S. Kim - Queries in Object-Oriented Databases. IEEE 4th Int'l Conference on Data Engineering, pp.31-38, Fevereiro 1988.

(3) E. Bertino, M. Negri, G. Pelagatti and L. Sbatella - Object-Oriented Query Languages: The Notion and Issues. IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No.3, pp. 223-237, Junho 1992.

(4) Y-H. Chang, A. Kamada, W. Meng - Using OODM for Schema Integration in a Heterogeneous Database Environment - Tec. report, SUNY at Binghamton, Outubro 1994.

(5) A. Chen - Outerjoin Optimization in Multidatabase System. 2nd Int'l Symp. on Distributed and Parallel Database Systems, pp. 211-218, 1990.

(6) P. Chen - The Entity_Relationship Model - Toward a Unified View of Data. ACM TODS, Março 1976.

(7) C-W. Chung - DATAPLEX: An Access to Heterogeneous Distributed Databases. CACM, Janeiro 1990.

(8) E. Codd - A Relational Model for Large Shared Data Banks. CACM. Junho 1970.

(9) U. Dayal - Processing Queries over Generalization Hierarchies in a Multidatabase System. VLDB Conf., 1983.

(10) U. Dayal and H-Y. Hwang - View Definition and Generalization for Database Integration in a Multidatabase System. IEEE TSE, pp. 628-644, 1984.

(11) O. Deux, et. al. - The Story of O2. IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 1, pp. 91-108, Março 1990.

(12) R. Elmasri and S. B. Navathe - Enhanced Entity-Relationship (EER) Model. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(13) R. Elmasri and S. B. Navathe - Data Modeling Using the Entity-Relationship Model. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(14) R. Elmasri and S. B. Navathe - The Relational Data Model and Relational Algebra. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(15) R. Elmasri and S. B. Navathe - Functional Dependencies and Normalization. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(16) R. Elmasri and S. B. Navathe - Multivalued Dependencies and Fourth Normal Form. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(17) R. Elmasri and S. B. Navathe - Object-Oriented Databases. Em "Fundamentals of Database Systems", The Benjamin/Cummings Publishing Company, Inc., 1994.

(18) D. Hsiao and M. Kamel - Heterogeneous Databases: Proliferations, Issues and Solutions. IEEE Trans. Knowledge and Data Engineering, vol. 1, no. 1, Março 1989.

(19) H-Y. Hwang and U. Dayal - Using the Entity-Relationship Model for Implementing Multi-Model Database Systems. 2nd Int'l Conference on E-R Approach, Washington, D.C., 1981.

(20) ISO/IEC - Database Language SQL - ISO/IEC 9075:1992.

(21) M. Kaul, K. Deosten and E. J. Neuhold - ViewSystem: Integrating Heterogeneous Information Bases by Object-Oriented Views. pp. 2-10, 1990.

(22) D. Keim, H-P. Kriegel and A. Miethsam - Object-Oriented Querying of Existing Relational Databases. 4th Int'l Conference Database and Expert System Applications - DEXA'93, pp. 325-336, Prague, Setembro 1993.

(23) W. Kim - Introduction to Object -Oriented Databases. The MIT Press, 1990.

(24) W. Kim - A Model of Queries for Object-Oriented Database Systems. Proc. of the International Conference on Very Large Databases (VLDB), Amsterdam, Holland, Agosto 1989.

(25) P. Larson - SQL-GATE: Providing SQL Access to Network and Hierarchical Databases. Demonstration in SIGMOD'90, Atlantic City, Maio 1990.

(26) W. Litwin - MALPHA: A Relational Multidatabase Manipulation Language. IEEE Conf. on Data Engoneering, pp. 86-93, 1984.

(27) W. Litwin, L. Mark and N. Roussopoulis - Interoperability of Multiple Autonomous

Databases. ACM Computing Surveys, Vol. 22, no. 3, pp. 267-293, Setembro 1990.

(28) H. Lu, B. Ooi and C. Goh - On Global Multidatabase Query Optimization. SIGMOD Record, Dezembro1992.

(29) J. Melton (Editor) - Database Language SQL (SQL3) - ISO/ANSI Working Draft, X3H2-92-055 DBL CNB-003, Julho 1992.

(30) W. Meng and C. Yu - Query Processing in Multidatabase Systems. Em "Modern Database Systems: The Object Model, Interoperability and Beyond", edited by W. Kim, Addison-Wesley / ACM Press, pp. 551-572, 1994.

(31) W. Meng, A. Kamada and Y-H. Chang - Transformation of Relational Schema to Object-Oriented Schema. 19th Annual Intl. Computer Software and Applications Conference (COMPSAC'95), Dallas, Texas, U.S.A., Agosto 1995.

(32) The Object Database Standard: ODMG-93. Edited by R. G. G. Cattell, Morgan Kaufmann, 1993.

(33) A. Rafii, et al. - Multidatabase Management in Pegasus. 1st Int'l Workshop on Interoperability in Multidatabase Systems, Japan, Abril 1991.

(34) A. Sheth and J. Larson - Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases. ACM Computing Surveys, vol. 22, no. 3, pp. 183-236, Setembro 1990.

(35) S. Spaccapietra and C. Parent - ERC+: an object based entity relationship approach. Em "Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development", Peri Loucopoulos and Roberto Zicari Eds. John Wiley, 1992.

(36) S. Spaccapietra - Object-Orientations and Conceptual Modeling. Tec. Report, Ecole Polytechnique Federale.

(37) Sybase, Inc. - Sybase SQL Server Transact-SQL User's Guide. 1994.

(38) Sybase, Inc. - Sybase SQL Server - Utility Programs for UNIX. 1994.

(39) Sybase, Inc. - Sybase SQL Server - Reference Manual: Vol I and II - System Procedures and Catalog Stored Procedures. 1994.

(40) B. Szejdo, M. Rusinkiewics and D. Embley - An Approach to Schema Integration and Query Formulation in Federated Database Systems. IEEE Data Engineering Conf. 1987.

(41) T. J. Teorey, D. Yang and J.P. Fry - A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. ACM Computing Surveys, vol. 18, No. 2, pp. 197-222, Junho 1986.

(42) G. Thomas, et al. - Heterogeneous Distributed Database Systems for Production Use. ACM Computing Surveys, vol. 22, no. 3, pp. 237-266, Setembro1990.

(43) L. Yan and T. Ling - Translating Relational Schema with Constraints into OODB schema. In Interoperable Database Systems, edited by D.Hsiao et. al., Elsevier Science, 1993.

(44) C. Yu, Y. Zhang, W. Meng, W. Kim, G. Wang, T. Pham and S. Dao - Translation of Object-Oriented Queries to Relational Queries. 11th IEEE Conference on Data Engineering, Taipei, Taiwan, Março 1995.

(45) C. Zaniolo - Design of Relational Views over Network Schemas. ACM SIGMOD, 1979.

Apêndice A - Arquivo squeeze.lex

Filtra informações do dicionário de dados, tal como se encontra no Apêndice B, para obter somente as informações necessárias.

```
                        typedef enum {false, true} boolean;
%%
                        boolean isattributes = false;
                        boolean isclassname = false;
                        boolean isparenthese = false;
                        boolean isreturn = false;
                        boolean isprikey = false;
                        boolean isforkey = false;
                        boolean isrelated = false;
                        boolean isObject = false;
                        boolean isdoes = false;
                        int             nattwords = 0;/* words counter
*/
                        int             nstar = 0;
                        FILE            *infile, *outpkey, *outfkey,
*outclass;

                        infile = fopen("sp_helpschemma_out", "r");
                        outpkey = fopen("outpkey", "w");
                        outfkey = fopen("outfkey", "w");
                        outclass = fopen("outclass", "w");

[0-9]+                  ;
index_name              {isattributes = false;
                        }
Object                  isObject = true;/* "Object does not" ends */
does                    {if (isObject) isdoes = true;} /* attributes
section*/
not                     {if (isdoes) {
                                isattributes = false;
                                isObject = false;
                                isdoes = false;
                                }
                        }
[\-]+                   {if (yyleng == 8) {
                                isattributes = true;
```

```
                                    nattwords = 0;
                                    }
                     else if (yyleng == 22) {
                                    isclassname = true;
                     /*             fprintf("\nCLASS NUMBER %d",
nclasses++);
                     */             }
                     }
primary              {
                       fprintf(outpkey, "\n%s", yytext);
                       isprikey = true;
                       isrelated = false;
                       isforkey = false;
                     }
foreign              {
                       fprintf(outfkey, "\n%s", yytext);
                       nstar = 0;
                       isforkey = true;
                       isrelated = false;
                       isprikey = false;
                     }
[\(]                 isparenthese = true;
return               {isrelated = false;
                     if (isparenthese) isreturn = true;}
status               {if (isreturn) {
                                    isprikey = false;
                                    isforkey = false;
                                    isrelated = false;
                                    isparenthese = false;
                                    isreturn = false;
                                    }
                     }
[\*]+                {
                     if ((nstar == 0) && (isforkey))
                        fprintf(outfkey, " related");
                     nstar++;
                     }
[A-Za-z0-9\_\:]+ |
NULL                 {if (isclassname) {
                                    isclassname = false;
                                    fprintf(outclass, "\nclass
%s\n", yytext);
                                    }
                      else if (isattributes) {
```

```
                                        fprintf(outclass, "%s ",
yytext);
                                        if (nattwords < 5)
                                                nattwords++;
                                        else {
                                                nattwords = 0;
                                                fprintf(outclass,
"\n");
                                             }
                                        }
                              else if (isprikey)
                                 fprintf(outpkey, " %s", yytext);
                              else if (isforkey)
                                 fprintf(outfkey, " %s", yytext);
                              }

\n                            |
.                             ;
%%
                              fclose(infile);
                              fclose(outpkey);
                              fclose(outfkey);
                              fclose(outclass);
```

Apêndice B - Arquivo sp_help.out

Informações do Dicionário de Dados do Sybase.

```
Name                    Owner                   Type
------------------------------ ------------------------------ ----------------------
publishers              dbo                     user table
Data_located_on_segment    When_created
------------------------------ --------------------------
default                    Jun 27 1994  4:04PM
Column_name    Type       Length Prec Scale Nulls Default_name   Rule_name     Identity
--------------- --------------- ------ ---- ----- ----- --------------- --------------- --------
pub_id      char          4 NULL  NULL    0 NULL        pub_idrule        0
pub_name      varchar       40 NULL  NULL    1 NULL        NULL            0
city        varchar       20 NULL  NULL    1 NULL        NULL            0
state       char          2 NULL  NULL    1 NULL        NULL            0
index_name    index_description              index_keys
-------------------- -------------------------------------------------------- ------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------
pubind    clustered, unique located on default      pub_id

(1 row affected)
keytype  object              related_object         object_keys                     related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
------------------------------------------------------------------------
primary  publishers           -- none --         pub_id, *, *, *, *, *, *, *        *, *, *, *, *, *, *, *
foreign  titles               publishers        pub_id, *, *, *, *, *, *, *        pub_id, *, *, *, *, *, *, *
(return status = 0)

Name                    Owner                   Type
------------------------------ ------------------------------ ----------------------
titles                  dbo                     user table

Data_located_on_segment    When_created
------------------------------ --------------------------
default                    Jun 27 1994  4:04PM
Column_name    Type       Length Prec Scale Nulls Default_name   Rule_name     Identity
--------------- --------------- ------ ---- ----- ----- --------------- --------------- --------
title_id     tid           6 NULL  NULL    0 NULL        title_idrule      0
title        varchar       80 NULL  NULL    0 NULL        NULL            0
type         char          12 NULL  NULL    0 typedflt    NULL            0
pub_id       char          4 NULL  NULL    1 NULL        NULL            0
price        money         8 NULL  NULL    1 NULL        NULL            0
advance      money         8 NULL  NULL    1 NULL        NULL            0
```

```
total_sales    int          4 NULL  NULL    1 NULL        NULL          0
notes          varchar    200 NULL  NULL    1 NULL        NULL          0
pubdate        datetime     8 NULL  NULL    0 datedflt    NULL          0
contract       bit          1 NULL  NULL    0 NULL        NULL          0

index_name  index_description                  index_keys
----------- ---------------------------------- ----------------------------------------
--------------------------------------------------------------------------------------
----------------------
titleidind  clustered, unique located on default      title_id
titleind    nonclustered located on default           title

(2 rows affected)
keytype  object          related_object      object_keys                 related_keys
-------- --------------- ------------------- ----------------------------------------
-----------------------------------------------
foreign  roysched        titles         title_id, *, *, *, *, *, *, *     title_id, *, *, *, *, *, *, *
foreign  salesdetail     titles         title_id, *, *, *, *, *, *, *     title_id, *, *, *, *, *, *, *
foreign  titleauthor     titles         title_id, *, *, *, *, *, *, *     title_id, *, *, *, *, *, *, *
foreign  titles          publishers     pub_id, *, *, *, *, *, *, *       pub_id, *, *, *, *, *, *, *
primary  titles          -- none --     title_id, *, *, *, *, *, *, *     *, *, *, *, *, *, *, *
(return status = 0)

Name                 Owner                Type
-------------------- -------------------- ---------------
authors              dbo                  user table

Data_located_on_segment   When_created
------------------------- ------------------
default                   Jun 27 1994  4:04PM

Column_name  Type      Length Prec Scale Nulls Default_name  Rule_name   Identity
------------ --------- ------ ---- ----- ----- ------------- ----------- --------
au_id        id          11 NULL  NULL    0 NULL        NULL          0
au_lname     varchar     40 NULL  NULL    0 NULL        NULL          0
au_fname     varchar     20 NULL  NULL    0 NULL        NULL          0
phone        char        12 NULL  NULL    0 phonedflt   NULL          0
address      varchar     40 NULL  NULL    1 NULL        NULL          0
city         varchar     20 NULL  NULL    1 NULL        NULL          0
state        char         2 NULL  NULL    1 NULL        NULL          0
country      varchar     12 NULL  NULL    1 NULL        NULL          0
postalcode   char        10 NULL  NULL    1 NULL        NULL          0

index_name  index_description                  index_keys
----------- ---------------------------------- ----------------------------------------
--------------------------------------------------------------------------------------
----------------------
auidind     clustered, unique located on default      au_id
```

```
aunmind     nonclustered located on default      au_lname, au_fname

(2 rows affected)
keytype  object           related_object      object_keys                 related_keys
-------- ---------------- ------------------- --------------------------- ------------------------------
--------------------------------------------
foreign  au_pix           authors             au_id, *, *, *, *, *, *, *   au_id, *, *, *, *, *, *, *
primary  authors          -- none --          au_id, *, *, *, *, *, *, *   *, *, *, *, *, *, *, *
foreign  blurbs           authors             au_id, *, *, *, *, *, *, *   au_id, *, *, *, *, *, *, *
foreign  titleauthor      authors             au_id, *, *, *, *, *, *, *   au_id, *, *, *, *, *, *, *
(return status = 0)

Name                Owner               Type
------------------- ------------------- ---------------
titleauthor         dbo                 user table

Data_located_on_segment   When_created
------------------------- ------------------
default                   Jun 27 1994  4:04PM

Column_name  Type      Length Prec Scale Nulls Default_name  Rule_name  Identity
------------ --------- ------ ---- ----- ----- ------------- ---------- --------
au_id        id          11 NULL  NULL    0 NULL        NULL          0
title_id     tid          6 NULL  NULL    0 NULL        NULL          0
au_ord       tinyint      1 NULL  NULL    1 NULL        NULL          0
royaltyper   int          4 NULL  NULL    1 NULL        NULL          0

index_name   index_description                   index_keys
------------ ----------------------------------- ------------------------------------------
-------------------------------------------------------------------------------------------
--------------------
taind        clustered, unique located on default    au_id, title_id
auidind      nonclustered located on default         au_id
titleidind   nonclustered located on default         title_id

(3 rows affected)
keytype  object           related_object      object_keys                          related_keys
-------- ---------------- ------------------- ------------------------------------ ------------------------------
--------------------------------------------
foreign  titleauthor      authors             au_id, *, *, *, *, *, *, *            au_id, *, *, *, *, *, *, *
foreign  titleauthor      titles              title_id, *, *, *, *, *, *, *         title_id, *, *, *, *, *, *, *
primary  titleauthor      -- none --          au_id, title_id, *, *, *, *, *, *     *, *, *, *, *, *, *, *
(return status = 0)

Name                Owner               Type
------------------- ------------------- ---------------
roysched            dbo                 user table
```

```
Data_located_on_segment      When_created
---------------------------- -------------------------
default                      Jun 27 1994  4:04PM

Column_name    Type        Length Prec Scale Nulls Default_name   Rule_name    Identity
-------------- ----------- ------ ---- ----- ----- -------------- ------------ --------
title_id       tid              6 NULL NULL      0 NULL           NULL                0
lorange        int              4 NULL NULL      1 NULL           NULL                0
hirange        int              4 NULL NULL      1 NULL           NULL                0
royalty        int              4 NULL NULL      1 NULL           NULL                0

index_name  index_description                  index_keys
----------- ---------------------------------- ----------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------
titleidind  nonclustered located on default    title_id

(1 row affected)
keytype  object         related_object      object_keys                     related_keys
-------- -------------- ------------------- ------------------------------- -----------------
------------------------------------------------------------
foreign  roysched       titles              title_id, *, *, *, *, *, *, *    title_id, *, *, *, *, *, *, *
primary  roysched       -- none --          title_id, *, *, *, *, *, *, *    *, *, *, *, *, *, *, *
(return status = 0)

Name                      Owner                     Type
------------------------- ------------------------- --------------------
sales                     dbo                       user table

Data_located_on_segment      When_created
---------------------------- -------------------------
default                      Jun 27 1994  4:04PM

Column_name    Type        Length Prec Scale Nulls Default_name   Rule_name    Identity
-------------- ----------- ------ ---- ----- ----- -------------- ------------ --------
stor_id        char             4 NULL NULL      0 NULL           NULL                0
ord_num        varchar         20 NULL NULL      0 NULL           NULL                0
date           datetime         8 NULL NULL      0 NULL           NULL                0

index_name  index_description                  index_keys
----------- ---------------------------------- ----------------------------------------------
--------------------------------------------------------------------------------------------
--------------------------------
salesind    clustered, unique located on default   stor_id, ord_num

(1 row affected)
keytype  object         related_object      object_keys                     related_keys
-------- -------------- ------------------- ------------------------------- -----------------
```

```
------------------------------------------------------------------
foreign  sales         stores        stor_id, *, *, *, *, *, *, *        stor_id, *, *, *, *, *, *, *
primary  sales         -- none --    stor_id, ord_num, *, *, *, *, *, *       *, *, *, *, *, *, *, *
foreign  salesdetail        sales         stor_id, ord_num, *, *, *, *, *, *          stor_id, ord_num,
*, *, *, *, *, *
(return status = 0)

Name                  Owner                 Type
------------------------------ ------------------------------ ----------------------
salesdetail           dbo                   user table

Data_located_on_segment     When_created
------------------------------ --------------------------
default                     Jun 27 1994  4:04PM

Column_name   Type       Length Prec Scale Nulls Default_name   Rule_name    Identity
--------------- --------------- ------ ---- ----- ----- -------------- -------------- --------
stor_id      char          4 NULL  NULL    0 NULL        NULL           0
ord_num      varchar       20 NULL  NULL    0 NULL        NULL           0
title_id     tid           6 NULL  NULL    0 NULL        title_idrule   0
qty          smallint      2 NULL  NULL    0 NULL        NULL           0
discount     float         8 NULL  NULL    0 NULL        NULL           0

index_name   index_description          index_keys
-------------------- -------------------------------------------------------- ----------------------------------------------------------------------
----------------------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------
titleidind    nonclustered located on default      title_id
salesdetailind  nonclustered located on default      stor_id, ord_num

(2 rows affected)
keytype  object           related_object        object_keys                 related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
------------------------------------------------------------------
foreign  salesdetail         sales          stor_id, ord_num, *, *, *, *, *, *          stor_id, ord_num,
*, *, *, *, *, *
foreign  salesdetail      titles        title_id, *, *, *, *, *, *, *          title_id, *, *, *, *, *, *, *
primary  salesdetail          -- none --      stor_id, ord_num, *, *, *, *, *, *          *, *, *, *, *,
*, *, *
(return status = 0)

Name                  Owner                 Type
------------------------------ ------------------------------ ----------------------
stores                dbo                   user table

Data_located_on_segment     When_created
------------------------------ --------------------------
default                     Jun 27 1994  4:04PM
```

```
Column_name    Type         Length Prec Scale Nulls Default_name   Rule_name      Identity
--------------- --------------- ------ ---- ----- ----- -------------- -------------- --------
stor_id        char            4 NULL  NULL    0 NULL        NULL              0
stor_name      varchar        40 NULL  NULL    1 NULL        NULL              0
stor_address   varchar        40 NULL  NULL    1 NULL        NULL              0
city           varchar        20 NULL  NULL    1 NULL        NULL              0
state          char            2 NULL  NULL    1 NULL        NULL              0
country        varchar        12 NULL  NULL    1 NULL        NULL              0
postalcode     char           10 NULL  NULL    1 NULL        NULL              0
payterms       varchar        12 NULL  NULL    1 NULL        NULL              0
Object does not have any indexes.

keytype  object           related_object        object_keys                    related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
----------------------------------------------------------------------
foreign  discounts        stores           stor_id, *, *, *, *, *, *, *      stor_id, *, *, *, *, *, *, *
foreign  sales            stores           stor_id, *, *, *, *, *, *, *      stor_id, *, *, *, *, *, *, *
primary  stores           -- none --       stor_id, *, *, *, *, *, *, *      *, *, *, *, *, *, *, *
(return status = 0)

Name                     Owner                     Type
------------------------------ ------------------------------ ----------------------
discounts                dbo                       user table

Data_located_on_segment     When_created
------------------------------ ---------------------------
default                     Jun 27 1994  4:04PM

Column_name    Type         Length Prec Scale Nulls Default_name   Rule_name      Identity
--------------- --------------- ------ ---- ----- ----- -------------- -------------- --------
discounttype   varchar        40 NULL  NULL    0 NULL        NULL              0
stor_id        char            4 NULL  NULL    1 NULL        NULL              0
lowqty         smallint        2 NULL  NULL    1 NULL        NULL              0
highqty        smallint        2 NULL  NULL    1 NULL        NULL              0
discount       float           8 NULL  NULL    0 NULL        NULL              0
Object does not have any indexes.

keytype  object           related_object        object_keys                    related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
----------------------------------------------------------------------
foreign  discounts        stores           stor_id, *, *, *, *, *, *, *      stor_id, *, *, *, *, *, *, *
primary  discounts        -- none --       discounttype, stor_id, *, *, *, *, *, *      *, *, *, *, *,
*, *, *
(return status = 0)

Name                     Owner                     Type
------------------------------ ------------------------------ ----------------------
```

```
au_pix                dbo                  user table

Data_located_on_segment    When_created
------------------------------ --------------------------
default                    Jun 27 1994  4:04PM

Column_name    Type         Length Prec Scale Nulls Default_name   Rule_name     Identity
--------------- --------------- ------ ---- ----- ----- --------------- --------------- --------
au_id          char          11 NULL  NULL    0 NULL        NULL            0
pic            image         16 NULL  NULL    1 NULL        NULL            0
format_type    char          11 NULL  NULL    1 NULL        NULL            0
bytesize       int            4 NULL  NULL    1 NULL        NULL            0
pixwidth_hor   char          14 NULL  NULL    1 NULL        NULL            0
pixwidth_vert  char          14 NULL  NULL    1 NULL        NULL            0
Object does not have any indexes.

keytype  object            related_object        object_keys                      related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
----------------------------------------------------------------------
foreign  au_pix           authors          au_id, *, *, *, *, *, *, *           au_id, *, *, *, *, *, *, *
primary  au_pix            -- none --        au_id, *, *, *, *, *, *, *            *, *, *, *, *, *, *, *
(return status = 0)

Name                   Owner                  Type
------------------------------ ------------------------------ ----------------------
blurbs                 dbo                    user table

Data_located_on_segment    When_created
------------------------------ --------------------------
default                    Jun 27 1994  4:04PM

Column_name    Type         Length Prec Scale Nulls Default_name   Rule_name     Identity
--------------- --------------- ------ ---- ----- ----- --------------- --------------- --------
au_id          id            11 NULL  NULL    0 NULL        NULL            0
copy           text          16 NULL  NULL    1 NULL        NULL            0
Object does not have any indexes.

keytype  object            related_object        object_keys                      related_keys
---------- ------------------------------ ------------------------------ ----------------------------------------------------------------------
----------------------------------------------------------------------
foreign  blurbs           authors          au_id, *, *, *, *, *, *, *           au_id, *, *, *, *, *, *, *
primary  blurbs            -- none --        au_id, *, *, *, *, *, *, *            *, *, *, *, *, *, *, *
(return status = 0)
```

Apêndice C - Arquivo syrelati

Esquema Relacional reconstituído a partir do banco de dados pubs2 modificado.

All relations created from Sybase data dictionary schema information

```
Relation[0]: publishers
 pub_id: char
 pub_name: varchar
 city: varchar
 state: char
  primary key[0]: pub_id

Relation[1]: titles
 title_id: tid
 title: varchar
 pub_id: char
 pubdate: datetime
  primary key[0]: title_id
  foreign name[0]: publishers
      foreign key[0]: pub_id
      related key[0]: pub_id

Relation[2]: authors
 au_id: id
 au_name: varchar
 address: varchar
 city: varchar
 state: char
 country: varchar
 zipcode: char
 phone: char
 birthdate: datetime
 age: int
  primary key[0]: au_id

Relation[3]: titleauthor
 au_id: id
 title_id: tid
 au_ord: tinyint
 royaltyper: int
```

```
    primary key[0]: au_id
    primary key[1]: title_id
    foreign name[0]: titles
        foreign key[0]: title_id
        related key[0]: title_id
    foreign name[1]: authors
        foreign key[0]: au_id
        related key[0]: au_id

Relation[4]: roysched
  title_id: tid
  lorange: int
  hirange: int
  royalty: int
    primary key[0]: title_id
    foreign name[0]: titles
        foreign key[0]: title_id
        related key[0]: title_id

Relation[5]: sales
  stor_id: char
  ord_num: varchar
  date: datetime
    primary key[0]: stor_id
    primary key[1]: ord_num
    foreign name[0]: stores
        foreign key[0]: stor_id
        related key[0]: stor_id

Relation[6]: salesdetail
  stor_id: char
  ord_num: varchar
  title_id: tid
  qty: smallint
  discount: float
    primary key[0]: stor_id
    primary key[1]: ord_num
    foreign name[0]: titles
        foreign key[0]: title_id
        related key[0]: title_id
    foreign name[1]: sales
        foreign key[0]: stor_id
        related key[0]: stor_id
        foreign key[1]: ord_num
```

```
        related key[1]: ord_num

Relation[7]: stores
  stor_id: char
  stor_name: varchar
  stor_address: varchar
  city: varchar
  payterms: varchar
    primary key[0]: stor_id

Relation[8]: discounts
  discounttype: varchar
  stor_id: char
  lowqty: smallint
  highqty: smallint
  discount: float
    primary key[0]: discounttype
    primary key[1]: stor_id
    foreign name[0]: stores
        foreign key[0]: stor_id
        related key[0]: stor_id

Relation[9]: au_pix
  au_id: char
  pic: image
  format_type: char
  bytesize: int
  pixwidth_hor: char
  pixwidth_vert: char
    primary key[0]: au_id
    foreign name[0]: authors
        foreign key[0]: au_id
        related key[0]: au_id

Relation[10]: blurbs
  au_id: id
  copy: text
    primary key[0]: au_id
    foreign name[0]: authors
        foreign key[0]: au_id
        related key[0]: au_id

Relation[11]: book
  title_id: tid
```

```
  edition: varchar
  print: varchar
  price: int
    primary key[0]: title_id
    foreign name[0]: titles
        foreign key[0]: title_id
        related key[0]: title_id

Relation[12]: paper
  title_id: tid
  vol.: varcahr
  no.: smallint
    primary key[0]: title_id
    foreign name[0]: titles
        foreign key[0]: title_id
        related key[0]: title_id

Relation[13]: authors-hobbies
  au_id: id
  hobbies: varchar
    primary key[0]: au_id
    primary key[1]: hobbies
    foreign name[0]: authors
        foreign key[0]: au_id
        related key[0]: au_id

Relation[14]: dependent
  au_id: id
  name: varchar
  sex: char
  birthdate: datetime
    primary key[0]: au_id
    primary key[1]]: name
    foreign name[0]: authors
        foreign key[0]: au_id
        related key[0]: au_id
```

Apêndice D - Arquivo o2class

Esquema Orientado a Objetos O2 - transformado a partir do esquema relacional Sybase do pubs2 modificado..

```
class publishers
  type tuple(pub_id: string
             pub_name: string
             city: string
             state: string)

class titles
type tuple(title_id: string
           title: string
           pubdate: date
           pub_id: publishers
           roysched: tuple(
                    lorange: integer
                    hirange: integer
                    royalty: integer)
           titleauthor: set(tuple(
                    authors: authors
                    au_ord: integer
                    royaltyper: integer)))

class book
inherit titles
type tuple(title_id: string
                edition: string
                print: string
                price: integer)

class paper
inherit titles
type tuple(title_id: string
                vol.: string
                 no.: integer)


class authors
type tuple(au_id: string
```

```
        au_name: string
        address: string
        city: string
        state: string
        country: string
        zipcode: string
        phone: string
        copy: string
        au_pix: tuple(
                pic: bitmap
                format_type: string
                bytesize: integer
                pixwidth_hor: string
                pixwidth_vert: string)
        hobbies: set(string)
        titleauthor: set(tuple(
                titles: titles
                au_ord: integer
                royaltyper: integer)))

class dependent
type tuple(au_id: string
        name: string
        sex: string
        birthdate: date)


class sales
  type tuple(stor_id: string
          ord_num: string
          date: date)

class salesdetail
  inherit sales
  type tuple(stor_id: string
          ord_num: string
          title_id: string
          qty: integer
          discount: float)

class stores
  type tuple(stor_id: string
          stor_name: string
          stor_address: string
```

```
            city: string
            payterms: string)

class discounts
  type tuple(discounttype: string
            stor_id: string
            lowqty: integer
            highqty: integer
            discount: float)
```