S-SQL: Uma Interface Semântica

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Ricardo Cézar Nery Oliveira e aprovada pela Comissão Julgadora.

Campinas, 24 de agosto de 1992

Prof. Drº. Geovane Cayres Magalhães

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para obtenção do Título de Mestre em Ciência da Computação.

UNIVERSIDADE ESTADUAL DE CAMPINAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

S-SQL: UMA INTERFACE SEMÂNTICA

por

Ricardo Cézar Nery Oliveira

Dissertação submetida como requisito parcial para
a obtenção do grau de Mestre em
Ciência da Computação

Profª. Dr. Geovane Cayres Magalhães
Orientador

Campinas, 24 de Agosto de 1992

A Nádia e Vinícius.

AGRADECIMENTOS

Ao *brother* Luiz Cláudio, pela implacável revisão dessa dissertação.

Ao *brother* Júlio, pelas farras que tanto "colaboraram" para o andamento deste trabalho.

À Ulisses, pela orientação na utilização do LEX e YACC.

À Geovane e Augusta. Ele, pela orientação. Ela, pela incrível atenção nas minhas estadias.

À Ângela pela amizade e acolhida.

À Bacca, uma amizade ímpar, um grande orientador na mais difícil das teses: a vida.

Aos meus pais, por proporcionarem todas as condições para que pudesse chegar aqui.

RESUMO

Apesar de serem reconhecidas, as críticas aos modelos de dados clássicos, em especial ao modelo relacional, não motivaram a indústria a investir no desenvolvimento dos modelos semânticos. Os usuários, dentro deste contexto, encontram-se desamparados à espera de um sistema de gerenciamento de banco de dados (SGBD) que ofereça as diversas facilidades que, embora fartamente discutidas, ainda não se encontram comercialmente disponíveis.

A dissertação que se segue propõe uma interface semântica para SGDBs relacionais que têm como base a linguagem SQL. SQL é uma linguagem de banco de dados bastante difundida, podendo ser considerada um padrão da indústria. A interface S-SQL é composta de um modelo que oferece os mecanismos de classificação, agregação, generalização, herança múltipla, classes derivadas com base em predicados e atributos, atributos multivalorados, bem como, uma linguagem de consulta associada, sintaticamente derivada da linguagem SQL.

ABSTRACT

Despite the acknowledgement of the critics to the record-oriented models, specially to the relational model, the industry was not motivated to invest on the development of the semantic models. In that context, users are hopelessly waiting for a DBMS that can provide the several facilities that, in spite of being largely discussed, are not commercially available yet.

This dissertation presents a proposition of a semantic interface for SQL relational DBMSs. SQL is a widely spread database language that can be considered an industry standard. The S-SQL interface comprises a model that provide the mechanisms of classification, aggregation, generalization, multiple inheritance, derived class based on predicates and attributes, multivalued attributes, as well as a query language syntactically derived from SQL.

Índice

Figuras

Capítulo 1

Introdução

1.1 Revisão Histórica

Os primeiros sistemas de informação foram implementados por conjuntos de programas que manipulavam os dados relativos às aplicações em arquivos diversos. Esse era um ambiente típico de sistema de processamento de arquivos, e, como tal, suportado apenas pelo sistema operacional. O fraco nível de compartilhamento e disponibilidade dos dados armazenados foram alguns dos problemas encontrados nessa abordagem que motivaram o desenvolvimento da tecnologia de sistemas de gerenciamento de banco de dados(SGBD).

Os SGBDs surgiram com o intuito de tornar mais flexível e eficiente o compartilhamento, gerenciamento, recuperação e armazenamento de grandes quantidades de dados. Pode-se resumir como objetivos centrais à época:

- Compartilhar e permitir a utilização concorrente dos dados por diversos usuários;
- Reduzir a redundância e inconsistência dos dados a serem armazenados;
- Oferecer um sistema de segurança confiável, a fim de proteger os dados de atualizações inadvertidas ou maliciosas;
- Fornecer uma visão abstrata dos dados, tentando esconder detalhes de implementação;
- Facilitar o acesso ao banco de dados através de linguagens de consulta e manipulação;

- Fornecer um sistema de recuperação atento às falhas de hardware e software, junto a um sistema de *backup* adequado;
- Permitir que algumas restrições, inerentes aos dados das aplicações, fossem incorporadas e mantidas pelo SGBD.

A área de modelos de dados, desde então, motivou uma série de pesquisas que objetivaram desenvolver os conceitos necessários à representação das propriedades das aplicações.

Basicamente, as propriedades de uma aplicação podem ser agrupadas nos aspectos estrutural e comportamental. O aspecto estrutural compreende a descrição das propriedades das entidades envolvidas e dos seus relacionamentos. O aspecto comportamental compreende a representação das operações e transações que interagem sobre entidades e relacionamentos. Na literatura, entretanto, grande parte dos modelos encontrados relegam a representação dos aspectos comportamentais aos programas.

Em parte, pela preocupação inicial em desenvolver SGBDs que fossem capazes de processar grandes quantidades de dados, os primeiros modelos propostos foram extremamente associados à unidade de estruturação física dos dados. Por isso, esses modelos, genericamente denominado de clássicos[BROD84], são também chamados de modelos orientados a registros.

O modelo hierárquico[TSCI76], por exemplo, vislumbra as aplicações como uma coleção de entidades (registros) organizadas hierarquicamente. O modelo DBTG[TAYL76] permite uma organização em rede, sendo assim, uma flexibilização do modelo anterior. Ambos se caracterizaram por oferecer uma linguagem de manipulação do tipo "um registro por vez" e por não protegerem os programas aplicativos das mudanças físicas ocorridas no banco de dados.

O modelo relacional[CODD70] surgiu contrapondo essas características. Através do seu forte embasamento teórico, este modelo enfatizou a independência física dos dados e propiciou uma linguagem de manipulação de dados orientada a conjuntos. No modelo relacional, uma aplicação é vista como uma coleção de relações que descrevem entidades e relacionamentos. Ao contrário dos modelos

anteriores, a representação dos relacionamentos entre as relações é feita através dos valores dos atributos, e não mais por ponteiros.

Ao mesmo tempo em que afirmava-se como opção definitiva entre os modelos clássicos, apareciam as primeiras critícas ao modelo relacional[KENT79]. De uma forma geral, a constatação era a mesma: a sobrecarga semântica da estrutura de registro impedia uma representação adequada dos diversos relacionamentos existentes em uma aplicação. A partir daí, surgem os modelos semânticos [CHEN76, CODD79, MYLO80, HAMM81, SHIP81, BROD81B, ZANI83] que desenvolveram uma série de conceitos e mecanismos que propiciaram uma maior captação das propriedades de uma aplicação, dentre as quais, ressaltam-se os mecanismos de abstração de dados.

Contudo, esses modelos, devido a uma gama de fatores, não obtiveram o reconhecimento da indústria, enquanto que, parte da comunidade acadêmica, voltava as suas atenções para os modelos de dados orientados a objetos. Os usuários, por sua vez, encontram-se, restritos a SGBDs relacionais que nem sequer suportam todos os conceitos propostos por esse modelo.

1.2 Objetivo e Organização da Dissertação

O objetivo desta dissertação é o de propor uma interface semântica para SGBDs relacionais SQL. Denominada de S-SQL, essa interface oferece as abstrações de classificação, generalização e agregação, bem como, herança múltipla, derivação de classes com base em predicados e atributos multivalorados[1]. Associadas à proposta da interface, encontram-se as linguagens de definição e manipulação de dados, que são derivações sintáticas da linguagem SQL.

Todo o processo de captação semântica é baseado em extensões ao dicionário de dados e na tradução de transações S-SQL em seqüências de transações SQL. Devido a um mapeamento simples e intuitivo dos mecanismos de abstração de dados, boa parte da manutenção da integridade semântica restringe-se à manutenção da integridade referencial.

---

1. A palavra multivalorado não existe na língua portuguesa. A mesma é um neologismo derivado da palavra inglesa *multivalued*.

O capítulo 2 aborda os conceitos e definições básicas sobre modelagem de dados, além de uma rápida revisão sobre os modelos de dados clássicos e as suas principais críticas.

No capítulo 3 é feita uma explanação sobre os mecanismos de abstrações de dados e um apanhado de alguns dos principais modelos semânticos. Encerra-o, uma pequena revisão do paradigma de orientação a objetos e uma comparação entre os modelos semânticos e os modelos orientados a objetos.

O capítulo 4 começa com a descrição dos principais motivos que levaram à confecção deste trabalho, seguida pela apresentação da interface S-SQL: linguagens de definição e manipulação, e representação gráfica.

O capítulo 5 inicia com a revisão da implementação de abordagens correlatas. Em seguida, é descrita a implementação da interface S-SQL, explanando-se o processo de tradução de transações, mapeamento de abstrações, extensão do dicionário de dados e módulos componentes da interface.

O capítulo 6 relata as conclusões do desenvolvimento deste trabalho, bem como, propostas para trabalhos futuros.

Capítulo 2

Modelos de Dados Clássicos

2.1 Modelagem de Dados

Modelagem é a construção de uma representação capaz de captar as propriedades estruturais e comportamentais[2] de uma aplicação.

Antes de se efetivar a modelagem de dados de qualquer aplicação, é fundamental o levantamento dos seus requisitos de informação e de processamento. Existem diversas metodologias destinadas a esse tipo de tarefa que, por não serem relevantes a este trabalho, não serão abordadas. Três itens fundamentais ao processo de modelagem de dados devem ser determinados a partir do processo de levantamento de informações, quais são:

- Entidades da aplicação e seus respectivos relacionamentos;
- Requisitos de informação da aplicação;
- Descrição das transações existentes.

Segundo Brodie[BROD84], um modelo de dados é uma coleção de conceitos, matematicamente bem definidos, que auxiliam a considerar e representar as propriedades estruturais e comportamentais de uma aplicação. Como ele próprio admite, esta é uma definição ideal, dado que muitos modelos foram desenvolvidos de forma intuitiva sem um formalismo associado.

---

2. Também denominadas de propriedades estáticas e dinâmicas.

Como visto, a completa representação de uma aplicação de banco de dados deve incluir as suas propriedades estruturais e comportamentais. O aspecto estrutural compreende a descrição das propriedades das entidades e relacionamentos e das suas restrições de integridades. Essas restrições podem ser descritas pelo usuário ou implícitas na própria semântica dos relacionamentos e entidades. Já o aspecto comportamental, compreende a especificação de operações primitivas sobre as entidades, a composição de transações junto aos seus pré e pós-requisitos, e a recuperação de erros.

Alguns autores também consideram as propriedades temporais como um aspecto a ser modelado em uma aplicação[SNOD85]. O suporte ao aspecto temporal pode variar desde o simples armazenamento de informações passadas, ao complexo controle e gerenciamento de versões das estruturas dos objetos de uma aplicação.

2.1.1 Conceitos Básicos

Um esquema é a representação das propriedades de uma aplicação por um modelo de dados. Nele são determinados os tipos de entidades, atributos, relacionamentos, restrições, operações e transações. Um sub-esquema, como o próprio nome induz, é um subconjunto das propriedades de um esquema. Um sub-esquema tem como função proporcionar um acesso localizado, pois, para a maioria dos usuários, não é desejável uma visão total dos dados de uma aplicação. A cada esquema corresponde um repositório de dados denominado de banco de dados.

Ferramentas devem ser obrigatoriamente desenvolvidas a partir de um modelo de dados com o objetivo de permitir a sua implementação e conseqüente utilização. Em geral, essas ferramentas incluem as linguagens de definição de dados (LDD), de consulta (LC) e de manipulação de dados (LMD)[3].

O conceito de entidade é bastante abrangente e, por isso, vago. Uma entidade é qualquer coisa relevante o bastante para merecer o armazenamento de suas propriedades no banco de dados. Atributos são usados para descrever as

3. Muitos autores não distinguem a LC da LMD, neste texto, também será assumida essa conjunção, exceto quando estiver explícito o contrário.

propriedades associadas a uma entidade. Alguns modelos exigem a definição de um identificador ou chave, que é o conjunto mínimo de atributos cujos valores identificam cada ocorrência de uma entidade. A cada entidade corresponderá uma coleção de instâncias relativas às ocorrências desta na aplicação.

Relacionamentos exprimem os diversos tipos de associações existentes entre entidades. Os tipos de relacionamento suportados por um modelo estão estreitamente ligados a sua capacidade de representação, tornando-se assim, um item importante na sua avaliação e comparação. Alguns modelos, como os clássicos, restringem-se a relacionamentos simples, como 1:1, 1:N e N:M. Outros, oferecem relacionamentos que incorporam e expressam parte da semântica da aplicação modelada, como é-parte-de, é-membro-de, etc[HULL87].

Para cada entidade, os modelos oferecem operações primitivas, como inclusão e exclusão, que irão atuar sobre as suas ocorrências. Essas operações não correspondem aos procedimentos existentes no mundo real e, por isso, alguns modelos permitem a composição dessas operações em transações, que tentam ser correspondentes mais apropriadas. Junto às transações, são associados predicados, que determinam os pré e pós-requisitos de execução, e mecanismos de recuperação de erros, que mantêm a característica básica de atomicidade destas.

A manutenção da integridade e coerência das informações contidas em um banco de dados, é de grande importância para a confiabilidade do mesmo. Restrições são regras que regulam e asseguram a manutenção das propriedades estruturais e comportamentais de um aplicação. De uma certa forma, as restrições também são consideradas como propriedades de uma aplicação, apesar de algumas delas serem inapelavelmente herdadas do modelo em uso. Por exemplo, no modelo hierárquico os relacionamentos devem ser necessariamente do tipo 1:N.

2.2 Modelos Clássicos

A seguir serão descritos alguns modelos de dados que têm o registro como unidade de estruturação básica . A capacidade de representação destes será primordialmente enfocada, em detrimento dos seus aspectos operacionais de implementação.

2.2.1 O Modelo Hierárquico

O modelo hierárquico fundamentou-se em um princípio que, embora lógico, foi visivelmente simplista, no qual o mundo possue um grande sentido de ordem e estruturação entre os objetos que o constitui, e uma das estruturas mais naturais seria a hierárquica [TSIC76]. Como mostra a Figura 2.1, uma aplicação representada no modelo hierárquico é uma coleção de hierarquias na qual os nodos são as entidades captadas na aplicação, e as ligações são os diversos relacionamentos 1:N entre essas entidades.

Esta relativa simplicidade do modelo hierárquico, torna o mesmo incapaz de expressar de maneira satisfatória os relacionamentos do tipo N:M. Esses relacionamentos devem ser representados por duas hierarquias distintas, cada uma implementando um sentido do relacionamento.

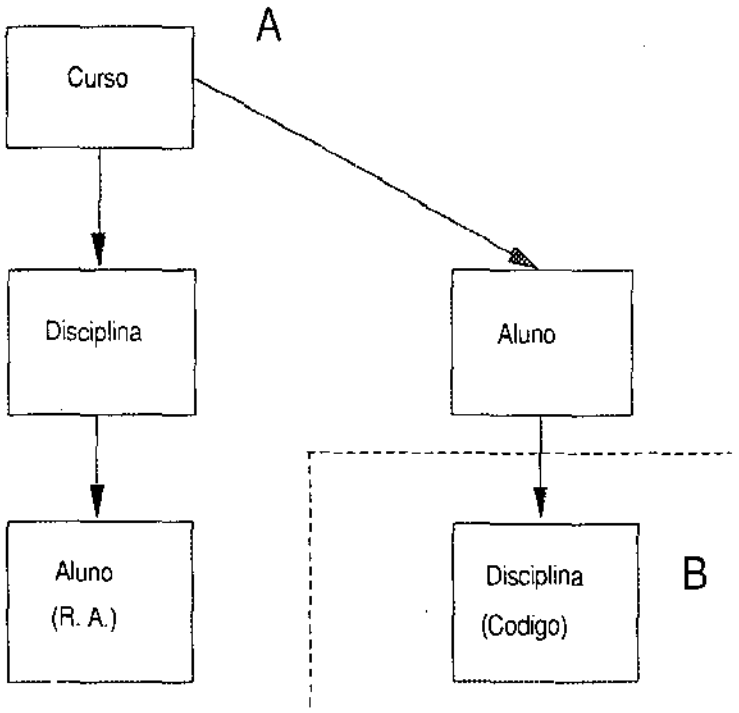


Figura 2.1: Aplicação no modelo hierárquico

Pode-se ainda, optar pela simplificação da representação do relacionamento N:M em uma única ligação 1:N. Nesse caso, deve-se escolher entre as entidades relacionadas, qual será o nodo pai da ligação. Todas essas decisões, devem ser sempre baseadas nos requisitos básicos de informação da aplicação modelada.

Como mostra a figura 2.1, existe um relacionamento N:M entre os registros `Aluno` e `Disciplina`, por isso, criou-se a hierarquia exibida na parte B da figura. Se a hierarquia da parte B fosse desprezada, esse relacionamento seria representado apenas pela ligação 1:N da parte A. Deduz-se com base na figura 2.1, que a consulta dos alunos de uma determinada disciplina e das disciplinas de um determinado aluno são igualmente importantes para a aplicação, e devem estar entre os seus principais requisitos de informação.

Atualmente, o modelo hierárquico possui apenas importância histórica na demonstração do processo evolutivo de desenvolvimento dos modelos de dados. Simplicidade e facilidade de implementação são reconhecidos como as suas principais virtudes[TSIC76].

2.2.2 O Modelo CODASYL/DBTG

Também conhecido como modelo de redes, o modelo CODASYL/DBTG é considerado um aprimoramento do modelo hierárquico. Esse modelo relaxa a restrição imposta pela estrutura hierárquica, na qual um nodo filho deve estar associado a um único nodo pai. Dessa forma, ao invés de uma rígida estrutura hierárquica, tem-se uma estrutura em rede. Entretanto, do mesmo modo que o modelo hierárquico, as ligações[4] entre nodos pai e filho[5] são restritas ao tipo 1:N.

Um relacionamento N:M é representado através da criação de um registro formado pelas chaves dos registros que se relacionam e mais, se por ventura houver, os atributos inerentes ao próprio relacionamento. Duas ligações 1:N complementarão a representação do relacionamento. A figura 2.2 mostra o relacionamento `Matrícula`, do tipo N:M, entre os registros `Aluno` e `Disciplina`.

---

4. Denominadas *sets* na terminologia CODASYL/DBTG.

5. Denominados, respectivamente, *owner record* e *member record* na terminologia CODASYL/DBTG.

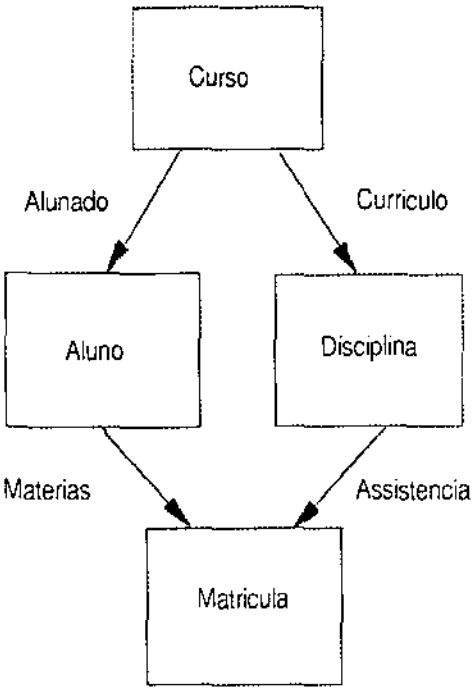


Figura 2.2: Aplicação no modelo CODASYL/DBTG

O modelo CODASYL/DBTG possui na sua LDD algumas facilidades a dirigir as estratégias de armazenamento e busca dos registros e suas ligações. Com isso, o programador, de posse dessas informações, projeta consultas que, em detrimento da independência física dos dados, são bem mais eficientes[TAYL76].

Essa estreita ligação entre modelagem e implementação dos dados, impôs a implementação de uma LMD procedimental. A LMD do modelo CODASYL/DBTG não foi projetada para servir de interface entre usuários finais e o SGBD, mas para servir programadores experientes, capazes de codificar as suas consultas e atualizações através de comandos da LMD e de expressões lógicas da linguagem hospedeira.

2.2.3 O Modelo Relacional

A capacidade de acomodar as modificações ocorridas ao longo do tempo de vida de uma aplicação pelos modelos anteriores deixava a desejar. Quase todo tipo de modificação no esquema demandava uma completa reorganização do banco de dados. Um modelo que enfatizasse a independência da visão do usuário de

qualquer outro aspecto do SGBD era premente. O modelo relacional, proposto por Codd[CODD70] no início da década de 70, trouxe idéias inéditas aos modelos da época:

- Embasamento teórico – O modelo é fundamentado na teoria das relações e no cálculo de predicados de primeira ordem.

- Independência física dos dados – O modelo não oferece diretivas que permitem aos usuários dirigir as estratégias de implementação do banco de dados. Por conseguinte, a visão dos usuários não é afetada pelas alterações acontecidas no nível físico.

Como mostra a figura 2.3, o modelo relacional representa os dados de uma aplicação através de um conjunto de relações que podem ser visualizadas através de uma forma tabular. As linhas e colunas de uma relação são denominadas, respectivamente, de tuplas e atributos. A cada atributo é associado um domínio definindo o conjunto de valores que esse atributo poderá assumir. Para cada relação é definida uma chave primária, sendo possível, também, a existência de outros conjuntos apropriados à função de identificador. Essas chaves são denominadas alternativas. Por exemplo, ambos os atributos registro acadêmico ou nome, podem ser designados como chave primária da relação `Aluno`.

Curso

Codigo	Nome
MAT001	Matematica
MAT002	Computacao
ENG001	Eng. Eletrica
⋮	⋮

Disciplina

Codigo	Disciplina	Curso
CC100	Banco de Dados	MAT002
CC101	Sistemas Oper.	MAT002
EE100	Eletronica Digital	ENG001
⋮	⋮	⋮

Matricula

R. A.	Codigo
895494	CC101
895495	CC100
906593	EE100
906593	CC100
⋮	⋮

Aluno

R. A.	Nome	Curso
895494	Ricardo	MAT002
895495	Frederico	MAT002
906593	Nadia	ENG001
⋮	⋮	⋮

Figura 2.3: Aplicação no modelo relacional

O modelo relacional não torna explícito nenhum tipo de relacionamento entre tabelas. Os relacionamentos são efetuados com base nos valores de atributos das relações, ou seja, através de referências simbólicas. Nesse momento, é importante determinar o conceito de chave estrangeira, que é o conjunto de atributos de um relação correspondente à chave de uma outra relação. Dessa forma, relacionamentos N:M são representados através de uma relação constituída pelas chaves das relações relacionadas. A relação `Matrícula` da figura 2.3 é um exemplo dessa representação.

Para formalizar o projeto de esquemas lógicos relacionais foi desenvolvida a teoria da normalização. Ela é constituída por um conjunto de regras, chamadas de formas normais, que, a partir de dependências funcionais e multivaloradas entre atributos chaves e não chaves, disciplinam o mapeamento dos dados de uma aplicação para um conjunto de relações que minimizam a redundância dos dados e as anomalias de atualização[KENT82]. Um atributo B é funcionalmente dependente de um atributo A, quando B é uma fato sobre A, ou seja, quando para cada valor de A, existe somente um único valor de B associado a este.

No seu primeiro artigo[CODD70], Codd notou que o cálculo de predicados de primeira ordem seria uma sub-linguagem adequada à consulta e manipulação das relações. Isso provocou uma grande mudança nos conceitos das LMDs, visto que o cálculo de predicados é uma linguagem orientada a conjuntos. Além disso, ele introduziu alguns operadores da álgebra relacional e explorou algumas propriedades de redundância e integridade das relações. Entretanto, os sistemas relacionais comerciais desenvolvidos negligenciaram parte dos conceitos propostos, como a noção de domínio e chaves primárias[STON88].

Durante a década de 70, transcorreu uma enorme discussão entre os adeptos dos modelos CODASYL/DBTG e relacional[MICH76], marcada inicialmente pela incompreensão das propostas adversárias por ambas as partes. Entretanto, no final desta mesma década, a comunidade de banco de dados optou pelo modelo relacional. Stonebraker[STON88] justifica essa decisão pelo surgimento dos primeiros protótipos relacionais a comprovarem que as implementações deste modelo poderiam ser de razoável eficiência, e pelo desenvolvimento de linguagens de consulta declarativas, como QUEL e SQL.

2.3 Limitações dos Modelos Clássicos

Os modelos clássicos são extremamente rígidos e mais adequados às aplicações de relativa uniformidade nos dados a serem representados. Kent[KENT79] efetuou uma série de críticas a esses modelos, em especial ao modelo relacional. A seguir estão resumidas algumas das principais críticas:

- Homogeneidade nos fatos em um tipo de entidade – A estrutura de registro é melhor adequada quando as entidades representadas por um determinado registro possuem os mesmos atributos. Quando existe uma considerável variação sobre os fatos relevantes dentro de um conjunto de entidades, tem-se que recorrer a modelagens deselegantes e confusas. Por exemplo: alunos especiais, de graduação e pós-graduação, podem requerer atributos particulares a cada tipo de aluno.

- Homogeneidade em um fato – A cada atributo é designado um determinado domínio. Porém, um fato nem sempre está associado a um único domínio.

Por exemplo: registrar matrículas em cursos especiais, onde são matriculados tanto alunos quanto professores.

- As descrições dos dados não são informações do modelo – Sistemas baseados nos modelos clássicos não são capazes de responder a perguntas, cujas respostas sejam nomes de campos ou tipos de registros. Segundo Kent, as descrições dos dados, incluindo-se as restrições, deveriam ser armazenadas como informações ordinárias e modificações nos dados descritivos causariam mudanças automáticas no comportamento do sistema. Entretanto, vale ressaltar, que esta não é uma crítica apropiada ao sistemas relacionais.

- Fraca correlação entre entidade e registro – Quase sempre não é possível uma correspondência 1:1 entre uma entidade e um tipo de registro. No modelo relacional, no caso de uma entidade possuir somente atributos multivalorados, pode-se chegar ao caso extremo de não termos sequer um tipo de registro associado à mesma.

- Relacionamentos estão sujeitos a várias representações – Exceto os relacionamentos do tipo N:M, que devem ser representados por um tipo de registro, todos os outros, dos tipos 1:1 e 1:N, estão sujeito a várias formas de representação.

- Referências simbólicas não são satisfatórias – Alguns dos problemas associados a esse tipo de referência:

  a. Referências simbólicas permitem referências a entidades inexistentes.

  b. Sinônimos não são permitidos. Isto torna difícil detectar diferentes referências a um mesmo registro.

  c. Identificadores não são imutáveis. Eles estão sujeitos às constantes modificações de uma aplicação.

  d. A extensão[6] de um tipo de entidade afeta a escolha do identificador. Um fato utilizado para a identificação de ocorrências de uma entidade

6. Extensão é o grupo de instâncias que compõe uma entidade.

pode deixar de ser apropriado quando esse conjunto de ocorrências torna-se maior.

- Registros não descrevem a sua estrutura semântica – Um determinado fato pode estar registrado em vários campos de um registro ou um determinado campo pode descrever fatos diferentes nas diversas ocorrências de um registro. No exemplo acima, na representação do relacionamento matrícula em cursos especiais, poder-se-ia ter dois campos, um para professores e outro para alunos, com a função única de identificar um determinado tipo de assistente.

As críticas feitas por Kent foram descritas através de uma abordagem funcional das limitações de representação desses modelos. Um outro enfoque possível seria o da incapacidade de representação semântica dos mesmos. Como ele próprio admite, o seu trabalho objetivou auxiliar a proposta de modelos alternativos que se sobrepusessem a essa série de problemas. Os modelos de dados semânticos, abordados no capítulo 3, são uma coleção de modelos alternativos que possuem um objetivo em comum: tentar ser mais efetivo na tarefa de incorporar as propriedades de uma aplicação.

Capítulo 3

Modelos Semânticos

3.1 Introdução

Um sistema de informação, nada mais é que um modelo de uma determinada parte do mundo real. Por modelarem a concepção que usuários possuem do domínio da aplicação, sistemas de informação são considerados modelos conceituais e, portanto, quanto mais próxima for a sua representação desta concepção mais preciso ele se tornará[BORG83].

Nos modelos clássicos, e principalmente no modelo relacional, a sobrecarga semântica da estrutura de registro impossibilita uma representação mais significativa. Nesses modelos, como mostra o capítulo anterior, o usuário está restrito a esta estrutura para representar qualquer inter-relacionamento entre dados[KENT79]. Por essa relativa simplicidade, os modelos clássicos não são capazes de incorporar e exprimir grande parte da semântica de uma aplicação.

Com base nesta constatação, uma série de modelos de dados, denominados de semânticos, desenvolveram uma gama de mecanismos que propiciaram uma maior captação das propriedades de uma aplicação. A princípio, esses modelos enfocaram primordialmente as propriedades estruturais.

Enfatização da separação conceitual entre os componentes físicos e lógicos, diminuição da sobrecarga semântica e disponibilidade de mecanismos convenientes de abstração foram citados por Hull e King[HULL87] como principais vantagens desses modelos em relação aos modelos tradicionais.

3.2 Principais Abstrações

Influenciado por pesquisas paralelas em inteligência artificial, mais especificamente na área de redes semânticas[SMIT77b], um dos principais mecanismos desenvolvidos pelos modelos semânticos foi a abstração[7] de dados, fundamental para uma representação do domínio da aplicação mais adequada e próxima à concepção do usuário. As abstrações tentam corresponder aos relacionamentos de dados que comumente aparecem nas aplicações de banco de dados. A seguir, serão descritas as principais abstrações encontradas nos modelos semânticos.

3.2.1 Classificação

Classificação é uma forma de abstração na qual um grupo de objetos é considerado um objeto de classe de mais alto nível[STEF84]. O mecanismo de classificação está ligado à definição de tipos em um modelo semântico. Uma classe descreve as propriedades comuns a todos os objetos participantes.

No contexto de uma universidade, por exemplo, a classe `Aluno` poderia ser formada pelo conjunto de alunos em situação legal, ou seja, matriculados ou afastados de forma regular. Como mostra a figura 3.1, nome e registro acadêmico poderiam ser alguns dos atributos pertencentes aos objetos desta classe. O mecanismo de classificação representa o relacionamento é uma instância de: João é uma instância de `Aluno`.

7. Entende-se por abstração a supressão de detalhes irrelevantes e o realce de propriedades essenciais no interesse de uma determinada visão.

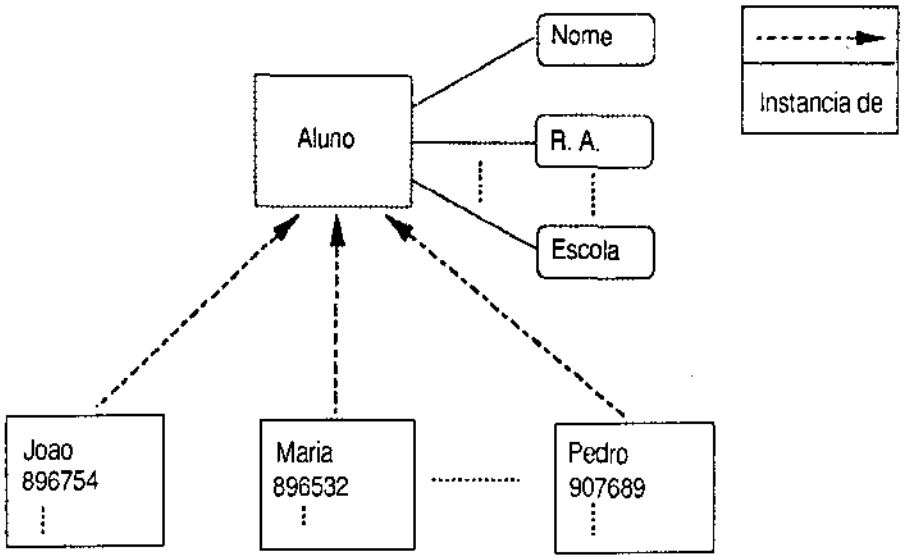


Figura 3.1: Classificação

Em alguns modelos[MYLO80], classes são também consideradas objetos, podendo possuir os seus próprios atributos. As classes que descrevem outras classes são denominadas de meta-classes. Os atributos de uma meta-classe representam as propriedades do grupo de objetos que constituem as classes. Percentagens de alunos do sexo masculino e feminino poderiam ser alguns dos atributos associados à meta-classe de `Aluno`.

3.2.2 Generalização

Generalização[SMIT77b] é a abstração na qual as diferenças entre classes de objetos similares são desprezadas, a fim de se construir uma nova classe na qual são enfatizadas as propriedades comuns.

A classe `Professor` poderia ser generalizada a partir das classes `Professor-Assistente`, `Professor-Adjunto` e `Professor-Titular`. Titulação e Departamento seriam dois, de um conjunto de atributos comuns a essas classes. A figura 3.2, exibe um diagrama dessa abstração.

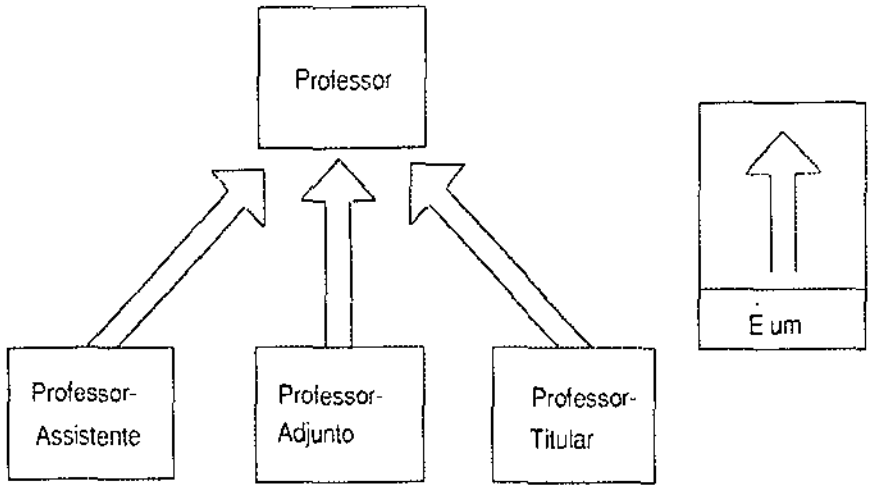


Figura 3.2: Generalização

Todavia, na maioria dos modelos semânticos encontra-se o mecanismo de especialização que é o inverso da generalização: a partir de uma determinada classe base, denominada de superclasse, deriva-se uma outra classe, denominada de subclasse, que, além dos seus próprios atributos, herdará os atributos da superclasse.

A herança múltipla é uma flexibilização do mecanismo de especialização, na qual é permitida que uma subclasse seja derivada a partir de duas ou mais superclasses. Como mostra a figura 3.3, a classe `Monitor` é derivada a partir das classes `Funcionário` e `Aluno`.

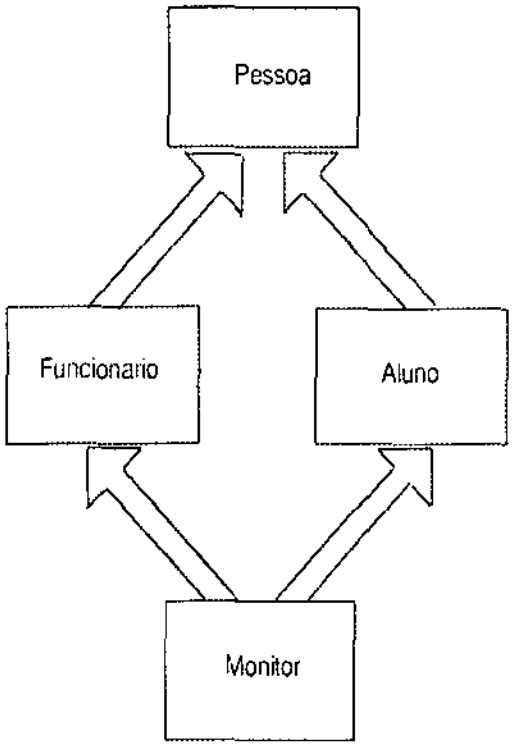


Figura 3.3: Herança Múltipla

Em um esquema semântico, uma seqüência de derivações de classes a partir de outras classes forma um grafo que pode ou não, a depender da permissão de herança múltipla, ser direto e acíclico. Erroneamente, o termo mais encontrado na literatura para denominar este grafo é hierarquia de generalizações. A própria figura 3.3 é um contra-exemplo, e, por isso, o termo mais apropriado é rede de generalizações.

Generalização e, consequentemente, especialização, implementam o relacionamento é um: `Professor-Adjunto` é um `Professor`. Generalização está ligada ao mecanismo de derivação de tipos. Na literatura encontra-se até quatro variações deste relacionamento:

- **Sobreposta** – Permite que subclasses possuam objetos em comum, sem que seja necessária a participação de cada objeto da superclasse em uma subclasse.

- **Disjunta** – Define subclasses que não podem possuir objetos em comum, sem que seja necessária a participação de cada objeto da superclasse em uma subclasse.

- **Coberta** – Especifica subclasses onde cada objeto da superclasse deve pertencer a pelo menos uma subclasse. Na especialização da classe `Aluno` em `Aluno-Graduação`, `Aluno-Pós` e `Aluno-Especial` esta forma de relacionamento pode ser empregada para modelar o fato de que é permitido a qualquer aluno efetuar uma graduação e uma pós-graduação ao mesmo tempo.

- **Particionada** – Especifica subclasses onde cada objeto da superclasse deve pertencer a uma e somente uma subclasse. Na figura 3.2, cada objeto da classe `Professor`, deve pertencer a um única subclasse, seja ela `Professor-Assistente`, `Professor-Adjunto` ou `Professor-Titular`.

Cada forma do relacionamento implica em semânticas diferentes para uma mesma atualização. Na retirada de um objeto de uma subclasse nas formas sobreposta e disjunta, a atualização não deve ser propagada para a superclasse. Na forma particionada deve haver propagação, e na forma coberta dependerá da existência do objeto em outra subclasse situada no mesmo nível da classe em que se requisitou a retirada.

3.2.3 Agregação

Agregação[SMIT77a] é a abstração pela qual um relacionamento entre classes diversas é considerado uma classe de nível mais alto. Como mostra a figura 3.4, o relacionamento entre os objetos das classes `Aluno`, `Turma` e `Semestre`, é abstraído como a agregação `Matrícula`. Através dessa abstração, torna-se possível pensar em `Matrícula`, desprezando-se detalhes das partes componentes.

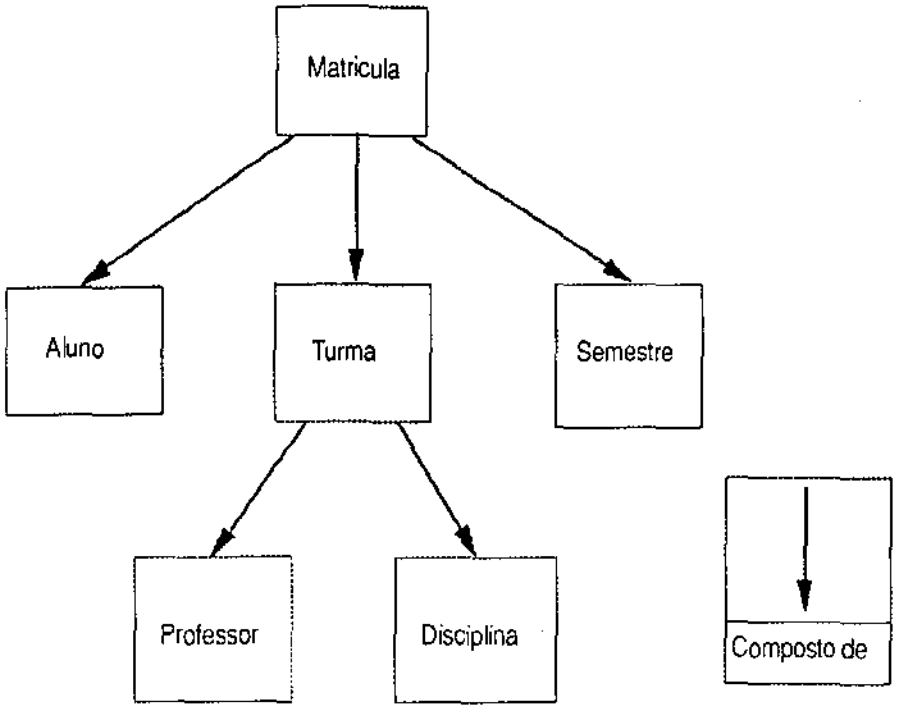


Figura 3.4: Agregação

Agregação está relacionada com o mecanismo de construção de tipos em um modelo semântico. Agregação representa o relacionamento é composto por. `Matrícula` é composta por `Aluno`, `Semestre` e `Turma`. Por sua vez, uma agregação poderá ter associada a si atributos que a descrevem. A média final poderia ser um dos possíveis atributos para `Matrícula`. Além dos seus próprios atributos, seguindo a hierarquia de composição de uma agregação em uma LMD de um modelo semântico, deve-se ter acesso aos atributos das classes componentes[ZANI83].

O mecanismo de agregação impõe algumas propriedades que devem ser observadas nas atualizações de objetos de uma hierarquia[SMIT77a]. Uma delas é o condicionamento da existência da ocorrência de um objeto na agregação à existência de todos os objetos componentes. Paradoxalmente, essa propriedade pode não ser válida para todos os relacionamentos que devam ser representados por uma agregação.

Na figura 3.4, a classe `Turma` é definida a partir da agregação das classes `Disciplina` e `Professor`. Sendo assim, a exclusão de uma disciplina deve

ocasionar a exclusão de todas as ocorrências dos objetos de Turma que têm como componente o objeto de Disciplina excluído. Já a demissão de um professor e a conseqüente exclusão deste, pode significar apenas a ausência temporária de um professor nas turmas em que o professor excluído lecionava, e, portanto, os objetos compostos da classe Turma não devem ser excluídos. Em resumo, o relacionamento é composto de pode assumir formas diferentes, implicando em semânticas diferentes de uma mesma atualização. Vale notar, que a atualização de um objeto componente pode provocar uma seqüência de atualizações hierarquia acima.

3.2.4 Associação

Associação[BROD83] é uma forma de abstração na qual uma coleção de objetos membros, pertencentes a uma determinada classe base, é considerada um objeto de grupo de nível mais alto. Esta abstração permite que os detalhes dos objetos membros sejam ignorados quando se considera o grupo. Associação representa o relacionamento é membro de.

Associação está ligada ao mecanismo de construção de tipos em um modelo semântico. Essa abstração é usada para modelar grupos de objetos de uma determinada classe. Como mostra a figura 3.5, a representação dos grupos de estudo aos quais os alunos podem se associar é feita através de uma associação. À classe Aluno pertecem os objetos que serão os membros dos objetos de grupo da associação Grupo de Estudo.

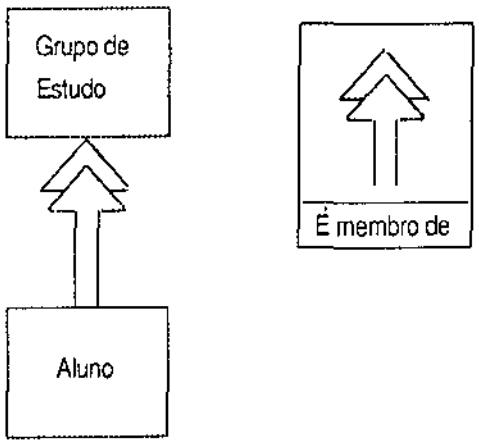


Figura 3.5: Associação

Os atributos conectados a uma associação irão descrever as propriedades pertinentes aos grupos. Área de interesse e orientador de grupo, poderiam ser dois possíves atributos da associação `Grupo de Estudo`.

O mecanismo de associação pode ser empregado de forma recorrente, ou seja, um grupo pode ter como membros outros grupos. Cada grupo relacionado por uma associação é uma subclasse da classe de origem. Porém, enquanto uma associação descreve o grupo, uma especialização descreve os objetos que compõem um grupo.

Apesar de ser a mais natural, associação não é a única forma de se modelar as situações típicas de grupos em uma classe. Alguns modelos optam por oferecer atributos multivalorados, ou até mesmo, na falta deste mecanismo, a modelagem dessa situação através de uma agregação. A figura 3.6 mostra como os grupos de estudo são modelados através de uma agregação, uma forma de representação típica do modelo E-R a ser visto.

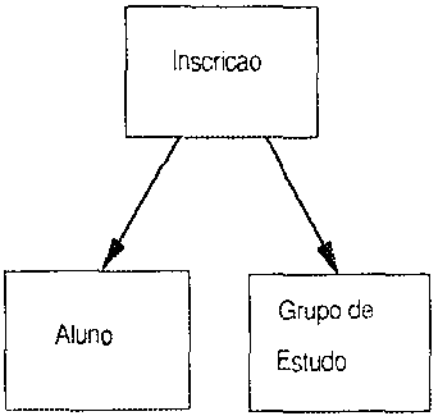


Figura 3.6: Modelagem de grupos com o uso de agregação

3.3 Representação das Abstrações

Os diversos modelos semânticos diferem tanto na forma quanto nas abstrações disponíveis, além do que, alguns tendem a enfatizar um determinado mecanismo de abstração em detrimento de outros: TAXIS[MYLO80] enfatiza sobremaneira o mecanismo de especialização, enquanto SDM[HAMM81], o faz com a classificação.

Como observaram Hull e King[HULL87], a representação das abstrações nesses modelos seguem duas abordagens básicas, uma utilizando-se de construtores específicos e a outra de atributos. As figuras 3.7 e 3.8 exibem a representação da agregação `Empréstimo`, sendo que a primeira pelo modelo Entidade-Relacionamento (E-R)[CHEN76] e a segunda pelo modelo funcional[SHIP81].

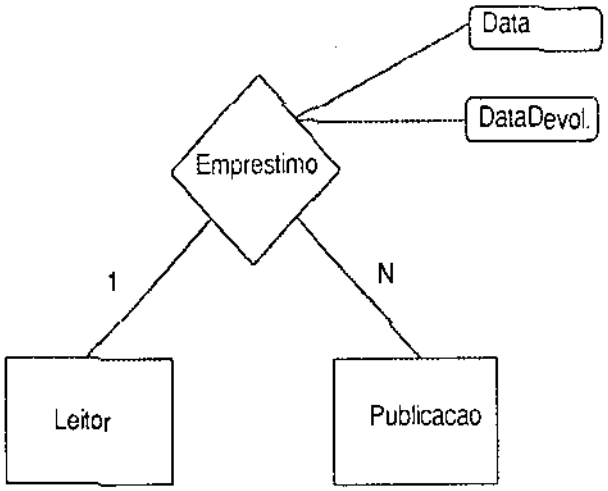


Figura 3.7: Agregação `Empréstimo` pelo modelo E-R

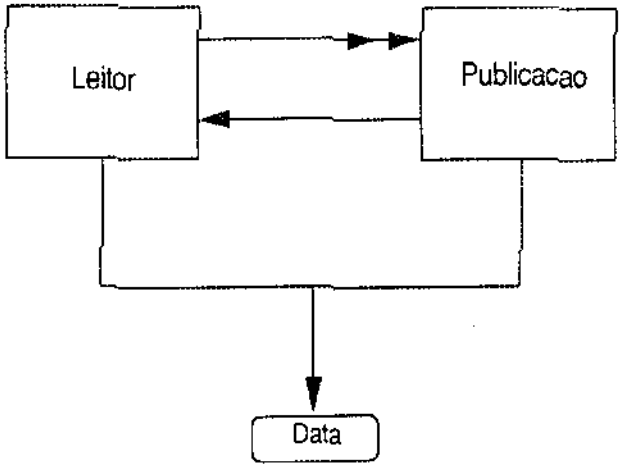


Figura 3.8: Agregação `Empréstimo` pelo modelo Funcional

O modelo E-R dispõe de um construtor específico para a abstração, o que lhe permite que atributos e restrições de integridades sejam associadas diretamente à mesma. Já o modelo funcional a representa através de atributos. Um atributo multivalorado, que tem como domínio a classe `Leitor` e como contra-domínio a classe `Publicação`, implementa um sentido do relacionameto. As informações diretamente associadas à agregação são representadas através de atributos que têm um domínio composto das classes `Leitor` e `Publicação`.

Modelos diferentes podem produzir esquemas radicalmente diferentes para uma mesma aplicação. De uma certa forma, a escolha da abordagem adotada funciona como um "divisor de águas" entre os modelos semânticos, afetando inclusive os mecanimos das LMD associadas aos modelos.

3.4 Outros Mecanismos

Sem dúvida alguma, as abstrações de dados apresentadas são meios efetivos na captação das propriedades e relacionamentos existentes em uma aplicação. Existem, entretanto, outras propriedades que não são satisfatoriamente incorporadas por essa abordagem estrutural. Por isso, os modelos semânticos se valem de outros recursos que incrementam e contribuem para a captação dessas.

Restrições de integridade são empregadas para a manutenção da coerência das informações contidas em um banco de dados. Algumas delas são implicitamente incorporadas pelas próprias abstrações. Por exemplo, um objeto em uma agregação depende da existência dos objetos componentes. Como visto na figura 3.7, a cardinalidade de um relacionamento regula o comportamento da própria abstração: uma publicação só pode ser emprestada, por vez, a um leitor, mas um leitor ter várias publicações em empréstimo. Outras restrições são de caráter computacional e limitam os valores que um determinado atributo pode assumir.

Um outro mecanismo de abstração bastante usado é a derivação de dados. Com base nas informações contidas em um esquema, um modelo semântico pode derivar atributos ou mesmo classes. Derivações de dados são baseadas em predicados. Na figura 3.9, `Bolsista` é uma subclasse de `Aluno` derivada a partir do predicado média escolar maior que nove e meio. É claro que os componentes derivados são sensíveis as modificações que ocorrem no banco de dados: ao final

de um semestre, se um aluno bolsista não conseguir manter a média escolar acima de nove e meio, o SGBD automaticamente o retira da subclasse `Bolsista`.

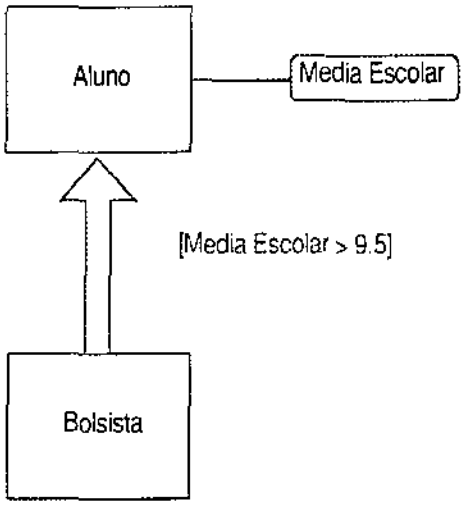


Figura 3.9: Derivação de dados

3.5 Aspectos de Modelagem

Uma das conseqüências imediatas da adoção de modelos semânticos, foi a facilitação do desenvolvimento de projetos lógicos de esquemas. As abstrações de dados disponíveis ocasionaram o fenômeno da localização, que estimula o projeto das propriedades de cada objeto isoladamente, e de forma coerente, uma integração posterior[BROD84]. Sendo assim, o processo de modelagem por um modelo semântico torna-se *top-down* e modular, inverso ao processo *bottom-up* oferecido pelo modelo relacional.

Por isso, e devido à ausência quase que completa de SGBDs semânticos, esses modelos estão sendo adotados como ferramenta auxiliar ao projeto lógico de esquemas relacionais. O usuário expressa a estrutura de sua aplicação em um esquema semântico e, através de um grupo de regras, transforma-o em um esquema relacional[TEOR86].

Alguns modelos utilizam-se das mesmas abstrações para efetuar a modelagem comportamental. Nesses modelos, transações complexas são construídas a partir de transações mais simples e/ou operações básicas seguindo as hierarquias de generalização, agregação e associação. Um exemplo dessa abordagem é o SHM+[BROD83].

Os modelos semânticos impõem restrições a serem observadas na confecção de um esquema. Algumas dessas restrições possuem um caráter extremamente intuitivo. A figura 3.10 exibe um esquema ilegal onde ocorre um ciclo em uma rede de generalização. Outras restrições são particulares a cada modelo. Por exemplo, o modelo E-R só permite que os atributos tenham como contra-domínio tipos atômicos e, em algumas extensões, multivalorados.

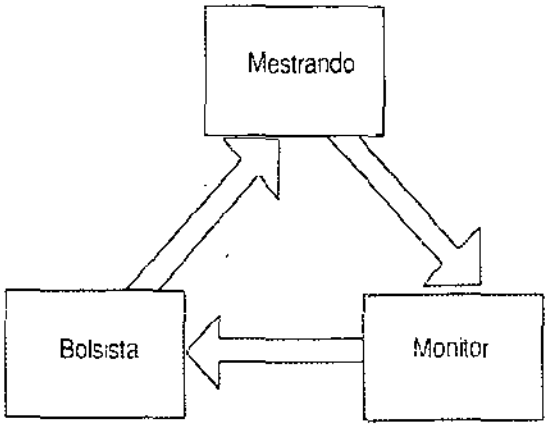


Figura 3.10: Esquema ilegal

3.6 Modelos Representativos

O objetivo desta seção é o de ilustrar os conceitos vistos acima, através da descrição sumária de alguns dos principais modelos semânticos existentes na literatura. Está fora do escopo desta qualquer comparação entre os modelos, mesmo porque, devido a falta de uma definição formal, os critérios de comparação

tendem a ser bastante subjetivos[BROD84]. O leitor interessado nessa abordagem deve procurar [HULL87, PECK88]. Da mesma forma que no capítulo 1, a capacidade de representação destes modelos, ou, quando possível, de expressividade das suas linguagens de consultas, serão primordialmente enfocadas em detrimento dos aspectos de implementação.

3.6.1 Entidade-Relacionamento

O modelo E-R[CHEN76] propõe conceitos razoavelmente simples, fato preponderante para a sua ampla difusão. O esquema de uma aplicação pelo modelo E-R, é uma rede composta por entidades e relacionamentos entre essas entidades. Na verdade, um relacionamento no modelo E-R restringe-se a representar a abstração de agregação. Atributos monovalorados podem ser associados às entidades e relacionamentos.

A figura 3.11 exibe um diagrama E-R que representa relacionamentos típicos encontrado em um sistema acadêmico. Retângulos representam entidades e losangos relacionamentos. O modelo E-R suporta algumas restrições de integridade não encontradas no modelo relacional, como entidades dependentes[8] e cardinalidade de relacionamentos. Em um diagrama E-R, a cardinalidade é representada através do assinalamento do caráter `1` ou `N` na linha que conecta o relacionamento às entidades. Já a entidade dependente, é representada por um retângulo em linha dupla.

8. Denominadas de entidades fracas no modelo E-R.

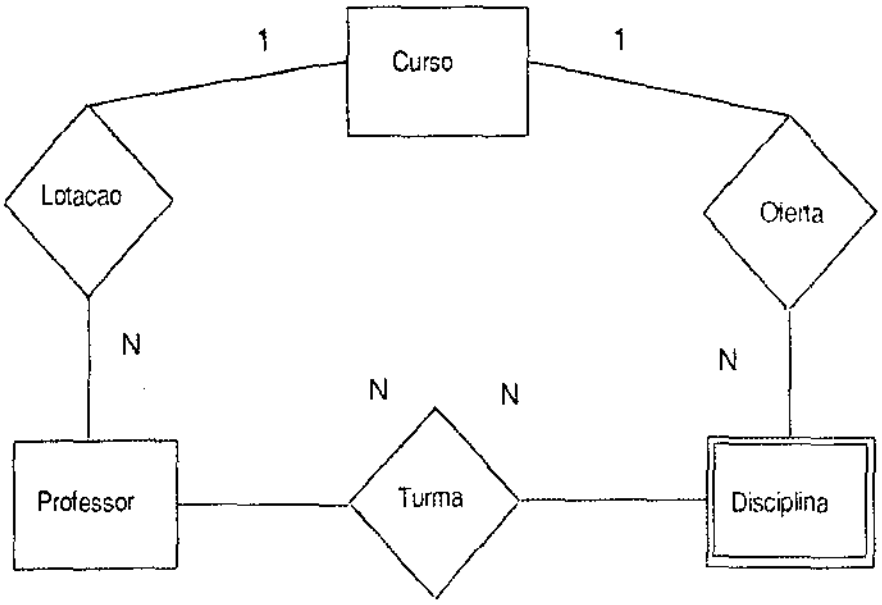


Figura 3.11: Modelo Entidade-Relacionamento

Não é permitido que o contra-domínio de um atributo seja uma outra entidade, situação que deve ser modelada através de um relacionamento. Na forma original, como proposto por Chen, o modelo E-R não oferece a abstração de generalização, tendo sido feitas algumas tentativas no sentido dessa extensão[TEOR86, WAGN87]. Associação, como visto na subseção 3.2.4, pode ser razoavelmente representada através de um relacionamento E-R.

Apesar de sua larga aceitação como ferramenta de modelagem, nenhum SGBD comercial implementa o modelo E-R e o seu principal emprego vem sendo o de ferramenta auxiliar ao projeto lógico de esquemas relacionais. Alguns SGBDs relacionais oferecem utilitários que automatizam o processo de tradução[ROGE87].

Várias são as justificativas para o relativo insucesso obtido pelo modelo E-R. O não surgimento de uma linguagem de consulta e manipulação de dados padrão associada ao modelo exerceu alguma influência. Stonebraker[STON88] alega outros motivos:

- À época do seu surgimento, a comunidade estava direcionada para o debate entre os modelos DBTG/CODASYL e relacional.

- O modelo E-R não ofereceu vantagens dramáticas, da maneira que o modelo relacional impôs ao seu predecessor. Segundo ele, o modelo E-R não expressa naturalmente relacionamentos não binários, além de tender a suportar linguagens de consulta que só efetuam conexões entre entidades que possuam entre si relacionamentos.

3.6.2 Funcional

Um dos objetivos do modelo funcional (FDM) foi prover uma linguagem de definição e manipulação simples e concisa, mas expressivamente poderosa, formada basicamente pelos construtores entidade e função. Associada ao modelo, a linguagem DAPLEX[SHIP81] tenta ser uma interface conceitualmente natural utilizando-se dos mesmos construtores.

Funções representam todo e qualquer relacionamento. Uma função pode exercer um papel de atributo, ou mesmo indicar a especialização de uma classe em outra. Em geral, ela tem como domínio uma ou mais entidades e como contra-domínio uma entidade ou um tipo atômico simples ou multivalorado.

Funções com múltiplos argumentos representam agregações, evitando-se, dessa forma, a criação de novas entidades para tal. Associação pode ser representada por uma função multivalorada[9], enquanto que uma generalização é expressa pela indicação da superclasse como imagem de uma função sem argumento que, por convenção, define uma entidade. As declarações DAPLEX da figura 3.12 exemplificam os vários empregos do construtor função.

9. Denotada por ==>>

```
Pessoa( ) ==>> ENTITY
DECLARE Nome(Pessoa) ==> STRING
DECLARE RG(Pessoa) ==> INTEGER
...

DECLARE Aluno( ) ==>> Pessoa
DECLARE RA(Aluno) ==> INTEGER
DECLARE Curso(Aluno) ==> Curso
DECLARE DataIngresso(Aluno) ==> STRING
DECLARE Turmas(Aluno) ==>> Turma
...

DECLARE Conceito(Aluno,Turmas(Aluno)) ==> INTEGER
DECLARE Faltas(Aluno,Turmas(Aluno)) ==> INTEGER
...

DECLARE Curso( ) ==>> ENTITY
DECLARE Nome(Curso) ==> STRING
DECLARE Departamento(Curso) ==> Departamento
DECLARE Vagas(Curso) ==> INTEGER
DECLARE Coordenador(Curso) ==> Professor
...
```

Figura 3.12: Descrição de dados em DAPLEX

Uma característica marcante do modelo funcional é a sua versatilidade na derivação de dados. Da forma como foi descrito o esquema acima, tem-se a impressão que o relacionamento entre `Aluno` e `Curso` ocorre em um único sentido. Entretanto, a declaração `DEFINE Alunos(Curso) ==>> INVERSE OF Curso(Aluno)` é suficiente para definir o outro sentido do relacionamento. Para permitir uma visão mais natural do esquema, o FDM possue uma variedade de facilidades para a definição de atributos derivados. Por exemplo, `DEFINE Professores(Aluno) ==>> Professor(Turmas(Aluno))` pode ser uma declaração bastante útil, caso se torne necessário considerar os objetos da classe `Professor` como uma propriedade diretamente associada à entidade `Aluno`.

3.6.3 SDM

Contrariamente, o SDM[HAMM81] incorporou uma grande variedade de primitivas de construção e derivação de dados ao seu modelo, de forma a permitir que, para cada propriedade existente em uma aplicação, encontre-se uma primitiva adequada, capaz de mapeá-la diretamente.

Um esquema SDM é uma coleção de entidades organizadas em classes, onde são especificados os atributos dos membros e das classes, as conexões entre as classes e os atributos derivados. Como no modelo funcional, não existem construtores explícitos para representar os diversos relacionamentos. Todavia, dois tipos de conexões entre classes, uma destinada a representar o mecanismo de generalização e a outra o de associação, estão disponíveis.

A conexão destinada a definir subclasses em SDM é determinada a partir de um predicado que a conecta às superclasses envolvidas. Quatro formas de especificação do predicado estão disponíveis:

- Baseada nos valores dos atributos da superclasse;
- Definida por operações de conjunto, como união e interseção, entre superclasses derivadas a partir de uma mesma classe base;
- Baseada no atributo de uma classe que tem a superclasse como contra-domínio;
- Controlada pelo usuário, ou seja, a qualificação da participação dos objetos na subclasse não é especificada, e as inclusões e retiradas ficam sob a responsabilidade do próprio usuário.

A figura 3.13 exemplifica as várias formas de conexão, respectivamente.

```
PROFESSOR
  interclass conection: subclass of FUNCIONÁRIO
    where cargo = 'PROFESSOR'
  description: todos os professores da universidade
  member attributes:
    Depto
      value class: DEPARTAMENTO
    ...

PROFESSOR_TEC_ADM
  interclass conection: subclass of FUNCIONÁRIO
    where is in PROFESSOR and is in TEC_ADM
  description: todos os professores que também
    são funcionários técnico-administrativos.
  member attributes:
    Carga_Horária_Dispensada
      value class: INTEGER
    ...

PROFESSOR_PESQUISADOR
  interclass conection: subclass of PROFESSOR where
    is a value of Pesquisador of PESQUISA
  description: todos os professores que possuem uma
    pesquisa em curso.
  member attributes:
    Salário_Adicional
      value class: INTEGER
    ...

PROFESSOR_CHEFE_DEPTO
  interclass  conection: subclass of PROFESSOR where
    specified
  description: todos os professores que são chefes de
    departamento
  member attributes:
    Salário_Adicional
      value class: INTEGER
    ...
```

Figura 3.13: Definição de subclasses em SDM

Para a conexão que define uma associação, três formas de especificação são providas:

- Baseada no valor comum de um determinado atributo da classe base;
- A partir de um grupo de subclasses derivadas de uma mesma classe base;
- Controlada pelo próprio usuário.

A figura 3.14 exemplifica as formas dessa conexão.

```
CURSO_ALUNOS
  interclass conection: grouping of ALUNOS on common
    value of Curso
  description: conjuntos de alunos agrupados por curso
  member attributes:
    Total_Alunos
      value class: number of members in Contents
    ...

DOCENTES_DISCENTES
  interclass conection: grouping of PESSOA
    consisting of classes ALUNOS, PROFESSORES
  description: todas as pessoas envolvidas em atividades
    de ensino e pesquisa
    ...

GRUPOS_DE_ESTUDO
  interclass conection: grouping of ALUNOS as
    specified
  description: conjunto de grupos de estudos
    formados por alunos
  member attributes:
    Area_Interesse:
      value class: Área
    ...
```

Figura 3.14: Definição de grupos em SDM

Os atributos que descrevem os membros das classes de um esquema, podem ser inter-relacionados a partir das informações dos relacionamentos já descritos no esquema. A figura 3.15 exemplifica as duas formas básicas de inter-relacionamentos existentes. A primeira, de mesma semântica que o FDM, através de inversão. A segunda, através de combinação[10], que é uma inversão combinada ao retorno do valor de um atributo especificado. Estes inter-relacionamentos são, sem dúvida, meios flexíveis e elegantes de propiciar ao usuário, múltiplas visões de uma mesma informação. O relacionamento entre um aluno e uma turma pode ser reconhecido a partir de qualquer uma das classes da figura abaixo.

10. *Matching* na terminologia SDM.

```
MATRÍCULA
  description: todas as matrículas efetuadas no semestre
  member attributes:
    Aluno
      value class: ALUNO
    Turma
      value class: TURMA
    ...

ALUNO
  description: todos os alunos em estado regular
  member attributes:
    RA
      value class: INTEGER
    Matrícula
      value class: MATRÍCULA
      inverse: Aluno
    Turmas
      value class: TURMA
      matching: Turma of MATRÍCULA on Aluno
    ...

TURMA
  description: todas as turmas oferecidas
  member attributes:
    Professor
      value class: PROFESSOR
    Disciplinas
      value class: DISCIPLINAS
    Matrículas
      value class: MATRÍCULAS
      inverse: Turmas
    Alunos
      value class: ALUNO
      matching: Aluno of MATRÍCULA on Turma
    ...
```

Figura 3.15: Definição de inter-relacionamentos em SDM

Em resumo, a grande variedade de primitivas e a estrutura natural de sua sintaxe fazem o SDM habilitar-se como uma linguagem para especificação de esquemas. Entretanto, a sua complexidade o torna inapropriado para ser implementado por um SGBD.

3.6.4 RM/T

Reconhecendo a pobreza semântica de um esquema relacional, uma coleção de tabelas sem qualquer inter-relacionamento expresso, Codd propôs uma extensão

deste modelo denominada de RM/T|CODD79|. As extensões do modelo relacional foram desenvolvidas em duas direções principais: a incorporação à álgebra relacional do tratamento de valores nulos e a categorização de entidades, junto à criação de novas regras de integridade associadas às representações dessas.

O RM/T define os seguintes tipos de entidade:

- Básicas – Correspondem, quase sempre, às entidades concretas existentes na aplicação: aluno, professor, universidade, etc;
- Associativas – São as entidades, em geral abstratas, constituídas a partir da agregação de outras entidades;
- Descritivas – Entidades, cujo papel principal na aplicação, é o de descrever outras entidades, sejam elas básicas, associativas ou até mesmo descritivas.

O modelo RM/T reconheceu a necessidade da criação de uma identificação única para cada instância representada em um banco de dados independente dos valores dos seus atributos. Para esse fim, o RM/T associa a cada entidade um tipo de relação, denominada `Relação-E`, que contém os *surrogates*[11] das suas instâncias.

Um outro tipo de relação, denominada `Relação-P`, é empregada para descrever as propriedades relativas às entidades. O número de `Relações-P` associadas a uma entidade é livre, e as ligações entre uma `Relação-E` e as respectivas `Relações-P`, são implementadas através de *surrogates*. Uma das regras de integridade definidas pelo RM/T proíbe a existência de uma tupla em uma `Relação-P`, sem que haja o *surrogate* relativo a essa tupla na `Relação-E`.

As figuras 3.16 e 3.17 descrevem a representação em RM/T dos relacionamentos típicos existentes entre entidades envolvidas em um sistema de empréstimos de uma biblioteca. A figura 3.16 exibe as entidades básicas `Leitor` e `Publicação`.

---

11. *Surrogate* é uma identificação designada a cada instância, única em todo banco de dados, criada e manipulada exclusivamente pelo SGBD.

Vale notar que *surrogates* evitam a criação de chaves artificiais. No exemplo da figura 3.16, o ISBN é uma chave primária em potencial, mas dois exemplares de uma mesma publicação possuem o mesmo ISBN.

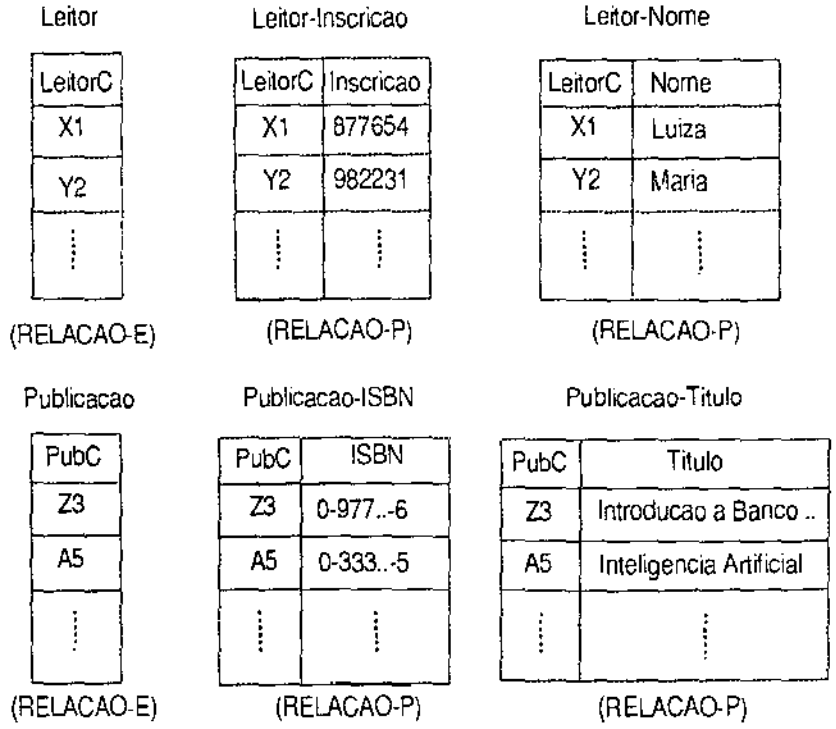


Figura 3.16: Representação RM/T das entidades `LEITOR` e `PUBLICAÇÃO`

A todos os tipos de entidades são associadas `Relações-E` e `Relações-P`. Todavia, para a representação de uma dependência multivalorada pode ser imperativo associar uma entidade a uma entidade descritiva. Da mesma forma, o RM/T associa à existência da instância de uma entidade descritiva à existência da instância da entidade que está sendo descrita.

Contrariamente, o modelo não associa a existência das ocorrências de uma entidade associativa às ocorrências das entidades que participam da agregação. A figura 3.17 mostra a representação da entidade associativa `Empréstimo` e da entidade descritiva `Autor`.

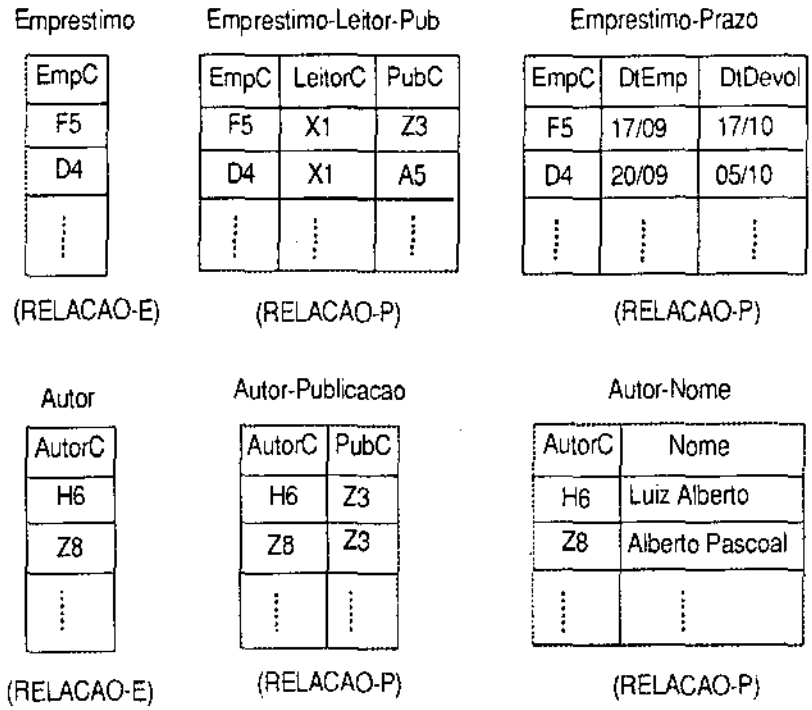


Figura 3.17: Representação RM/T das entidades EMPRÉSTIMO e AUTOR

O RM/T também permite outros inter-relacionamentos, como especialização, associação e precedência de eventos. Todos estes inter-relacionamentos são também assegurados por regras de integridade previamente definidas pelo modelo.

Grande parte da extensão semântica propiciada pelo RM/T foi efetuada pela simples extensão do dicionário de dados do modelo relacional, através da criação de algumas relações, que descrevem os inter-relacionamentos existentes, junto a novos operadores, denominados de operadores de grafo, que proporcionam uma manipulação conveniente dessas relações. Contudo, o RM/T é muito complexo se comparado ao modelo relacional.

3.6.5 GEM

O GEM[ZANI83] é uma linguagem de manipulação de dados para um modelo semântico do tipo E-R denominado de DSIS. O fato relevante é que o GEM é uma extensão da LMD relacional QUEL. Um dos resultados primordiais do GEM foi mostrar que, com extensões simples, uma linguagem relacional pode

se adequar a um modelo semântico, sem necessariamente sacrificar a sua facilidade de uso ou poder de consulta.

Como mostra a figura 3.18, um esquema GEM é bastante simples, assemelhando-se às declarações de relações em um SGBD relacional. As principais extensões feitas foram *surrogates*, agregação, generalização, valores nulos e atributos multivalorados.

```
DEPARTAMENTO(Nome:c, Chefe:PROFESSOR, Área:i2)
              key(DeptoNome);

CURSO(Nome:c, Coordenador:PROFESSOR, Depto:DEPARTAMENTO)
              key(Nome);

DISCIPLINA(Nome:c, Curso:CURSO, CargaHorária:i2)
              key(Nome);

TURMA(Número:i2, Prof:PROFESSOR, Discip:DISCIPLINA)
              key(Discip, Número);

MATRÍCULA(Aluno:ALUNO, Turma:TURMA, Conceito:i2, Faltas:i2)
              key(Aluno,Turma);
...
FUNCIONÁRIO(Nome:c, RG:c, CPF:c, Sexo:c, Reservista:c null
             allowed, Idiomas:{c}, [TEC_ADM(Estat/CLT:i2,Cargo:c),
             PROFESSOR(Titulação:i2, Universidade:c) ])
              key(Nome), key(RG);
...
```

Figura 3.18: Esquema em GEM

Generalizações são expressas através de declarações de relações internas à declaração de uma relação superclasse. No exemplo da figura 3.18, TEC_ADM e PROFESSOR são sub-relações da relação FUNCIONÁRIO.

Agregações são especificadas através das declarações de atributos nos quais os contra-domínios são entidades descritas no esquema. Precisamente, esses atributos possuem como valor o *surrogate* da entidade a qual se faz referência.

Vale observar que, apesar de TURMA ser uma agregação entre PROFESSOR e DISCIPLINA, o atributo Prof não participa da chave da relação, já que é permitido a um professor, ministrar uma mesma disciplina em diversas turmas. Atributos multivalorados são especificados por um tipo atômico entre chaves.

O GEM facilita sobremaneira a construção de uma consulta cuja qualificação utiliza-se de atributos pertencentes a outras relações. Isto é feito através de *surrogates* conjuntamente à notação de ponto, que dispensa o usuário de explicitar as junções necessárias a este tipo de consulta, ao passo que oferece uma disposição funcional à linguagem. Uma consulta simples que lista todos os cursos da área 1 é implementada através do comando GEM abaixo, cuja sintaxe é bem mais simples que o seu equivalente QUEL.

GEM

```
retrieve(Curso.Nome)
where(Curso.Depto.Área = 1)
```

QUEL

```
range of Curso is Curso
range of Dept is Dept
retrieve (Curso.Nome)
where (Curso.Dept = Dept.#)
      and (Dept.Área = 1)
```

Figura 3.19: Comando GEM que utiliza a notação de ponto

Generalizações, valores nulos e atributos multivalorados são facilmente manipulados através do GEM. Dentre uma série de facilidades, ressalta-se o operador is, usado para verificar se a instância de uma classe pertence a uma subclasse desta, e os operadores de conjuntos, que simplificam sobremaneira as consultas baseadas em atributos multivalorados. A figura 3.20 exemplifica esses conceitos através de uma consulta que lista o nome de todos os funcionários que são professores e técnico-administrativos, e que têm como um dos seus idiomas o alemão.

```
retrieve (Funcionário.Nome)
where ("ALEMÃO" ∈ Funcionário.Línguas  and
       Funcionário is Professor and
       Funcionário is Tec-Adm)
```

Figura 3.20: Comando GEM que utiliza os operadores is e ∈

3.6.6 POSTGRES

Apesar de não ser tão conciso e uniforme quanto os modelos RM/T e GEM, POSTGRES[ROWE87] talvez seja a mais completa extensão do modelo relacional. Através dos mecanismos de tipo abstratos de dados (TAD), tipos de dados procedimento e herança, o modelo simula uma variedade de construções semânticas como agregação, generalização e atributos derivados. Algumas características quase que exclusivas aos modelos de dados orientados a objetos, como a associação de procedimentos às relações e o acesso a estados passados do banco de dados, são também oferecidas pelo POSTGRES.

A figura 3.21 exibe um esquema POSTGRES. Como o GEM, a LDD do POSTGRES é relativamente simples, assemelhando-se à de um SGBD relacional. A definição de uma relação pode ser feita a partir da especialização de outras e, da mesma forma que nos modelos orientados a objetos, uma relação não só herda os atributos, bem como os procedimentos associados à relação especializada. Uma chave primária pode ser definida para cada relação, sendo que o sistema de armazenamento do POSTGRES também designa a cada tupla um *surrogate* a ser armazenado no atributo "invisível" OID[12].

TAD propicia a extensão do conjunto de tipos atômicos oferecido pelo modelo. Para definição de um TAD, como o tipo DATA abaixo, são necessárias as especificações do seu nome, tamanho da representação interna, procedimentos de conversão da representação interna para externa e vice-versa, e um valor padrão. Para a definição dos operadores inerentes a um TAD, além das informações semânticas, como precedência e associatividade, é necessária a definição de informações empregadas pelo POSTGRES para a construção de índices e otimização de consultas a partir dos mesmos[STON83].

O modelo também oferece construtores para tipos estruturados como vetores. Na relação PERIÓDICO, o atributo Número é definido como um vetor de inteiros, sem limite superior, que indica a quantidade de exemplares dos diversos números de um periódico da bilioteca. Um segundo tipo estruturado permite que os atributos assumam valores do tipo procedimento. Os valores de um tipo procedimento são constituídos a partir dos comandos da linguagem de consulta POSTQUEL. Ao se

12. *Object Identification.*

fazer referência a um atributo deste tipo, o SGBD se encarregará de executar a consulta contida no mesmo[STON84] e por conseqüência, pode-se dizer que o valor associado ao atributo é uma relação. As tuplas resultantes da execução são manipuladas através da mesma notação de ponto adotada pelo GEM.

```
create LEITOR  (Inscrição=char[6], DataSuspensão=DATA,
                DiasSuspensão=int2, ÁreaInteresse=POSTQUEL,
                Pendência=PENDÊNCIAS )  key (Inscrição);

create PUB (Título=char[40], DataAquisição=DATA,
            Área=ÁREA, Quantidade=int2);

create LIVRO (ISBN=char[13], Autores=POSTQUEL,
              Edição=int2)  inherits(PUB);

create PERIÓDICO (Editor=char[40], Número=int2[]) inherits(PUB);

create ÁREA(Nome=char[40]);

create EMPRÉSTIMO (Inscrição=char[6], PubOID=PUB, DataDev=DATA);

create AUTORES(Nome=char[20], Sobrenome=char[20]);

define type DATA(InternalLength=6, InputProc=ChartoData,
                 OutputProc=DatatoChar, Default="010180");

define type PENDÊNCIAS is
   retrieve (Título=P.Título)
   from E in EMPRÉSTIMO
        P in PUB
   where E.Inscrição=$.Inscrição and
         DataDev < TODAY and
         E.PubOID=P.OID
end

define type ÁREA(int4) is
   retrieve (ÁREA.all)
   where ÁREA.OID = $1
end

define procedure SUSPENSO(LEITOR) returns
     bool is (language="C", filename="AindaSus")
...
```

Figura 3.21: Esquema POSTGRES

Duas categorias de construtores do tipo procedimento estão disponíveis: variável e parametrizada. O tipo procedimento variável é indicado através da palavra chave POSTQUEL e os valores contidos nesses atributos são consultas POSTQUEL diversas. Já o tipo procedimento parametrizado, é definido através de uma única consulta a ser empregada em todas as tuplas de uma relação, diferenciada apenas nos valores assumidos pelos parâmetros da consulta que são obtidos a partir dos próprios atributos da tupla. PENDÊNCIAS, na figura acima, é um tipo procedimento parametrizado que obtém todos os empréstimos de um LEITOR, cujas as devoluções se encontram em atraso[13].

Procedimentos podem ou não estar associados a uma relação. De uma forma geral, eles transferem algum tipo de processamento do escopo da aplicação para o SGBD. Um procedimento é definido para o sistema especificando-se nome, tipos dos argumentos, tipo do valor retornado, a linguagem em que foi escrito e o arquivo de armazenamento do código objeto. Os argumentos são especificados com base nos tipos definidos pelo POSTGRES. Um procedimento é associado a uma relação quando o mesmo possui um único parâmetro e o seu tipo corresponde a uma variável-tupla de mesmo nome que a relação associada. Na figura 3.21, o procedimento SUSPENSO é um procedimento associado à relação LEITOR.

3.6.7 TAXIS

Dentre as propostas que englobam as áreas de linguagens e banco de dados, TAXIS[MYLO80] se destaca, na medida em que integra a captação dos aspectos estruturais e comportamentais de uma aplicação através de mecanismos de abstração. TAXIS é virtualmente orientada às aplicações denominadas de "sistemas de informação interativos" que, segundo os autores, são caracterizados por requisitar atualizações intensivas no banco de dados e manipular um grande volume de transações pequenas e de estrutura previsível.

De uma forma geral, a linguagem oferece relações e operações associadas, transações e mecanismos para o tratamento de exceções. Como um dos seus princípios de projeto, TAXIS emprega os mecanismos de abstrações de dados, em

13. Na consulta, o caráter ! indica tupla atual.

especial, as noções de classes, propriedades e hierarquias de especialização, para incorporação da semântica estrutural e comportamental.

A figura 3.22 descreve um esquema TAXIS. Nesse esquema, pode-se notar que as propriedades que descrevem uma classe são agrupadas nas seguintes categorias:

- **Chaves** – Utilizadas na identificação de uma instância;
- **Características** – Agrupam as propriedades invariantes ao longo do tempo;
- **Atributos** – Abrigam as propriedades que são variáveis ao longo do tempo.

Como classes também são consideradas objetos, TAXIS permite a definição de meta-classes para a representação das suas propriedades gerais. Classes são sempre relacionadas através do mecanismo de especialização e, com o objetivo de facilitar a especificação semântica destas, TAXIS oferece um conjunto pré-definido de classes, a partir do qual as outras classes devem ser especializadas. No exemplo da figura 3.22, a classe pré-definida `VARIABLE-CLASS` suporta um objeto do tipo relação, que permite inclusões e exclusões de instâncias de uma classe, e a classe pré-definida `FORMATTED-CLASS` suporta a definição de classes, cujos valores, obedecem a um determinado padrão de caracteres.

```
VARIABLE-CLASS LEITOR with
Keys
  Id_Leitor: (Inscrição);
Characteristics
  Inscrição: INSCRIÇÃO;
  Nome     : NOME;
  ...
Attribute-properties
  DataSupensão: DATA;
  DiasSupensão: INTEIRO-NÃO-NEGATIVO;
  ...
end

VARIABLE-CLASS PUBLICAÇÃO with
Characteristics
  Título: string;
  ...
Attribute-properties
  DataAquisição: DATA;
  Quantidade: INTEIRO-NÃO-NEGATIVO;
  ...
end

VARIABLE-CLASS LIVRO with
Keys
  Id_Livro: (ISBN, Seqüência);
Characteristics
  ISBN: VALORES_ISBN;
  Seqüência: INTEIRO-NÃO-NEGATIVO;
  ...
Attribute-properties
  Edição: INTEIRO-NÃO-NEGATIVO;
  ...
end

FORMATTED-CLASS VALORES_ISBN with
  {Repeat(DIGIT,3)+{|'-'|}+Repeat(DIGIT,4)+{|'-'|}+Repeat(DIGIT,7)+
   {|'-'|}+Repeat(DIGIT,2)}

INTEIRO-NÃO-NEGATIVO:={|0::999|} is-a INTEGER;
```

Figura 3.22: Componente estrutural de um esquema TAXIS

De acordo com a sua filosofia de projeto, transações são também consideradas classes. Como exibe a figura 3.23, o corpo de uma transação define a sua lista de parâmetros, bem como as suas variáveis locais. Cada procedimento da transação é considerado como uma propriedade da transação e, da mesma forma que as propriedades de um classe, estas devem pertencer às categorias pré-requisito, ação ou resultado.

Já que são vistas como classes, transações podem ser especificadas a partir da especialização de outras transações. Este processo é feito através da especialização dos parâmetros, que acompanha a hierarquia de especialização que modela a parte estática da aplicação, e da redefinição dos procedimentos, que deve obedecer aos postulados de definição e especialização de transações definidos pelo TAXIS.

Um dos postulados determina que se um procedimento E foi especializado a partir de um procedimento E' então E => E'. Na figura 3.23, o pré-requisito Disponível? e a ação Retorno? foram especializados de forma a refletir a peculiaridades da subclasse PERIÓDICO. Entretanto, o pré-requisito Disponível? foi especializado incorretamente, já que a expressão booleana p.disp > 0, que o redefine, não implica a expressão p.disp > 1 do pré-requisito base. Por exemplo, para o valor de p.disp igual a 1, o pré-requisto Disponível? tem o valor falso para as transações que envolvem publicações e o valor verdadeiro para as transações que envolvem periódicos.

Exceções também são encaradas como classes e, dessa forma, podem ser especializadas. Uma exceção está sempre associada a um procedimento de pré-requisito ou de resultado, e torna-se ativa quando a avaliação deste procedimento resulta no valor falso. Cabe ao procedimento que chama a transação indicar qual a transação a ser invocada, quando uma exceção é ativada. Na figura 3.23, PUB-NÃO-ENCONTRADA é uma exceção associada ao pré-requisito EXISTENTE? e PESQ-LISTA-PEDIDOS, a transação que irá tratá-la.

Em resumo, TAXIS é um modelo no qual as abstrações de classificação e generalização são empregadas para a modelagem de toda a parte comportamental de uma aplicação, englobando as expressões, exceções e transações.

```
TRANSACTION-CLASS OBTÉM-EMPRÉSTIMO with
  parameter-list
    Obtém-Empréstimo: (p,l);
  locals
    p : PUBLICAÇÃO
    l : LEITOR;
  pre-requisites
    Sem_Suspensão? : l.DataSuspensão + l.DiasSuspensão < Today;
    Disponível?: p.Disp > 1
    Existente?:  p.Disp > 0 except PUB-NÃO-ENCONTRADA(Pub: p)
  action
    Empreste: p.Disp ← p.Disp - 1;
    Retorno: p.DtEmpréstimo ← Today;
             p.DiasEmpréstimo ← 15;
   returns
    Disp: P.Disp;
end

pre-req Disponível?: on (PERIÓDICO,LEITOR).. Obtém-Empréstimo
                is p.Disp > 0

action Retorno: on (PERIÓDICO, LEITOR)..Obtem-Emprestimo
                is p.DtEmpréstimo← Today;
                   p.DiasEmpréstimo ← 1;

TRANSACTION-CLASS REQUISITA-PUBLICAÇÃO
...
action Empréstimo: OBTÉM-EMPRÉSTIMO (p,l) exec-handler
              Valida-Pedido for PUB-NÃO-ENCONTRADA is
              PESQ-LISTA-PEDIDOS
...
```

Figura 3.23: Componente comportamental de um esquema TAXIS

3.6.8 SHM+

O SHM+ estende o modelo proposto por Smith e Smith[SMIT77a][SMIT77b] com conceitos que suportam conjuntos e modelagem comportamental. O princípio fundamental do SHM+ é o de utilizar na modelagem comportamental contrapartes dos mecanismos de abstração empregados na modelagem estrutural: agregação/seqüência, generalização/escolha e associação/repetição. Associada ao modelo foi proposta uma metodologia, denominada de ACM/PCM[14][BROD81B], para o projeto e especificação das propriedades estruturais e comportamentais das aplicações de banco de dados.

---

14. *Active and Passive Component Modelling.*

No SHM+, a especificação de um objeto, incluindo os seus atributos, relacionamentos, ações e transações, é feita através de diagramas e de um mecanismo similar a um TAD que capturam, respectivamente, as propriedades gerais e detalhadas da aplicação.

Os diagramas de especificação estrutural são denominados no SHM de "esquemas do objeto". A figura 3.24 exibe as suas notações gráficas.

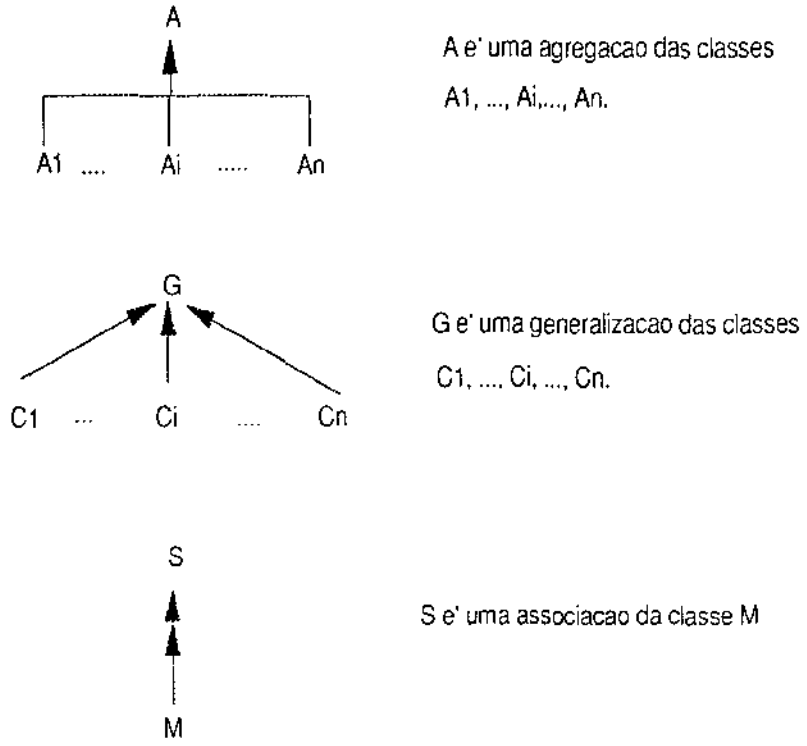


Figura 3.24: Notação gráfica SHM+ para esquemas de objetos

Para a modelagem comportamental, o SHM+ oferece operações primitivas, três formas de abstração empregadas na composição dessas operações, e duas formas de abstração de procedimentos: ação e transação. As operações primitivas disponíveis restringem-se a atuar em apenas um objeto por vez. Como foi dito, abstrações de controle, contrapartes dos mecanismos de abstração de dados, são usadas na confecção de ações, a partir da composição das operações primitivas.

Uma ação é uma operação orientada à aplicação, que assegura a manutenção de todas as restrições que envolvem um objeto. Uma ação é projetada para fornecer todo o contexto necessário à invocação de uma única operação primitiva de

modificação (inserção, exclusão ou alteração), e, por isso, da mesma forma que em TAXIS, pré-condições, pós-condições e exceções são usadas para esse feito.

Dentre as abstrações de controle, seqüência é destinada a modelar as ações que agem sobre uma agregação. Sendo `Empréstimo` uma agregação, a ação `Insere-Empréstimo`, consiste em uma seqüência de operações primitivas, como: verificar a existência e disponibilidade da publicação, verificar a situação do leitor, requisitar e incluir a data de retorno, etc. Já a abstração de controle escolha, modela as ações que envolvem uma generalização, nesse caso, a modelagem da transação `Insere-Publicação`, consiste na inclusão de um objeto da classe `Publicação`, seguida da escolha entre as inclusões na classe `Livro` ou `Periódico`.

Uma transação é uma operação orientada à aplicação que pode modificar um ou mais objetos. Também através das abstrações de controle, uma transação é formada pela composição de ações. Como nas ações, as pós-condições, pré-condições e exceções são necessárias para a garantia da manutenção da integridade semântica. Por exemplo, a transação `Prorroga-Empréstimo`, é constituída pela seqüência das ações `Verifica-Situação-Leitor` e `Modifica-Data-Retorno`.

Como mostra a figura 3.25, o ACM/PCM impõe uma rígida hierarquia de invocação.

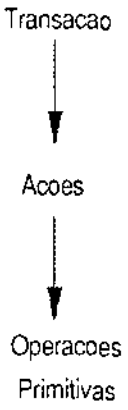


Figura 3.25: Hierarquia de invocação ACM/PCM

A representação gráfica das ações e transações SHM+ é denominada de esquema de comportamento e busca integrar, em uma única representação, as propriedades estruturais e comportamentais da aplicação. Para a representação das ações, um esquema de comportamento é obtido acrescendo-se ao esquema de objeto a ação a ser representada e as operações primitivas efetuadas em cada componente. A figura 3.26 exibe um esquema de objeto da operação `Insere-Empréstimo` e da transação `Prorroga-Empréstimo`.

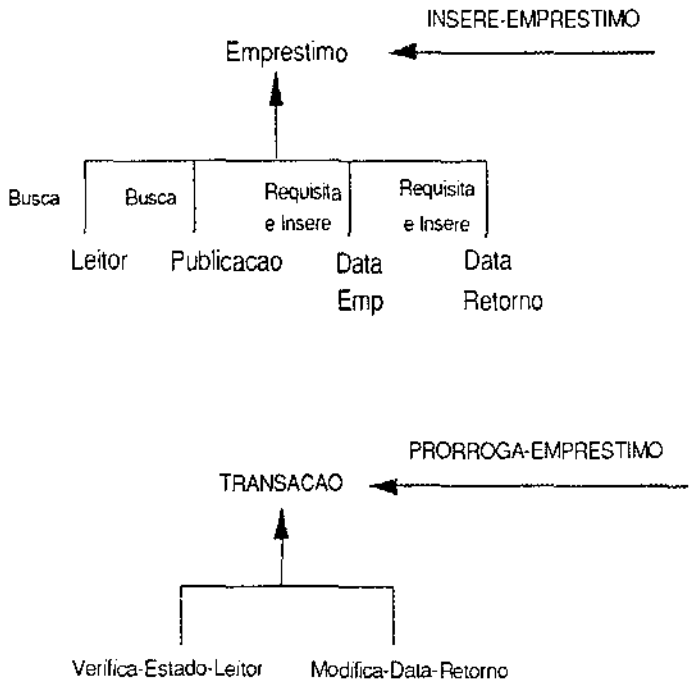


Figura 3.26: Esquemas de comportamento SHM+

O emprego dos esquemas de comportamento pelo ACM/PCM torna possível a especificação das propriedades gerais de ações e transações de uma maneira explícita e modular. Entretanto, a especificação de informações detalhadas das ações e transações é feita em termos de restrições que são expressas como predicados, tornando a mesma precisa e abstrata. A especificação de ações e transações possuem, basicamente, os mesmos componentes: o nome da operação, a lista de parâmetros, o conjunto de objetos utilizados, o objeto modificado pela operação primitiva, os conjuntos de pré e pós-condições e, no caso das ações, a operação primitiva. A figura 3.27 exibe a especificação da ação `Insere-Empréstimo`.

```
AÇÃO Insere-Empréstimo (l,p)
ENTRA (l:leitor, p:publicação, de:data-empréstimo,
       dd:data-devolução)
SAI (Empréstimo)
PRÉ-CONDIÇÃO: Publicação-Disponível (p)?
              Situação-Legal (l)?
PÓS-CONDIÇÃO: Empréstimo-Efetuado(l,p,de,dd)?
OPERAÇÃO: INSERIR Empréstimo(l,p,de,dd)
```

Figura 3.27: Especificação da ação `Insere-Empréstimo`

Em síntese, a integração dos mecanismos destinados à modelagem estrutural às abstrações correspondentes na modelagem comportamental, fazem do SHM+ e ACM/PCM propostas nas quais sobressaltam as características de modularidade, coesão e uniformidade.

3.7 Modelos Orientados a Objetos

3.7.1 Introdução

O paradigma de orientação a objetos[RENT82] vem sendo apresentado como solução de grande parte dos problemas enfrentados no desenvolvimento de software. Se em algumas áreas a utilização desse paradigma datam do início da década de 70, torna-se surpreendente notar que só recentemente a comunidade de banco de dados despertou interesse por essa abordagem. Esse interesse tardio, deve-se em parte ao sucesso alcançado pelo modelo relacional que centralizou grande parte das pesquisas realizadas.

O desenvolvimento de aplicações mais complexas que as convencionais, como CAD/CAM, CASE e OIS, vem sendo a fonte de requisitos para os vários protótipos desenvolvidos atualmente. De acordo com Zdonik[ZDON90], os SGBDs orientados a objetos são especialmente apropriados às aplicações que possuem complexidade não somente na representação dos seus dados, mas também no gerenciamento do processo de construção de programas.

Copeland[COPE84] cita importantes limitações dos SGBDs atuais, que os tornam incapazes de atender plenamente os requisitos das novas aplicações:

- Conjunto fixo e pobre de tipos de dados;
- Impossibilidade de definição de novos tipos;
- Construtores para agregação de tipos também limitados;
- Restrições artificiais oriundas das implementações;
- Habilidade restrita das estruturas existentes para suportar as complexidades e variações dos dados a serem modelados;
- Capacidade de modelagem pobre visto que estruturas mais complexas do mundo real são simplificadas no esquema conceitual ou codificadas nas estruturas disponíveis;
- As operações de atualização são orientadas para a máquina, não correspondendo às modificações existentes no mundo real que geralmente exigem uma série de atualizações nas relações existentes;
- Incapacidade de acesso eficiente e conveniente aos estados passados do banco de dados;
- Fraca integração entre a linguagem de manipulação de dados e a linguagem de propósito geral.

3.7.2 Fundamentos

A abordagem de desenvolvimento funcional tenta criar soluções a partir de decomposições sucessivas de um problema. Essas decomposições levam em conta apenas o lado funcional e/ou operacional da solução. Sistemas desenvolvidos em linguagens como C e Pascal são coleções de sub-rotinas que partilham um conjunto de dados globais, o que, quase sempre, implica em uma fraca ligação entre as ações executadas em uma sub-rotina e as entidades que as sofrem ou executam no mundo real. De uma certa forma, a abordagem funcional cria uma grande distância entre o espaço do problema e o espaço da solução|BOOC86|.

O objetivo do desenvolvimento orientado a objetos é o de criar abstrações que simulem o comportamento do mundo real, e dessa forma, encurtar a distância entre

um problema e a sua solução. Nesse contexto, um objeto corresponde a uma entidade do mundo real, composta de um conjunto de atributos, que o distingue em relação aos outros objetos, e de um conjunto de procedimentos, que define o seu comportamento em relação aos outros objetos.

Como mostra a figura 3.28, uma classe é a descrição da implementação de um conjunto de objetos do mesmo tipo. Uma classe pode ser criada a partir da especialização de outras. Esse mecanismo é comumente denominado de herança. Uma classe herda não somente os atributos bem como os procedimentos associados à superclasse, e tanto os atributos quanto os procedimentos, podem ser redefinidos na subclasse para atender as suas particularidades.

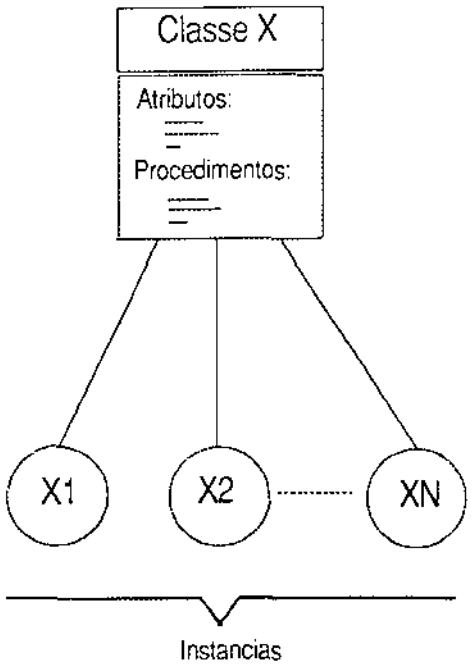


Figura 3.28: Definição de uma classe de objetos

Todo e qualquer atributo ou procedimento declarado em uma classe é parte interna ao objeto. Por isso, a definição de uma classe está intimamente ligado ao conceito de TAD[VELO86]. O encapsulamento facilita a implementação de classes de objetos, que passam a não depender dos detalhes internos de outros objetos. A reutilização de classes surge como conseqüência direta do encapsulamento e herança.

Um objeto interage como outros objetos através do envio de mensagens. Uma mensagem é a solicitação de execução de um procedimento a um objeto por outro objeto. Polimorfismo é um outro conceito peculiar que diz respeito à capacidade de classes diferentes de responder a uma mesma mensagem de acordo com o seu comportamento próprio. Identidade é a propriedade que distingue cada objeto de todos os outros. Um objeto deve ser distinguido de outro independente dos valores de seus atributos.

3.7.3 Situação Atual

O desenvolvimento de SGBDs orientados a objetos segue hoje duas orientações: ajuste da tecnologia de bancos de dados convencionais e inclusão da característica de persistência nas linguagens de programação orientadas a objetos[AGRA89]. A primeira concentra os seus esforços na introdução ou adaptação da noção de objetos às tecnologias atuais e na criação de linguagens suficientemente gerais que possam estender a funcionalidade de um banco de dados a uma linguagem de programação. Na segunda, a linguagem deve ser estendida com as características de persistência, restrições de integridade, concorrência, gatilhos e facilidade de consultas, ou seja, a linguagem deve suportar as necessidades mínimas de um SGBD.

Bancilhon[BANC87] cita alguns pontos que julga caracterizar o estado em que se encontra o campo de banco de dados orientados a objetos:

- A inexistência de um modelo comum. A falta de um consenso sobre quais as características mínimas que devem ser incorporadas a um modelo para que ele seja considerado orientado a objetos. Há até quem argumente, que os próprios conceitos do paradigma ainda são obscuros;
- A falta de fundamentação teórica;
- A grande atividade experimental.

A mudança de modelos orientados a registros para modelos orientados a objetos ocasionou uma mudança radical nos aspectos de interfaces, arquitetura física e operação dos SGBDs[DITR86]. Grande parte dos problemas enfrentados

ainda não estão satisfatoriamente resolvidos. A área de banco de dados orientados a objetos ainda encontra-se na sua infância, sendo, portanto, um campo fértil para desenvolvimento de pesquisas e alvo de grandes discussões.

3.8 Modelos Semânticos Versus Orientados a Objetos

Os modelos orientados a objetos diferem no conjunto das características incorporadas. Características como o gerenciamento de versões, são muitas vezes encontradas nesses modelos, estando, entretanto, muito mais ligadas aos requisitos das aplicações para as quais esses modelos são dirigidos do que ao próprio paradigma em si.

Os modelos semânticos geralmente incluem as noções de hierarquia de tipos e de identidade. Porém, os mecanimos de TAD e de operações associadas aos objetos são quase que exclusivos dos modelos orientados a objetos. Eles permitem que um modelo orientado a objetos possar ser estendido de uma maneira ilimitada.

Por mais interessantes que as características do paradigma de orientação a objetos possam parecer à primeira vista, elas ainda são passíveis de controvérsia, já que alguns problemas relacionados às suas implementações não estão satisfatoriamente resolvidos. Por exemplo, o caráter estrutural de uma linguagem de manipulação de dados conflita com o encapsulamento de dados proposto.

As abordagens diferem radicalmente na maneira de incorporar as semânticas das aplicações. Como visto, os modelos semânticos enfocam o problema de uma forma estrutural. Já os modelos orientados a objetos possuem uma abordagem comportamental: grande parte da semântica da aplicação deve ser implementada através dos procedimentos que determinam o comportamento de um objeto.

Capítulo 4

S-SQL

4.1 Motivação

Alguns modelos semânticos proeminentes, como E-R, FDM e SDM, possuem atualmente uma importância histórica. A indústria não foi muito receptiva a esses modelos, já que investimentos maciços tinham sido feitos na direção do desenvolvimento de SGBDs relacionais. Uma honrosa exceção trata-se do SIM, que implementa um subconjunto do modelo SDM.

Também a desencorajar ainda mais as implementações desses modelos, contribuiu a dissociação das propostas destes das suas respectivas LMDs. Outro fato a contribuir é o caráter extremamente conservador do mercado de informática, pois, por mais inacreditável que possa parecer, ainda existem locais que ainda possuem todo o seu desenvolvimento baseado na draconiana linguagem COBOL e em um SGBD DBTG ou hierárquico. Os próprios SGBDs relacionais foram vítimas deste conservadorismo, já que só detiveram a maioria das instalações de computação quinze anos após a proposta inicial de Codd.

Como visto no capítulo anterior, outra vertente de desenvolvimento dos modelos semânticos seguiu a direção da expansão das capacidades dos SGBDs relacionais. O próprio Codd foi um dos primeiros integrantes desta abordagem, apesar de sua proposta se restringir ao nível conceitual.

POSTGRES talvez seja o melhor representante dessas extensões. Contudo, por possuir um ambiente tão desconexo, a ponto de procedimentos e TAD serem implementados fora do SGBD, através de uma linguagem convencional, fica difícil

enxergá-lo como uma alternativa, no mínimo, adequada para a implementação das aplicações não convencionais, que é o seu principal foco de atenção.

Apesar de não ser tão poderoso quanto o POSTGRES, o GEM foi uma extensão simples e concisa. Infelizmente, como será visto no capítulo 5, as opções de implementação foram visivelmente influenciadas por alguns aspectos particulares da máquina de banco de dados na qual foi implementado[TSUR84].

A construção de interfaces semânticas para SGBDs relacionais é uma outra abordagem desenvolvida atualmente. Elas se caracterizam por serem alternativas de baixo custo e, até certo ponto, efetivas, no sentido de uma melhor captação semântica e aprimoramento da interação entre usuários e SGBD. Essas interfaces são em geral gráficas, comercializadas como pacotes associados a SGBDs relacionais, tendo em seu público alvo usuários inexperientes que, através deste ambiente, podem interagir com o SGBD de uma maneira bem mais amena[ROGE87].

Sem dúvida alguma, para grande parte dos usuários, por ser um investimento extremamente alto, a migração de um SGBD relacional para um semântico, sob um critério custo/benefício, não é uma opção muito atrativa. E, sob esse ponto de vista, os modelos semânticos não estão sendo encarados como opção de desenvolvimento e investimento pela indústria.

Vale observar, que a comunidade acadêmica encarou as aplicações comerciais com um certo preconceito. Afirmativas de que as suas modelagens são extremamente simples e que o modelo relacional é suficientemente capaz de atender esse tipo de aplicação são comuns[STON88]. Entretanto, qualquer pessoa que tenha uma mínima experiência nessas aplicações verifica essa inverdade. Os SGBDs relacionais são utilizados simplesmente como meio de implementação, não satisfazendo, uma série de outros requisitos[KENT79].

A linguagem SQL, por exemplo, não teve o alcance esperado com os usuários finais, pois, em geral, eles a acham complicada. Um outro problema é a extrema liberdade de modelagem proporcionada pelo modelo relacional, o que propicia o desenvolvimento dos mais absurdos projetos lógicos, pois boa parte dos técnicos desprezam ou desconhecem as formas normais. Tudo isso, sem falar no

fraco, ou mesmo nenhum, suporte à manutenção de restrições de integridade oferecido por esses SGBDs.

4.2 Interface S-SQL

A dissertação que se segue, descreve uma interface que estende as capacidades de captação semântica de um SGBD relacional. A interface implementa um modelo que torna disponível um número de características mínimas, porém essenciais, como classificação, agregação, generalização, derivação de classes, *surrogates* e atributos multivalorados. O objetivo principal desta dissertação é o de transformar um SGBD relacional em uma base mais adequada ao desenvolvimento de aplicações e, com isso, verificar a viabilidade e adequação de um SGBD relacional como base de implementação de um modelo semântico.

Um dos pontos que diferencia a proposta desta dissertação das abordagens mencionadas anteriormente é o fato de que a implementação desse modelo está amarrada simplesmente a um SGBD relacional que ofereça a linguagem SQL. Com essa visão prática, tenta-se, de uma certa forma, evitar a vala comum de mais uma proposta de um modelo semântico, ou mesmo de extensão de um SGBD relacional, pois, para instalações SQL, o seu emprego pode ser imediato. Serão também propostas, as LDD e LMD associadas ao modelo. Esses e outros aspectos estão detalhados nas seções e capítulos posteriores.

Doravante, para facilitar o entendimento do texto, a interface a ser descrita será denominada de *Semantic*-SQL (S-SQL). Não se trata, entretanto, de uma proposta de extensão para a linguagem SQL. Este nome advém, tão simplesmente, da influência sintática exercida pela linguagem SQL, na LDD e LMD da interface proposta. A semelhança sintática com a linguagem SQL não é ocasional: a interface é basicamente dirigida a usuários que lidam com um SGBD SQL.

4.2.1 LDD

A figura 4.1 exibe os comandos S-SQL necessários à construção de parte do esquema da aplicação descrita no apêndice A. A sintaxe da LDD pode ser

encontrada no apêndice B desta dissertação, e a sua respectiva representação gráfica no apêndice D.

```
Create Class Pessoa(Nome char (40), RG char (12),
                    Endereço  Endereço) Key (RG)

Create Class Aluno(RA char(6), Curso Curso, Média float,
                   Esportes  (char(20))) Key (RA)

Create Class Funcionário(Matrícula char(6), Cargo Cargo,
                         Salário integer) Key (Matrícula)

Covering Subclasses of Pessoa are Aluno, Funcionário

Create Class Monitor(Bolsa  integer, Disciplina  Disciplina)

Create Class Bolsista(Bolsa  integer)

Partial Subclass of Funcionário, Aluno is Monitor

Derived Subclass of Aluno is Bolsista Where (Média > 9.5)

Create Class Tec-Adm(Vantagens integer, Órgão  Órgão)

Create Class Professor(Titulação  char(4), Depto  Departamento)

Overlapping Subclasses of Funcionário are Tec-Adm, Professor

Create Class Pesquisa(Coordenador  Professor, Tema char(20))

Create Class Pesquisador(Adicional  integer)

Derived Subclass of Professor is Pesquisador  Where is a
        value of Coordenador From Pesquisa

Create Class Órgão(Nome  char(40), Endereço  Endereço)

Create Class Departamento(Nome  char(40), Instituto  Órgão)

Create Class Curso(Nome  char(40), Depto  Departamento)

Create Class Disciplina(Nome char(40), Depto Departamento)

Create Class Oferta(Disciplina  Disciplina, Turma  char(2),
                    Vagas integer) Key (Disciplina, Turma)

Create Class Matrícula(Aluno     Aluno,   Oferta  Oferta,
                       Notas (integer)) Key (Aluno,Oferta)

Create Class Endereço (Rua      char(40), Bairro  char(20),
                       Cidade  char(20), CEP     char(5))
```

Figura 4.1: Esquema em S-SQL

Como pode ser notado, cada classe contém somente os seus atributos distintos. O comando `Subclasses of` define a especialização de classes em

subclasses. As quatro variações deste comando, `Overlapping`, `Disjoint`, `Covering` e `Partitioning`, denotam a forma de especialização requisitada.

Em uma herança múltipla várias classes podem ser especializadas a partir de um grupo de classes bases. Por exemplo, o comando `Overlapping SubClasses of Aluno, Funcionário are Monitor, Estagiário` não só exprime a herança múltipla, mas as propriedades de que nem todo Aluno/Funcionário é um Estagiário ou Monitor, e que um Estagiário pode também ser um Monitor. Contudo, as classes bases devem pertencer a uma mesma categoria[15] que, para permitir a presença de elementos comuns nas subclasses, deve ser obrigatoriamente definida pelos tipos de especialização `Overlapping` ou `Covering`.

A fim de evitar uma LDD de sintaxe deselegante, a interface S-SQL oferece o comando `SubClass of`, que deve ser empregado quando uma especialização envolve uma única subclasse. As variações `Total` e `Partial` indicam, respectivamente, a inclusão total ou parcial das instâncias da superclasse na subclasse. Já a variação `Derived`, denota a derivação de uma subclasse a partir de um predicado de pertinência. Esse predicado pode ser baseado nos atributos simples da classe base ou em um atributo que possui a classe base como contra-domínio.

Atributos multivalorados são indicados por um tipo atômico simples entre chaves. Esses atributos podem conter zero ou mais elementos. O atributo `Esportes` da classe `Aluno` é um exemplo desse tipo de atributo.

A declaração `Key` denota os atributos que fazem parte de uma chave, e que, conseqüentemente, não devem assumir valores nulos. O modelo também permite que atributos tenham como contra-domínio os *surrogates* de uma entidade.

Uma agregação é representada através do emprego destes conceitos. Os atributos de uma agregação, que fazem referência às entidades que a compõe, devem ser declarados como chave da agregação. Sendo assim, quando instâncias de uma entidade componente são retiradas do banco de dados, por não ser

15. Entende-se por categoria, um conjunto de classes independentes que partilham as mesmas superclasses.

permitido que um atributo chave assuma o valor nulo, as instâncias da agregação que fazem referência às instâncias componentes são também retiradas.

Na definição de um esquema, deve-se ter em mente as seguintes regras para a resolução de conflito de identificadores:

- **Regra 1** – Em uma herança simples, não é permitido a qualquer subclasse definir um atributo que tenha o mesmo identificador de um atributo da superclasse.
- **Regra 2** – Em uma herança múltipla, em caso de conflito de identificadores, os atributos das superclasses primeiramente indicadas na definição da categoria, têm precedência sobre os atributos das outras superclasses.

4.2.2 Representação Gráfica

Visualisar e analisar os relacionamentos existentes entre as entidades de uma aplicação através dos comandos da LDD de um modelo, é uma tarefa extremamente árdua. Por isso, uma representação gráfica adequada às características oferecidas por um modelo é fundamental para sua adoção. Este foi um forte apelo do modelo E-R, e a representação gráfica da interface S-SQL tenta seguir essa direção.

A figura 4.2 exibe a representação gráfica de parte dos relacionamentos contidos no esquema apresentado na seção anterior. Cada classe é representada por um retângulo, e uma especialização desta, é diagramada através de uma seta em linha dupla. O predicado de pertinência associado a uma subclasse derivada, deve ser indicado entre colchetes.

Da mesma forma, uma agregação também é vista como uma classe, já que o modelo não oferece um construtor dedicado à sua representação. Um domínio simples é representado por "retângulos de cantos arredondados". Bem ao estilo do modelo funcional, setas rotuladas por identificadores indicam a ligação dos atributos de uma classe aos seus domínios. Os atributos participantes da chave de uma classe são indicados por uma seta preenchida. Já os atributos que não fazem parte da chave, por um uma seta, e os multivalorados, por uma seta dupla.

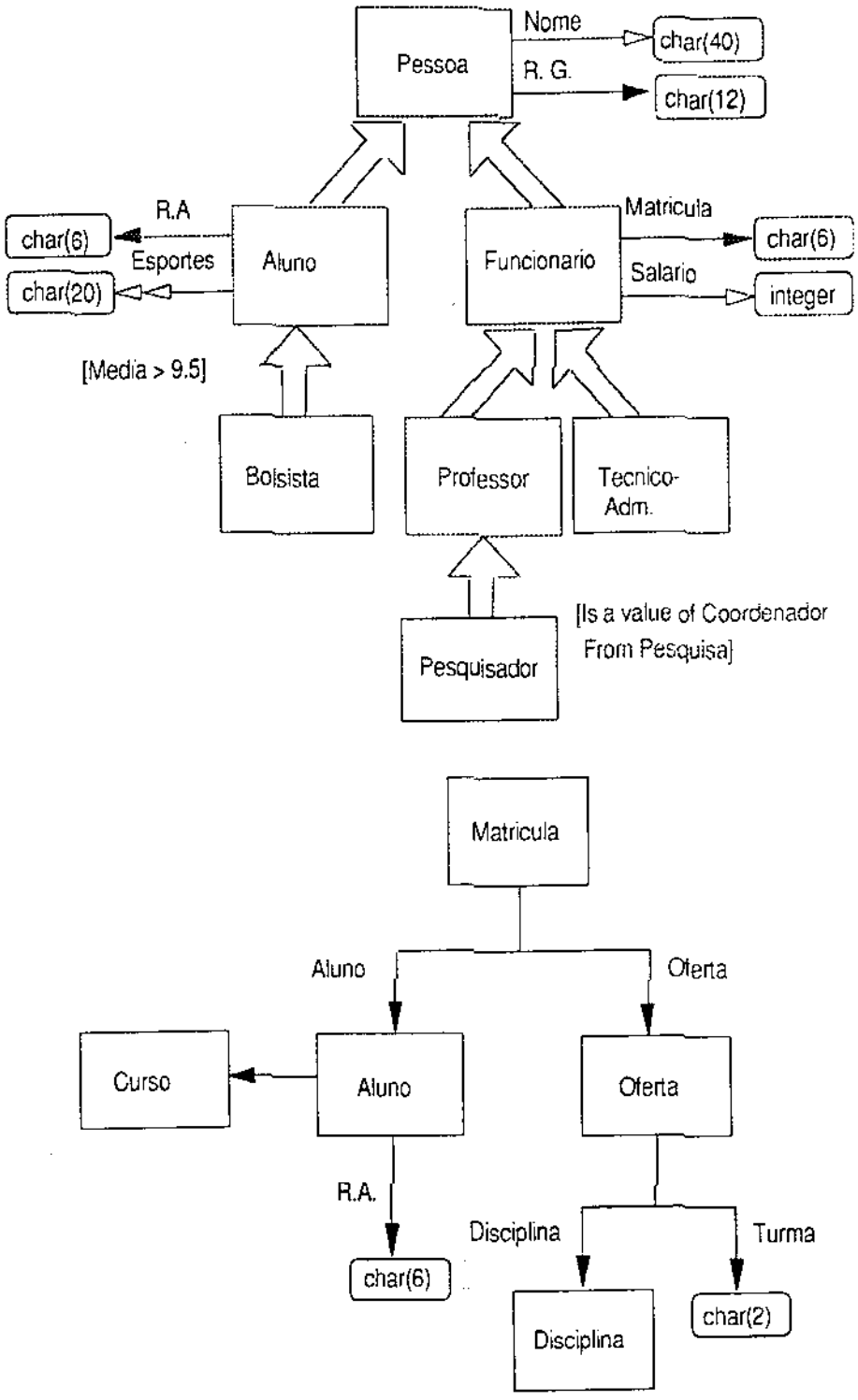


Figura 4.2: Representação gráfica S-SQL

4.2.3 LMD

Além de tornar disponível os mecanismos de abstração oferecidos pelo modelo, a LMD visa oferecer, quando possível, uma sintaxe mais fácil e intuitiva que a da SQL padrão. O processo de tradução da LMD S-SQL para SQL será discutido no capítulo 5. O apêndice C contém a descrição sintática das extensões da LMD S-SQL.

Um dos requisitos necessários à LMD de um modelo semântico para suportar a abstração de agregação, é o acesso aos atributos de instâncias componentes a partir da instância composta. Nesse sentido, a LMD S-SQL seguiu a notação de ponto utilizada pelo GEM.

A figura 4.3 exemplifica uma recuperação, onde a qualificação das instâncias é baseada nos valores dos atributos das instâncias componentes de uma agregação.

```
Select Nome
From Curso
Where Depto.Instituto.Nome = "Instituto de Matemática"
```

Figura 4.3: Cursos ligados ao Instituto de Matemática

Alguns críticos dos modelos semânticos afirmam que, por expressarem os relacionamentos existentes entre as entidades, estes modelos tendem a ser suportados por LMDs que só se utilizam dos caminhos expressos por esses relacionamentos. Como mostra a figura 4.4, na LMD S-SQL, nada impede que o usuário efetue uma consulta através de junções relacionais, constituindo, dessa forma, um caminho não expresso anteriormente.

```
Select F.Matrícula, F.Nome
From Funcionário F,
     Funcionário R
Where F.Salário > R.Salário and
      R.Cargo.Nome = "Reitor"
```

Figura 4.4: Funcionários que percebem mais que o Reitor

Poderão existir ocasiões em que a consulta deverá ser expressa com base no valor de um atributo do tipo classe (*surrogate*). Nesse caso, a LMD S-SQL deve oferecer operadores que efetuem as comparações necessárias.

Como o valor deste tipo de atributo trata-se na verdade de um inteiro, a LMD S-SQL usa os próprios operadores = e !=. Contudo, sob o ponto de vista semântico, só devem ser permitidas as comparações entre *surrogates* de uma mesma classe, ou de classes especializadas a partir de uma mesma classe base. Nenhum dos outros operadores relacionais (<,>, ...), fazem qualquer sentido semântico, e por isso, não devem ser aplicados a esses atributos. A figura 4.5 apresenta uma consulta que utiliza o operador =.

```
Select Func2.Matrícula
From Funcionário Func1,
     Funcionário Func2
Where Func1.Cargo = Func2.Cargo and
      Func1.Nome = "Luiz Cláudio"
```

Figura 4.5: Funcionários que possuem o mesmo cargo de Luiz Cláudio

Desde que fosse permitido o acesso ao *surrogate* de uma instância, a consulta da figura 4.3 poderia ser implementada por uma junção. Apesar de não se justificar a necessidade de tal acesso, a LMD S-SQL torna disponível este valor através do atributo de identificador padrão <Nome-da-tabela>#. Este atributo é criado automaticamente pela interface na definição de cada relação. A figura 4.6 ilustra a recuperação das instâncias componentes através do emprego de junções e *surrogates*.

```
Select Curso.Nome
From Curso,
     Departamento,
     Órgão
Where Depto = Departamento# and Instituto = Órgão# and
      Órgão.Nome = "Instituto de Matemática"
```

Figura 4.6: Cursos ligados ao Instituto de Matemática

Para a inclusão de objetos em uma agregação, é necessário que se tenha o valor dos *surrogates* das instâncias componentes. A solução adotada pela LMD S-SQL foi a de aplicar predicados que identificam essas instâncias, nas suas

respectivas classes. Todavia, se um dos predicados fornecidos, for válido para mais de uma instância, ou se não for válido para todas as instâncias da classe componente, a inclusão será rejeitada. A figura 4.7 exemplifica essa abordagem.

```
Insert into Departamento (Nome, Instituto)
Values ("Ciência da Computação",
        Nome = "Instituto de Matemática")
```

Figura 4.7: Inclusão de um departamento

A retirada de instâncias de uma agregação é extremamente simples. Como mostra a figura 4.8, a qualificação pode ser baseada nos valores dos atributos de instâncias componentes. Este comentário também é válido para o comando `Update`. Vale lembrar que a interface é responsável pela manutenção da integridade referencial no banco de dados.

```
Delete Curso
Where  Depto.Instituto.Nome = "Instituto de Matemática"
```

Figura 4.8: Exclusão dos cursos associados ao Instituto de Matemática

Como ilustra a figura 4.9, o acesso aos atributos de uma superclasse, a partir de uma subclasse, é transparente para o usuário.

```
Select Nome, RA
From Aluno
Where Curso.Nome = "Ciência da Computação"
```

Figura 4.9: Nome e RA dos alunos de Ciência da Computação

Algumas vezes, na seleção de registros, torna-se necessário determinar a qual subclasse um determinado *surrogate* está associado. Para esse fim, a LMD S-SQL oferece o operador `IS-A`. A figura 4.10 exemplifica o seu emprego.

```
Select RG, Nome
From Pessoa
Where Pessoa# IS-A Aluno and
      Pessoa# IS-A Funcionário
```

Figura 4.10: Alunos que são funcionários

É claro que este tipo de consulta só possui significado semântico quando a forma de especialização permite que as subclasses envolvidas possuam elementos em comum.

O operador `IS-A` também pode ser útil em consultas que envolvem uma agregação. A figura 4.11 mostra uma consulta que retorna todas a disciplinas nas quais os monitores se matricularam. Vale notar, que a agregação `Matrícula` composta de `Aluno` e `Oferta`, e que `Monitor` é uma subclasse da classe `Aluno`.

```
Select Aluno.Nome, Oferta.Disciplina.Nome
From Matrícula
Where Aluno IS-A Monitor
```

Figura 4.11: Disciplinas que os monitores cursam

A inclusão de uma instância em uma rede de generalizações implica em uma série de restrições que devem ser observadas pela interface. Por exemplo, a inclusão de uma instância em uma classe que deriva outras classes, através de uma especialização particionada ou coberta, não é permitida. O comando da figura 4.12 seria recusado pela interface S-SQL, já que qualquer pessoa deve ser obrigatoriamente um aluno ou um funcionário.

```
Insert into Pessoa (Nome, RG, Endereço)
Values ("Joaquim","2739933",
         Rua = "Cel Durval de Matos nº 55")
```

Figura 4.12: Inclusão incorreta

Há ocasiões em que um objeto que já se encontra representado por uma instância em uma determinada classe tem que ser instanciado em uma outra classe na rede de generalizações. Por exemplo, um funcionário técnico-administrativo que é admitido como professor. Para que um objeto tenha a mesma identificação nas diversas instâncias que o representam na rede, é necessário que, ao se incluir uma nova instância, tenha-se disponível o *surrogate* das instâncias já inclusas.

Para contornar essa situação, a LMD S-SQL oferece a cláusula `Surrogate from <Classe> Where <Predicado>`. A Classe especificada na cláusula

`Surrogate from` deve ser uma superclasse da classe na qual está sendo feita a inclusão. A figura 4.13 exemplifica o uso desta extensão.

```
Insert into Professor (Titulação, Depto)
Values ("MSc", Nome = "Ciência da Computação")
Surrogate from Funcionário
Where  (Matrícula = "896755")
```

Figura 4.13: Técnico-administrativo contratado como professor

No exemplo apresentado, caso a especialização de `Funcionários` nas classes `Tec-Adm` e `Professor`, não permitisse instâncias comuns entre as classes, a interface "entenderia-o" como uma transferência de instanciação, e, dessa forma, retiraria a instância correspondente da classe `Tec-Adm` e respectivas subclasses.

Por motivos óbvios, a inclusão ou exclusão de instâncias nas subclasses derivadas a partir de predicados, não é permitida. Essas operações devem ser disparadas por transações que, através de inclusão ou exclusão de instâncias na classe base ou da modificação dos valores dos atributos que participam de um predicado, tornam o predicado de pertinência válido ou inválido.

Adotando esse comportamento, a interface é responsável pela inclusão e exclusão de instâncias em uma classe derivada. Entretanto, é de inteira responsabilidade do usuário, atualizar os atributos das instâncias incluídas. Devido à inclusão automática de instâncias, não é permitida a declaração de atributos obrigatórios nas classes derivadas.

Tomando como exemplo a primeira coordenação de pesquisa para um professor, o usuário deveria proceder a transação de acordo com a figura 4.14.

```
Insert into Pesquisa(Coordenador, Tema)
Values (Matrícula = "892821","Sistemas Operacionais")

Update Pesquisador Set(Adicional = 50000)
Where Matrícula = "892821"
```

Figura 4.14: Inclusão da primeira pesquisa de um professor

A inclusão de uma instância em uma classe especializada por uma herança múltipla, deve também ser feita através do comando `Insert into-Values-Surrogate from-Where`, já que a interface exige a existência da instância a ser incluída nas classes bases. A figura 4.15 exibe os comandos que incluem um aluno como monitor da universidade.

```
Insert into Funcionário(Matrícula, Função, Salário, Cargo)
Values ("928984",Nome = "Monitor",0,NULL)
Surrogate from Pessoa
Where  (RG = "878734")

Insert into Monitor(Bolsa, Disciplina)
Values (80000, Nome = "Lógica de Programação")
Surrogate from Pessoa
Where  (RG = "878734")
```

Figura 4.15: Aluno contratado como monitor

Uma subclasse derivada por herança múltipla total é considerada como um caso especial de derivação por predicado. Nesse caso, o predicado de pertinência é a existência da instância em todas as classes bases. Dessa forma, se a especialização da subclasse `Monitor` fosse total, o sistema automaticamente incluiria um novo registro nesta classe, após a execução da primeira transação de inclusão da figura 4.15.

A retirada de objetos de uma rede de generalizações também implica na manutenção de propriedades semânticas pela interface. Como exemplo, a retirada de instâncias de uma classe derivada a partir de uma especialização sobreposta ou disjunta, não deve provocar a propagação para a superclasse. A figura 4.16 ilustra um comando de exclusão na subclasse `Aluno`.

```
Delete Aluno
Where (Aluno.Curso.Nome = "Ciência da Computação")
```

Figura 4.16: Retirada dos alunos de Ciência da Computação

A retirada de um objeto de uma classe, em uma rede de generalizações, é uma tarefa extremamente complexa, pois, pode provocar uma seqüência de atualizações no banco de dados, que envolvem, além de retiradas na própria classe,

retiradas nas subclasses, em alguns casos superclasses, e agregações, nas quais a classe, subclasses e superclasses participam como componentes.

Atributos multivalorados, além de proporcionarem, em muitos casos, uma modelagem mais natural, visam facilitar algumas consultas que, de outra forma, seriam extremamente difícies de expressar. Um dos operadores comumente utilizado sobre esses atributos é o operador IN, que indica a pertinência de um determinado elemento ao conjunto de elementos de um atributo multivalorado. A consulta 4.17 exemplifica a sua utilização.

```
Select RA, Nome, Curso.Nome
From Aluno
Where "futebol" IN Esportes
```

Figura 4.17: Alunos que praticam futebol

Outros operadores de conjuntos, como igualdade (=), contém (=>) e está contido (<=), também são providos. Esses operadores aceitam como operandos atributos multivalorados de um mesmo tipo base. A figura 4.18 exemplifica o emprego do operador <=.

```
Select Nome, Curso.Nome
From Aluno
Where {"futebol", "tênis"} <= Esportes
```

Figura 4.18: Alunos que praticam futebol e tênis

Algumas funções de grupo, como Min e Max, também podem ser utilizadas com atributos multivalorados que tenham o tipo inteiro como base. Para os atributos que têm como base o tipo caracter, a única função válida é Count. Tipicamente, em duas situações essas funções se aplicam. Na primeira, a função é aplicada em uma condição de recuperação, na segunda, em um atributo contido em uma cláusula Select. A figura ilustra uma recuperação que emprega a função Avg.

```
Select Aluno.Nome, AVG(Notas)
From Matrícula
Where Oferta.Disciplina.Nome = "Contabilidade I" and
      AVG(Notas) > 6.0
```

Figura 4.19: Alunos aprovados na disciplina Contabilidade I

A interface S-SQL também permite que os operadores de conjunto sejam aplicados a conjuntos criados dinamicamente pelo usuário. A especificação desses conjuntos deve indicar o elemento base, a condição de pertinência e, em alguns casos, o atributo de agrupamento. A figura 4.20 ilustra essa aplicação.

```
Select Aluno.RA, Aluno.Nome
From Matricula, Disciplina
Where {Oferta.Disciplina group by Aluno} >=
      {Disciplina* Where Depto.Nome = "Hidráulica"}
```

Figura 4.20: Alunos matriculados em todas as disciplinas de Hidraúlica

À primeira vista, a especificação de um conjunto, pode parecer estranha aos usuários já acostumados com a sintaxe `Select-From-Where` da SQL. Entretanto, numa análise mais apurada, ela se mostra bastante intuitiva: as variáveis tuplas são especificadas em uma cláusula `From` externa à especificação do conjunto e, por não ser apropriada, a cláusula `Select` é simplesmente abolida.

Como visto, a especificação dos elementos de um atributo multivalorado é feita por um conjunto de valores do tipo base, separados por vírgulas e entre chaves. A figura 4.21 exibe a inclusão de uma aluno que pratica futebol e tênis.

```
Insert into Aluno(Nome, RG, Endereço, RA,
                  Curso, Média, Esportes)
Values ("Vinícius Oliveira","232321", Rua = "Cel. Durval 1042",
        "887521", Nome = "Economia", NULL, {"futebol",
"tênis"})
```

Figura 4.21: Inclusão de um aluno

A inclusão ou exclusão de elementos de um atributo multivalorado deve ser feita através do comando `Update`. Os sinais `+` e `-`, precedendo um conjunto de valores, são utilizados para indicar a soma e subtração de elementos,

respectivamente. A figura 4.22 mostra a inclusão de mais dois esportes para um determinado grupo de alunos.

```
Update Aluno
Set Esportes = +{"Volei","Basquete"}
Where Curso.Nome = "Ciência da Computação"
```

Figura 4.22: Inclusão de esportes para os alunos de Computação

4.3 Modificações de Esquemas

A capacidade de uma interface em acompanhar facilmente todo o processo evolutivo de uma aplicação é, sem dúvida alguma, um grande atrativo para sua utilização.

Um SGBD relacional, pela simplicidade do modelo, torna todo o processo de modificações de esquemas igualmente simples: geralmente, são permitidas as exclusões de tabelas e adições de campos em uma tabela. O usuário é o único responsável por possíveis inconsistências semânticas que alguma modificação possa provocar.

Já uma interface que implementa um modelo semântico é, consequentemente, o "reverso da moeda": ela deve ser a única responsável pela manutenção da coerência do esquema na efetivação das modificações.

A abordagem adotada pela interface S-SQL foi a de simplificar o modelo em alguns aspectos e de restringir as modificações permissíveis, a fim de tornar o processo de modificação de esquema menos traumático.

Apesar das várias possibilidades de modificação de um esquema apontadas pela taxonomia desenvolvida por Barnejee[BARN87], decidiu-se por simplificá-las a sete operações básicas, categorizadas em três grupos:

1. Modificações do conteúdo de uma classe

    - Inclusão de um atributo

- Exclusão de um atributo
- Inclusão de uma chave
- Exclusão de uma chave

2. Modificações nas classes
   - Inclusão de uma classe
   - Exclusão de uma classe
3. Modificações nos relacionamentos
   - Conexão de uma classe como subclasse em uma categoria existente

A inclusão de um atributo em uma classe deve respeitar as regras contidas na seção 4.2.1, embora ambos, inclusão e exclusão, possam modificar a precedência descrita pela regra 2. A exclusão de um atributo provoca, também, a exclusão das chaves da qual este atributo participa. Caso o atributo excluído participe do predicado de derivação de uma classe, esta também deverá ser excluída.

A inclusão de uma classe em um esquema é considerada um fato isolado e em nada altera os relacionamentos existentes. Já a exclusão de uma classe, desde que ela se encontre em uma rede de generalizações, segue uma abordagem semelhante à adotada por Bamejee, ou seja, a transferência de todas as referências da classe para a superclasse e a definição da sua superclasse como superclasse direta das suas subclasses.

A exclusão de uma classe especializada por uma herança múltipla pode significar um impasse. Como existem mais de uma superclasse habilitadas a herdar as suas referências, o modelo assume como classe mais "prioritária" a primeira classe indicada na especialização. No caso de exclusão de uma classe derivada por predicados, as referências deverão ser herdadas pela superclasse imediata a classe base.

A única restrição explicitamente imposta pela interface é a proibição da exclusão das classes que são raízes em uma rede de generalizações, visto que elas não possuem superclasses para herdar as suas referências.

A exclusão de classes que não participam de uma rede de generalizações, provoca a exclusão dos atributos que as fazem referência, pois, é evidente que não há o menor sentido semântico em se fazer referência a uma classe inexistente.

A inclusão de uma classe em uma rede de generalizações também é extremamente simplificada. Ela só pode ser incluída no papel de uma subclasse. Vale ressaltar, que os atributos definidos na subclasse devem respeitar as regras de conflito de identificadores contidas na seção 4.2.1.

Também estão disponíveis comandos para incluir e excluir chaves de uma classe, propiciando, dessa forma, modificações nos atributos que definem uma agregação.

As sintaxes dos comandos de modificações de esquema estão definidas no apêndice B. A figura 4.23 ilustra alguns comandos que modificam o esquema da figura 4.1.

```
Alter Class Funcionário Add (DataContrato char(6))
Alter Class Aluno Drop (Esportes)
Alter Class Disciplina Add Key (Nome)
Alter Class Pessoa Drop Key (RG)
Drop Class Monitor
Create Class Estagiário (DataTérmino char(6))
Include Estagiário as Funcionário subclass
```

Figura 4.23: Modificações do esquema

4.4 Justificativas

O leitor pode ter sentido a ausência de algumas características não incorporadas pela interface S-SQL que podem ser bastante úteis para uma melhor captação semântica. Esta seção visa justificá-la, levando em conta a real

necessidade, compatibilidade com a linguagem SQL, complexidade adicional e exeqüibilidade da extensão.

A implementação de valores nulos, por exemplo, não foi considerada uma extensão, pois a grande maioria dos SGBDs atuais já trazem disponível essa característica.

Campos longos poderiam também ser facilmente implementados. Eles são bastante utilizados para arquivar *bitmaps* e outras informações não estruturadas. Contudo, a recuperação das informações de um campo longo, só possui um real sentido a partir de interfaces gráficas, pois o sistema em si não é capaz de oferecer nenhum operador destinado a este tipo de atributo.

O mecanismo de derivação de classes com base em predicados, como foi proposto, pode, em muitos casos, não ser satisfatório, já que somente é permitido a utilização dos atributos simples na composição do predicado. Apesar de mais flexível, a participação de atributos de outras classes produziria, senão um processo de tradução de atualizações mais complexo, uma ineficiência brutal nas recuperações de informações.

Por sua vez, o mecanismo de associação permitiria uma maior poder de expressão ao modelo, já que possibilitaria um distinção explícita entre as propriedades de um conjunto e de seus membros. Entretanto, por ser satisfatoriamente substituível por uma agregação, ela não torna o modelo menos poderoso. Além disso, a sua inclusão iria requerer uma maior extensão da LC/LMD, tornando-as mais complexas e distantes da SQL padrão.

A verificação de cardinalidade nas agregações não foi implementada, já que é permitido a um atributo fazer referência a uma entidade, sem haver portanto, a necessidade de se construir uma agregação para tal. No esquema da figura 4.1, a tabela `Aluno` fazendo referência a `Curso` é um exemplo dessa característica. Com isso, as agregações 1:N e 1:1 devem ser diretamente mapeadas para as entidades, o que permite, inclusive, uma maior eficiência na recuperação de informações. Dessa forma, as agregações, nesse modelo, são nitidamente restritas à cardinalidade M:N.

Facilmente a LMD da interface poderia ser estendida a fim de suportar o aspecto temporal; isto, inclusive, já foi feito para o QUEL[SNOD85]. Contudo, a implementação do suporte temporal no topo de um SGBD relacional é extremamente ineficiente, para não dizer impraticável[LUM84].

TAD é sem dúvida alguma uma extensão extremamente útil e de grande valia[STON86]. Entretanto, a forma pela qual está sendo proposta a interface, ou seja, como a "casca" de um SGBD relacional, também impede que esta característica seja implementada de uma forma adequada e eficiente, pois modificações internas, no código do SGBD, tornam-se necessárias. Ademais, para que TAD seja realmente uma característica útil, deveria ser desenvolvida uma linguagem associada ao modelo, na qual o usuário possa especificá-lo.

Apesar de bastante simples, a linguagem SQL já provê um mecanismo de derivação de atributos. Um mecanismo mais complexo, que pudesse derivar valores a partir dos dados de diversas tabelas, mereceria ser investigado. Porém, desde já, tudo leva a crer que a eficiência não seria uma de suas virtudes, pois, embasado unicamente em recuperações, cada acesso a um atributo derivado deve provocar o seu completo recálculo.

A implementação de restrições simples, como valores permitidos de um atributo, não proporcionaria um maior poderio ao modelo. A fraca eficiência obtida, também seria o impecílio para a implementação de uma extensão destinada a manter restrições arbitrárias. De qualquer forma, esta extensão também deveria ser alvo de maiores investigações.

Capítulo 5

Implementação

5.1 Abordagens Anteriores

Mapear as abstrações oferecidas por um modelo semântico em um SGBD relacional, envolve uma série de alternativas de implementação, que devem ser analisadas sob os parâmetros de desempenho, espaço de armazenamento, manutenção de integridades, etc.

Uma rede de generalizações, por exemplo, proporciona algumas observações interessantes. Um problema a ser enfrentado é a manutenção da dependência de inclusão entre as classes, que determina que qualquer instância de uma subclasse seja também uma instância da superclasse.

A figura 5.1 ilustra a abordagem adotada pelo GEM[TSUR84], que implementa as classes participantes de uma hierarquia[16] de generalização em uma única relação. Atributos especiais, definidos internamente pela própria ferramenta, identificam as subclasses às quais cada registro está associado.

16. O termo hierarquia é usado propositalmente, já que o GEM não permite herança múltipla.

Relacao PESSOA

R.G.	Nome	D1	R. A.	Curso	Matricula	Cart. Trabalho
2736273	Ricardo	1	895494	Comput.	———	———
2363197	Nadia	1	902345	Eletrica	———	———
3456436	Julio	2	———	———	459504	4.506.390.933
2456777	Carolina	2	———	———	495033	2.199.992.142
⋮	⋮	⋮	⋮	⋮	⋮	⋮

D1 = 1 => Sub-classe ALUNO

D1 = 2 => Sub-classe FUNCIONARIO

Figura 5.1: Implementação de generalização no GEM

Sem dúvida, o mapeamento adotado foi um tanto excêntrico. Há um enorme desperdício de espaço de armazenamento, com atributos desnecessários para cada registro. Este problema é minimizado pela compressão de dados, procedimento permitido pelo IDM-500, a máquina de banco de dados utilizada na implementação. Uma possível justificativa para esse mapeamento, é a grande eficiência obtida na recuperação de informações e na propagação de atualizações.

Smith e Smith[SMIT77] caminharam no sentido contrário e propuseram uma implementação, na qual cada classe, com todos os seus atributos, seria mapeada para uma relação distinta. Da mesma forma, essa abordagem provoca um grande desperdício de armazenamento, já que os atributos de uma superclasse devem ser necessariamente repetidos nas subclasses. Além disso, o desempenho das atualizações torna-se crítico: qualquer alteração feita nos atributos de uma superclasse deve ser obrigatoriamente propagada para todas as subclasses.

Kung[KUNG90] utiliza-se de visões para implementar generalizações em um SGBD SQL. A sua proposta é bastante flexível, a ponto de distinguir os diversos relacionamentos de especialização, citados no capítulo 3. Na sua proposta, cada

classe é representada por uma relação que contém exclusivamente os atributos diretamente associados. Como mostra a figura 5.2, os usuários vêm uma classe através de uma visão, construída pela junção da relação base e a visão associada à superclasse.

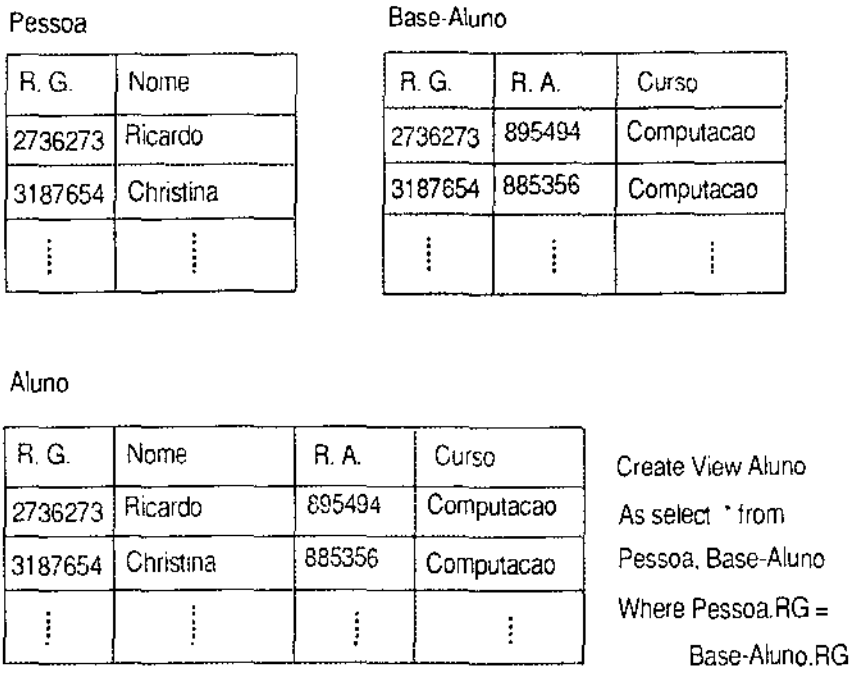


Figura 5.2: Visões na implementação de generalização

O leitor mais atento pode indagar-se a respeito do problema de atualizações de visões. Todavia, vale observar que a própria modelagem impõe que as relações resultantes estejam na terceira forma normal[BLAH88], e, por isso, pelo menos teoricamente, todas as atualizações efetuadas nessas visões poderiam ser propagadas pelo SGBD para as relações bases.

Ao contrário das abordagens anteriores, o desempenho na recuperação de instâncias é sensivelmente sacrificado, pois grande parte das recuperações irá invocar uma junção entre as diversas relações bases. Índices nos atributos chaves funcionam como um paliativo, já que o tempo gasto passa a ser proporcional ao tamanho da menor tabela em questão. Contudo, se a recuperação incluir uma qualificação e o otimizador de consultas do SGBD utilizado for suficientemente "inteligente", a ponto de verificar a possibilidade de execução de parte dessa qualificação antes da junção, o tempo gasto poderá ser reduzido ainda mais.

Outro fator a colaborar para o baixo desempenho das recuperações nesta abordagem, é a execução de junções desnecessárias, que ocorrem quando uma consulta não faz referência aos atributos das superclasses. Entretanto, isto não invalida totalmente o mecanismo de visões, pois, em geral, as aplicações modeladas dão origem a hierarquias que possuem, em média, dois a três níveis, um número consideravelmente pequeno.

Reconhecendo que em algumas aplicações o desempenho na recuperação de informações é crucial, Kung propõe a visão materializada, uma relação que é a "encarnação" de uma visão. Pequenas modificações tornam-se necessárias nas transações requisitadas, de forma a propagar as atualizações efetuadas na visão, para a sua materialização.

Outros inconvenientes inerentes ao mecanismo de visões são relacionados à retirada de instâncias. Em geral, pela semântica de uma rede de generalizações, a retirada de instâncias de uma classe deve também provocar a retirada dessas instâncias das subclasses. Esta propriedade, conflita com a propagação automática de exclusões de uma visão feita por um SGBD, que segue a direção das superclasses, ou seja, das relações bases. Surge, então, a necessidade de pequenas transformações em algumas transações, de forma a forçar um comportamento condizente. A figura 5.3 exibe a transformação de uma transação de exclusão, assumindo que a especialização de `Pessoa` em `Aluno` seja sobreposta ou disjunta.

TRANSAÇÃO RECEBIDA

```
Delete from Aluno
Where  (Aluno.Nome = "João")
```

TRANSAÇÃO MODIFICADA

```
Delete from Base-Aluno
Where Base-Aluno.RG in
  (Select Aluno.RG from Aluno
   Where Aluno.Nome = "João")
```

Figura 5.3: Transformação de uma transação de exclusão

Uma instância, quando excluída a partir de uma superclasse, desaparece apenas das visões das subclasses, permanecendo nas suas relações bases. Esses registros espúrios, provocam um terrível efeito colateral, pois, futuramente, a inclusão de instância com valores de chaves iguais a àquelas anteriormente retiradas causam o surgimento de instâncias ilegais nas visões das subclasses.

Em [WAGN88] é proposta uma extensão para árvore-B de forma a diminuir o número de acessos a disco necessários para a recuperação de informações em uma rede de generalizações, onde as classes encontram-se distribuídas em diversas relações. Esta estrutura, denominada de *ABTree*, sendo única para toda rede, encerra a necessidade das várias árvores-B que é requisitada por este mapeamento na sua implementação em um SGBD relacional típico.

As agregações também permitem alternativas para o seu mapeamento. A forma de mapeá-las, porém, está estreitamente ligada à cardinalidade do relacionamento. A figura 5.4 mostra o mapeamento da agregação `Matrícula` em uma relação distinta, pois, para relacionamentos de cardinalidade M:N, em concordância com as formas normais, este é o único mapeamento possível[BLAH88].

Aluno

R. A.	Nome	Curso
895494	Ricardo	MAT002
895495	Frederico	MAT002
906593	Nadia	ENG001
⋮	⋮	⋮

Disciplina

Codigo	Disciplina	Curso
CC100	Banco de Dados	MAT002
CC101	Sistemas Oper.	MAT002
EE100	Eletronica Digital	ENG001
⋮	⋮	⋮

Matricula

R. A.	Codigo
895494	CC101
895495	CC100
906593	EE100
906593	CC100
⋮	⋮

Figura 5.4: Mapeamento de uma agregação M:N

Já as agregações de cardinalidade 1:N e 1:1, permitem uma série de alternativas de mapeamento[KENT79], que devem ser analisadas com base em parâmetros um tanto vagos, e possivelmente não disponíveis na definição do esquema, como a freqüência de determinadas consultas.

Outro aspecto a ser observado na implementação de agregações, é o condicionamento da existência da ocorrência de um objeto na agregação à existência de todos os seus objetos componentes. Esta propriedade, está intimamente relacionada à manutenção da integridade de referências em um SGBD relacional.

Date[DATE81] revê o problema de integridade referencial, a partir de uma linguagem orientada à manutenção de restrições arbitrárias em um SGBD relacional. A sua proposta é bastante rebuscada, permitindo, inclusive, que uma chave estrangeira faça referência a um conjunto de relações com base em um quantificador. A proposta finaliza indicando as extensões que devem ser feitas ao dicionário de dados de um SGBD relacional, necessárias à captação das diversas dependências entre os atributos de um esquema.

Casanova[CASA88a] implementa um monitor que, a partir das informações das restrições de integridade referencial fornecidas na fase de projeto, modifica ou propaga as transações recebidas. Esse monitor foi projetado para receber uma seqüência qualquer de transações de atualizações, e ao fim desta, disparar as respectivas propagações, com base nas informações acumuladas das próprias atualizações.

A proposta de Cammarata[CAMM89] tem em comum com a proposta anterior a característica de efetuar a manutenção da integridade referencial após um seqüência de operações. As coincidências terminam por aí. A proposta de Cammarata restringe-se a apontar as violações das integridades. Além disso, o processo de verificação é baseado simplesmente em uma extensão do dicionário de dados, não há acúmulo de informação durante o processo de atualização, tornando-o extremamente ineficiente.

Vale observar, que como as referências no modelo relacional são baseadas em valores, a manutenção da integridade de referências, torna-se um problema mais complexo que nos modelos semânticos que se utilizam de *surrogates*. Por exemplo, quando o valor de um atributo chave em uma relação à qual é feita uma referência é modificado, deve-se propagar a alteração para todos os atributos que a fazem referência.

O acesso aos atributos das entidades componentes, a partir de um objeto composto, deve ser, também, um problema enfocado. Stonebraker[STON84] utiliza-se do tipo de dados QUEL, que permite que os atributos tenham, como valor, comandos da linguagem de consulta QUEL. Esse tipo de dados é utilizado para simular uma série de facilidades, inclusive a de agregação. A recuperação de objetos é particularmente potencializada, já que é permitido a um atributo do tipo QUEL, recuperar instâncias componentes em diferentes tabelas. A figura 5.5 ilustra esse aspecto.

```
Append to
Aluno( RA   = "895494",
       Nome = "Ricardo",
       Emprestimos = "Range of L is Livro
                      Retrieve (L.All)
                      Where L.RA = "895494"
                      Range of R is Revista
                      Retrieve (R.All)
                      Where R.RA = "895494")
```

Figura 5.5: Tipo de dados QUEL

Para a recuperação das instâncias componentes, mais uma vez é utilizado o mecanismo de visões, associado à modificação de transações. O sistema, ao receber uma transação de consulta que envolve um tipo QUEL, seleciona os registros da tabela de origem que satisfazem à qualificação.

Para cada registro selecionado, serão criadas uma ou mais visões, a partir das consultas contidas nos atributos do tipo QUEL. Para cada visão, serão selecionados registros, com base na parte da qualificação que contém os atributos relativos a visão criada. A figura 5.6 mostra este processo.

CONSULTA RECEBIDA

```
Range of A is Aluno
Retrieve (A.Empréstimo.Nome)
Where A.RA = "895494" and
      A.Emprestimo.Situação =
      "Atraso"
```

CONSULTA TRANSFORMADA

```
Range of A is Aluno
Retrieve (A.Empréstimo)
Where A.RA = "895484"

Para cada visão criada a partir
do
valor de A.Empréstimo, efetue
esta consulta:

Retrive (E.Nome)
Where E.Situação = "Atraso"
```

Figura 5.6: Recuperação de instâncias componentes a partir de visões

Como visto no capítulo 3, o GEM adota uma abordagem não tão poderosa, porém, extremamente simples. Ao incluir um objeto em uma agregação, o usuário deve fornecer predicados que identificam cada instância componente. Como o acesso às instâncias componentes de uma tabela é sempre feito através de junções, o GEM simplesmente fornece uma sintaxe mais fácil para tal.

O mapeamento dos atributos multivalorados não merece maiores comentários, visto que é equivalente a uma agregação de cardinalidade M:N, tendo, portanto, a mesma representação. Porém, a natureza dos operadores de conjunto, aplicáveis nos atributos multivalorados, é complementamente diferente dos operadores dos atributos atômicos, e, apesar das linguagens de manipulação relacionais serem orientadas a conjuntos, elas não oferecem qualquer tipo de suporte a esses operadores.

Tsur[TSUR84] mostra que todos os operadores de conjunto, podem ser simulados através de uma transformação de consultas. Esta transformação é baseada em princípios extremamente simples da teoria de conjuntos. Por exemplo, um conjunto A está contido em um conjunto B sempre que a cardinalidade de A for igual a cardinalidade de $A \cap B$. Sendo assim, o padrão GEM:

$$\{X_1 \text{ by } Y_1 \text{ where } Z_1\} < \{X_2 \text{ by } Y_2 \text{ where } Z_2\}$$

é traduzido para QUEL como:

```
count(X1 by Y  where Z ) =
count(X1 by Y1,Y2 where Z1 and Z  and X =X  and Y2=Y ).
```

5.2 Abordagem S-SQL

A seção anterior mostrou uma variedade de alternativas utilizadas na extensão de um SGBD relacional. A análise dessas abordagens leva a acreditar que, em geral, a eficiência sacrificou a tomada de soluções simples e elegantes. A implementação S-SQL discorda dessa filosofia, e, a partir desse ponto de vista, foram desenvolvidos os seguintes princípios que nortearam a sua implementação:

- Evitar a redundância de informações;
- Manter o banco de dados sempre em estado consistente, após a execução de qualquer transação;
- Tentar evitar, na medida do possível, incorporar à ferramenta qualquer tipo de computação que possa ser efetuada pelo SGBD SQL;
- Ser portátil.

5.2.1 Mapeamento

Basicamente, toda classe irá corresponder a uma única tabela. E, sendo assim, a ferramenta, através da transformação de consultas, será a responsável pela transparência na manipulação e apresentação das informações ao usuário.

A cada tabela a ferramenta acrescentará um único atributo, destinado a conter o *surrogate* do registro. A única exceção do mapeamento trata-se das classes que possuem atributos multivalorados. Nesse caso, a classe será representada por N + 1 relações distintas, onde N indica o número de atributos multivalorados. As relações que os representam irão conter unicamente dois atributos: um, destinado a armazenar os elementos do próprio atributo, e outro, destinado a associar o elemento à sua instância na tabela base. A figura 5.7 exemplifica este mapeamento.

Aluno

Aluno#	RA	Curso	Media
00043	886756	00021	8.5
00056	918283	00019	9.0
⋮	⋮	⋮	⋮

Aluno-Esportes

Aluno#	Esportes
00043	Futebol
00043	Basquete
00056	Volei
⋮	⋮

Figura 5.7: Mapeamento de atributos multivalorados

Com o intuito de melhorar o desempenho das recuperações, são criados índices para cada chave declarada, assim como, para os *surrogates* que identificam as instâncias em cada tabela.

5.2.2 Tradução de Transações

Esta seção visa ilustrar o processo de transformação de transações S-SQL em SQL. É interessante notar, que este processo não é unívoco, pois, em boa parte dos casos, uma transação S-SQL dá origem a várias transações SQL. Os aspectos de clareza e naturalidade da transformação foram privilegiados em dentrimento da eficiência da mesma. Também não será comentada a validação semântica das transações. O objetivo maior desta seção é o de verificar a adequação de uma linguagem relacional, em particular SQL, como base de implementação de uma LMD semântica.

A transformação de um atributo que utiliza a notação de ponto, é bastante simples e intuitiva e já foi comentada na seção 4.4. Em geral, o processo de transformação irá variar de acordo com o local de utilização do atributo na

transação. Supondo um atributo A, associado à variável-tupla V, que, por sua vez, está ligada a uma tabela T, em uma cláusula Select, a interface processará a seguinte transformação:

S-SQL

```
Select V.A.A1.A2...An
From T V
Where ...
...
...
```

SQL

```
Select V1.A
From   T V1,
       T1 V2,
       ...,
       Tn Vn
Where V.A = V1.T# and
      V1.A1 = V2.T# and
      ...
      Vn-1.An-1 = Vn.T# and
      (...)
```

Em uma cláusula Where, o processo difere levemente:

S-SQL

```
Select ...
From   T V
Where V.A.A1.A2...An Θ X
...
```

SQL

```
Select ..
From   T V
       T1 V1,
       ...,
       Tn Vn
Where ...
       (V.A = V1.T# and
       V1.A1 = V2.T# and
       ...
       Vn-1.An-1 = Vn.T# and
       Vn.An Θ X
       ...)
```

V_1 ... V_n são identificadores únicos gerados pela ferramenta. Θ e X representam, respectivamente, operador e operando válidos.

Na inclusão de uma instância, para cada tipo de atributo classe (*surrogate*) que houver, a interface gera uma consulta baseada no predicado dado, a fim de recuperar o *surrogate* associado a cada instância componente. Sendo Sur uma variável hospedeira, a transação de inclusão da figura 4.7 na página 76, originaria a seguinte transação SQL:

```
Select Instituto#
Into :Sur
From Instituto
Where Nome = "Instituto de Matemática"
```

Baseada nos valores retornados pela variável Sur que retém sempre o último *surrogate* gerado ou recuperado pela ferramenta, é criada uma nova transação de inclusão, substituindo-se as cadeias que definem os predicados pelos *surrogates* associados às instâncias componentes.

Como será visto na seção 5.3.5, as transações de exclusão, são transformadas em transações de consulta, para que a ferramenta, através do módulo de propagação de exclusões, efetue não só a exclusão requisitada, bem como a sua propagação pelas tabelas relacionadas.

Já as transações de atualização, têm a sua qualificação estendida quando esta envolve atributos de classes componentes:

S-SQL

```
Update T Set (...)
Where T.A.A1.A2 Θ X
...
```

SQL

```
Update T Set (...)
Where T# in
    ( Select V.T#
      From  T V
            T V1,
            ...,
            Tn Vn
      Where (V.A = V1.T1# and
             V1.A1 = V2.T2# and
             ...
             Vn-1.An-1 = Vn.Tn#
             and V.A2 Θ X
             ...)
```

Para as transações que recuperam instâncias de classes participantes de uma hierarquia de generalização, o processo de transformação é, de uma certa maneira, relativamente próximo ao processo de tradução de junções implícitas. A diferença básica é que a interface não consegue, baseada no conteúdo sintático da transação, relacionar os atributos em questão às superclasses em que estão definidos. Para isso, ela deve recorrer às informações contidas no dicionário de dados.

Sendo S uma subclasse qualquer e SP uma superclasse de S, uma consulta que refere-se a um atributo SP_i de SP, a partir de S, teria a seguinte transformação:

S-SQL

```
Select SP.
From S
Where …
```

SQL

```
Select SP
From S  V ,
     SP V ,
     … ,
Where V .S# = V .SP# and
      …
```

O operador de pertinência IS-A é traduzido simplesmente para o operador =, enquanto que o operando que identifica a classe, é traduzido para o padrão <Nome-da-classe>#. A transação contida na figura 4.10 na página 76 teria a seguinte tradução:

```
Select P.RG, P.Nome
From   Pessoa P,
       Aluno A,
       Funcionário F
Where  (P.Pessoa# = A.Aluno# and
        P.Pessoa# = F.Funcionário#)
```

As transações de inclusão em uma rede de generalizações são passíveis do mesmo problema que as transações de consulta. A ferramenta deve determinar, com base no dicionário de dados, a qual superclasse está associado cada atributo. A figura 5.8 mostra a transformação de uma transação que inclui um objeto na classe Professor. As duas primeiras transações SQL, todavia, são destinadas a tornar disponível um *surrogate*.

S-SQL

```
Insert into Professor(Nome,RG,Endereço,Matrícula,Cargo,
                      Salário,Titulação,Depto)
Values("Isaac Douglas","454.345", Rua="Euclides da Cunha 20" and
        Bairro="Graça","874788",Nome="Prof. Titular", 3000000,
        "MSc",Nome="Ciência da Computação")
```

SQL

```
Update Surrogate
Set Surrogate#=Surrogate# + 1

Select Surrogate#
Into :Sur
From Surrogate

Insert into Pessoa(Pessoa#,Nome,RG,Endereço)
            Values (:Sur,"Isaac Douglas","454.345",00034)

Insert into Funcionário(Funcionário#,Matrícula,Cargo,Salário)
            Values (:Sur,"874788",00403,3000000)

Insert into Professor(Professor#,Titulação,Depto)
            Values (:Sur,"MSc",00501)
```

Figura 5.8: Transformação de uma inclusão em uma subclasse

As transações de inclusão que se utilizam da cláusula `Surrogate from` geram uma consulta adicional que retorna o valor do *surrogate* associado à instância incluída. A figura 5.9 ilustra o processo de transformação da transação S-SQL da figura 4.13 na página 78. Como já foi dito no capítulo 4, se o predicado que determina o *surrogate* não validar ou validar mais de uma instância, a interface rejeitará a inclusão.

```
Select Funcionário#
Into :Sur
From Funcionário
Where (Matrícula = "896755")

Insert into Professor (Professor#, Titulação, Depto)
Values (:Sur,"MSc", 00101)
```

Figura 5.9: Tradução de inclusão com a cláusula `Surrogate From`

Caso a classe na qual está sendo feita a inclusão, seja participante de uma especialização disjunta ou particionada, a interface gerará comandos de exclusão, para cada subclasse da categoria, baseados no *surrogate* designado pela cláusula `Surrogate From`.

Da mesma forma, as inclusões nas classes derivadas com base em predicados também possuem um processo de transformação extremamente simples. Sendo `S` uma classe derivada com base em um predicado `P`, a partir da superclasse `SP`, para cada inclusão em `SP`, a interface geraria a seguinte transação complementar:

```
Insert into S(S#)
Select SP#
From SP
Where P and SP# = :Sur
```

Vale lembrar, que o predicado `P` foi expresso em S-SQL, e, provavelmente, a transacao acima necessitaria ser traduzida.

Já as transações que provocam modificações nos atributos constituintes de um predicado, determinam uma transformação mais complexa. De uma forma geral, uma transação que obedece o formato padrão `Update SP Set (SP`$_1$`=Expr`$_1$`, … ,SP`$_n$`=Expr`$_n$`) Where Q`, determina o seguinte processo de tradução:

```
Insert into TmpSur(Surrogate#)
Select SP#
From SP
Where Q

Update SP
Set (SP1=Expr1, … ,SPn=Exprn)
Where SP# in (Select Surrogate# from TmpSur)

Insert into S(S#)
Select SP.SP#
From SP,
     TmpSur
Where SP.SP#=TmpSur.Surrogate# and P and
      not (SP.SP# in (Select S.S# from S))

Delete from S
Where S.S# in (Select SP.SP# from SP, TmpSur
               Where SP.SP#=TmpSur.Surrogate# and not P)
```

A tabela TmpSur tem a exclusiva função de armazenar as instâncias que foram modificadas pelo comando Update. Essa tabela não é indispensável ao processo de tradução, contudo, a sua inclusão otimiza a execução das transações de inclusão e exclusão na subclasse S, pois somente as instâncias de SP que tiveram os valores dos atributos modificados têm o predicado P verificado.

Como foi visto no capítulo 4, uma subclasse S pode possuir um predicado de derivação P que não é baseado nos valores dos atributos de uma superclasse SP, mas no valor de um atributo $C_{..}$, pertencente a uma classe C e de domínio SP. Para essas classes, o processo de tradução de transações difere em alguns pontos do processo anterior. Cada inclusão de um objeto em C gera a seguinte transação complementar:

```
Insert into S(S#)
Select C.C..
From C
Where C.C# = :Sur and
      not (C.C.. in (Select S.S# from S))
```

Da mesma forma, as transações que envolvem a modificação do valor do atributo de pertinência $C_{..}$ requisitam transformações mais complexas. Como mostra a tradução seguinte, o processo requer duas tabelas de trabalho, sendo uma para armazenar as instâncias de C que satisfazem a uma qualificação Q e outra para guardar os valores de $C_{..}$ dessas mesmas instâncias, anteriores à transação dada:

```
Insert into TmpSur(Surrogate#)
Select C.C#
From C
Where Q

Insert into TmpSur2(Surrogate#)
Select distinct C.C#
From C
Where C# in (Select Surrogate# from TmpSur)

Update C
Set (C1=Expr1, ... ,Cn=Exprn,...)
Where C# in (Select Surrogate# from TmpSur)

Delete from S
Where S.S# in
   (Select Surrogate#
    From TmpSur2
    Where not (Surrogate# in (Select C  from C)))

Insert into S(S#)
Select distinct C.C
From C,
     TmpSur
Where TmpSur.Surrogate# = C.C# and
      not (C.C in (Select S.S# from S))
```

Já a tradução da exclusão de instâncias em uma classe C, com base em uma condição Q, é um subconjunto das transações SQL acima. O leitor está convidado a deduzí-la.

Uma classe derivada a partir de uma herança múltipla total também é vista pela ferramenta, como um caso especial de derivação com base em um predicado. Sendo SP_1, SP_2, ..., SP_n superclasses de uma subclasse S, qualquer inclusão em uma das superclasses produz a seguinte transação adicional:

```
Insert into S(S#)
Select SP1.SP1#
From SP1
Where SP1.SP1# = :SUR and
      SP2.SP2# in (Select SP2# from SP2) and
        ...
      SPn.SPn# in (Select SPn# from SPn)
```

De uma forma geral, todas as transformações envolvendo atributos multivalorados implicam em uma junção entre a tabela base da classe e a tabela

que armazena os elementos do atributo. A figura 5.10 exibe a transformação de um atributo multivalorado S_i, pertencente a uma tabela S, em uma cláusula Select.

S-SQL

```
Select V1.Si
From S V1, ...
Where ...
```

SQL

```
Select V .S
From S V ,
     S_S  V
Where V .S# = V .S# and
      (...)
```

Figura 5.10: Transformação envolvendo um atributo multivalorado

Da mesma forma que o operador IS-A, o operador IN é simplesmente transformado no operador =. A figura 5.11 mostra a tradução da transação contida na figura 4.17 na página 80.

```
Select A.RA, P.Nome, C.Nome
From   Pessoa P,
       Aluno A,
       Curso C,
       Aluno_Esportes AE
Where  P.Pessoa# = A.Aluno# and
       A.Curso = C.Curso# and
       A.Aluno# = AE.Aluno# and
       ("Futebol" = AE.Esportes)
```

Figura 5.11: Tradução de uma transação que utiliza-se do operador IN

A implementação dos operadores de conjunto seguiu a mesma abordagem adotada pelo GEM, embora tenham sido necessárias algumas adaptações, devido às idiossincrasias apresentadas pela SQL[DATE84].

Por exemplo, na implementação da operação de igualdade, vale o mesmo princípio básico: dois conjuntos A e B são iguais, se, e somente se, a cardinalidade de ambos for igual à cardinalidade da interseção. Sendo assim, as transações que apresentam o padrão:

```
Select …
From S1 V1,
     S2 V2,
     …
Where {V1.S1n  Where Q1 Group by V1.S1n} =
      {V2.S2n  Where Q2 Group by V2.S2n}
```

são traduzidas para:

```
Select …
From  S1 V1,
      S2 V2,
      …
Where (V1.S1n  = V2.S2m and Q1 and Q2 )
Group by V1.S11, V2.S21, …
Having count(*) = (Select count (distinct Vn+1.S1n)
                   From S1 Vn+1
                   Where Q1 and
                   V1.S1n = Vn+1.S1n) and
       count(*) = (Select count (distinct V  .S  )
                   From S2 V
                   Where Q2 and
                   V .S  = V  .S  )
```

Por definição, um conjunto não pode possuir elementos repetidos, e, por isso, a cláusula `distinct` é necessária às subconsultas. A figura 5.12 mostra a tradução da transação da figura 4.20 na página 81, que retorna os alunos que se matricularam em todas as disciplinas de um determinado departamento.

```
Select A.RA, A.Nome
From  Matrícula M,
      Disciplina D,
      Oferta O,
      Aluno A,
      Departamento DE
Where M.Aluno = A.Aluno# and
      (M.Oferta = O.Oferta# and
      O.Disciplina = D.Disciplina# and
      D.Depto = DE.Departamento# and
      DE.Nome = "Hidráulica")
Group by M.Aluno, A.RA, A.Nome
Having count(*) = (Select count(distinct D1.Disciplina#)
                   From Disciplina D1,
                        Departamento DE1
                   Where D1.Depto = DE1.Departamento# and
                         DE1.Nome = "Hidráulica" )
```

Figura 5.12: Tradução da transação da figura 4.20

Igualmente, um atributo multivalorado S , pertencente a uma tabela S, quando utilizado em uma qualificação que envolve operadores de conjuntos, é entendido pela ferramenta como definindo implicitamente o conjunto (S_S .S Group by S_S_n.S#). A figura 5.13 exemplifica a tradução de uma consulta S-SQL que emprega operadores de conjuntos e atributos multivalorados.

```
S-SQL

Select A1.RA, A2.RA
From Aluno A1,
     Aluno A2
Where A1.RA != A2.RA and
      A1.Esportes = A2.Esportes


SQL

Select A1.RA, A2.RA
From Aluno A1,
     Aluno A2,
     Aluno_Esportes AE1,
     Aluno_Esportes AE2
Where A1.Aluno# = AE1.Aluno# and
      A2.Aluno# = AE2.Aluno# and
      (A1.RA != A2.RA and
       AE1.Esportes = AE2.Esportes)
Group by AE1.Aluno#, AE2.Aluno#, A1.RA, A2.RA
Having count(*) = (Select count (distinct AE3.Esportes)
                   From Aluno_Esportes AE3
                   Where AE3.Aluno# = AE1.Aluno#)
       and
       count(*) = (Select count (distinct AE4.Esportes)
                   From Aluno_Esportes AE4
                   Where AE4.Aluno# = AE2.Aluno#)
```

Figura 5.13: Recuperação dos alunos que praticam os mesmos esportes

Pelas traduções passadas, o leitor pode facilmente inferir o processo de transformação de uma consulta, cuja cláusula `Select` possui uma função aplicada a um atributo multivalorado. Entretanto, as funções aplicadas a atributos multivalorados também podem ser usadas em uma cláusula `Where` para compor uma qualificação. Em uma qualificação, cada comparação formada por uma função F, aplicada a um atributo multivalorado S_S_n.S, em conjunção com um operador Θ, e um operando X, é traduzida para a subconsulta:

```
Exists (Select *
        From S_S_n
        Where S_S_n.S# = S.S#
        Having F(S_S_n.S_n) Θ X)
```

A figura 5.14 mostra a tradução da consulta da figura 4.19 na página 81.

```
Select A.Nome, AVG(MN.Notas)
From Matrícula M,
     Matrícula_Notas MN,
     Aluno A,
     Oferta O,
     Disciplina D
Where  A.Aluno# = M.Aluno and
       M.Matrícula# = MN.Matrícula# and
       (M.Oferta = O.Oferta# and
        O.Disciplina = D.Disciplina# and
        D.Nome = "Contabilidade I" and
        Exists (Select *
                From Matrícula_Notas MN1
                Where MN1.Matrícula# = M.Matrícula#
                Having AVG(MN1.Notas) > 6.0))
Group by MN.Matrícula#, A.Nome
```

Figura 5.14: Tradução envolvendo funções e atributos multivalorados

A figura 4.22 na página 82, mostra uma transação para inclusão de elementos em atributos multivalorados. Essas transações obedecem o padrão `Update S Set(... , S_n=+{E_n1, ... ,E_nk}, ...) Where Q`. Para cada atributo multivalorado `S_n` contido na cláusula `Set`, a interface produz a seguinte transação:

```
Insert into S_S_n (S#, S_n)
Select S.S#, S_S_n_Temp.Valor
From  S,
      S_S_n_Valor
Where Q
Minus
Select S_S_n.S#, S_S_n.S_n
From  S,
      S_S_n
Where Q and
      S.S# = S_S_n.S#
```

Onde `S_S_n_Temp` é uma tabela de trabalho, em que o atributo `Valor`, é destinado a conter os elementos E_{n1}, ... , E_{nm}.

5.2.3 Traduções Inconsistentes

Um problema de ordem semântica surge através de atributos multivalorados. A figura 5.15, exibe uma consulta destinada a retornar alunos e seus respectivos esportes. Todavia, devido à inexistência de suporte ao *outer join*[CODD79], só são retornados aqueles que praticam esportes.

```
Select RA, Nome, Esportes
From Aluno
```

Figura 5.15: Recuperação de alunos e esportes praticados

Semanticamente, isto pode, e deve, ser considerado um erro, pois, caso o usuário só desejasse verificar os alunos praticantes de esportes, deveria ter acrescentado a consulta a cláusula `Where Exists(Esportes)`. O problema acima pode ser exemplificado de uma maneira mais enfática. A figura 5.16 exibe uma consulta S-SQL e a sua respectiva tradução. O que se deseja com a mesma, é selecionar os alunos que praticam mais do que dois esportes, bem como, o aluno de R.A. 895494, seja qual for o número de esportes que ele pratique. Contudo, de acordo com a tradução, se o aluno de R.A. 895494 não praticar esportes, simplesmente, ele não será selecionado.

```
S-SQL

Select RA, Esportes
From Aluno
Where Count(Esportes) > 2 or
      RA = '895494'

SQL

Select A1.RA, AE1.Esportes
From Aluno A1,
     Aluno_Esportes AE1
Where A1.Aluno# = AE1.Aluno# and
      (Exists(Select *
              From Aluno_Esportes AE2
              Where AE2.Aluno# = A1.Aluno#
              Having Count(AE2.Esportes) > 2)
       or A1.RA = '895494')
Group by AE1.Esportes
```

Figura 5.16: Tradução incorreta de uma consulta S-SQL

Contudo, não há incorreções na tradução. O problema é o mesmo: só participam da qualificação os alunos que praticam esportes. Visivelmente, não é essa a intenção do usuário na especificação da consulta S-SQL. Em resumo, só é possível uma correta implementação de atributos multivalorados, se o SGBD SQL prover o *outer join*.

Outro problema que deve ser, no mínimo, classificado como estético, diz respeito às consultas que possuem mais de um atributo multivalorado em uma cláusula `Select`. A figura 5.17 exibe uma consulta S-SQL destinada a recuperar todos os esportes e passatempos[17] praticados pelos alunos, junto à sua respectiva tradução.

17. Passatempos é também um atributo multivalorado da classe [illegible].

S-SQL

```
Select RA, Esportes, Passatempos
From Aluno
```

SQL

```
Select A.RA, AE.Esportes, AP.Passatempos
From Aluno A,
     Aluno_Esportes AE,
     Aluno_Passatempos AP
Where A.Aluno# = AE.Aluno# and
      A.Aluno# = AP.Aluno#
```

Figura 5.17: Alunos e seus respectivos esportes e passatempos

Com base nas operações da álgebra relacional, o resultado da consulta acima deve ser obtido através do produto cartesiano das tabelas `Aluno`, `Aluno_Esportes` e `Aluno_Passatempos`, seguido de uma restrição. Com isso, como exemplifica a figura 5.18, a consulta produzirá uma saída na qual cada esporte praticado por um aluno está associado a todos os seus passatempos e vice-versa.

RA	Esportes	Passatempos
895494	Volei	Fotografia
895494	Volei	Selos
895494	Futebol	Fotografia
895494	Futebol	Selos
895599	Basquete	Pesca
895599	Basquete	Praia
895599	Basquete	Cinema

Figura 5.18: Saída hipotética envolvendo dois atributos multivalorados

Este é um problema implicitamente relacionado à quarta forma normal: esportes e passatempos são fatos independentes. Apesar de não ser possível obtê-la através de uma linguagem relacional, é desejável uma tradução na qual os valores dos atributos multivalorados fossem impressos lado a lado.

5.2.4 Suporte SQL

De uma forma geral, as traduções que envolveram as abstrações de agregação e generalização foram efetuadas de uma maneira simples. Já as transações que incluiram atributos multivalorados, requisitaram traduções de maior complexidade. Este fato, de uma certa forma, ressaltou algumas críticas efetuadas por Date[DATE84] à linguagem SQL.

Uma das inúmeras críticas que pôde ser comprovada por este trabalho, foi a falta de ortogonalidade das funções SQL. Segundo Date, as chamadas de funções são expressas de uma maneira heterodoxa, o que restringe a sua utilização a um certo número de ocasiões. Em particular, não é permitido o "aninhamento" de chamadas.

Outras restrições, aparentemente arbitrárias, estão relacionadas à composição de expressões. Uma delas, determina que uma comparação, quando envolve uma subconsulta, deve sempre obedecer a seguinte ordem: `x Θ (S)`, sendo `x` e `Θ` operando e operador válidos, e `S` uma subconsulta na qual deve estar aplicada uma função de grupo. Na verdade, esta restrição ainda é mais drástica: a linguagem SQL não permite a composição de uma comparação com base nos valores retornados por duas sub-consultas.

Esta simples restrição tem implicações abrangentes. Devido a ela, é quase que obrigatório o uso das cláusulas `Group by` ou `Having` para solucionar consultas que envolvem grupos. A tradução dos operadores de grupo, por exemplo, seria muito mais natural, caso a mesma não existisse.

5.3 Arquitetura

A interface S-SQL foi implementada como uma camada da SQL, versão 2.0, do SGBD ORACLE, e todo o protótipo foi escrito em linguagem C da MICROSOFT, versão 5.0. Ela executa sobre o sistema operacional MS-DOS, em qualquer configuração que disponha de no mínimo 2MB de memória, requisito necessário ao ORACLE, versão 5.1, para coexistir com aplicações C. A implementação visa ser altamente portátil, sobretudo pela utilização dos comandos SQL ter sido restrita ao conjunto padrão, bem como, pela razoável compatibilidade existente entre as diversas versões dos compiladores C.

A figura 5.19 ilustra a arquitetura da interface. Basicamente, para uma melhor compreensão semântica das transações S-SQL, tornou-se necessária a extensão do dicionário de dados do SGBD relacional, bem ao estilo das relações propostas por Codd no RM/T[CODD79]. O módulo de interpretação de comandos S-SQL efetua a análise léxica e sintática das transações requisitadas pelo usuário. A manutenção das informações contidas no dicionário de dados, é uma tarefa efetuada pelo módulo de criação e manutenção de esquemas. O módulo de tratamento de transações é o responsável pela transposição de uma transação S-SQL para transações SQL. O módulo de propagação de exclusões propaga essas operações pelas diversas tabelas de um esquema. O módulo de execução dinâmica possibilita a execução de recuperações. As seções seguintes detalham esses módulos.

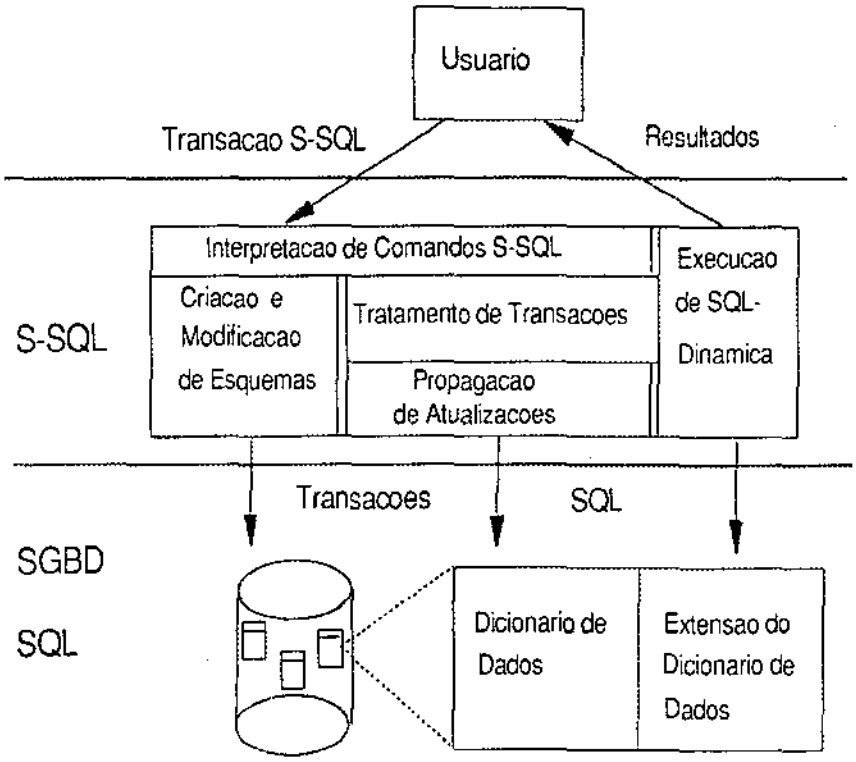


Figura 5.19: Arquitetura da interface

5.3.1 Dicionário de Dados

A extensão do dicionário de dados tornou possível todo o processo de captação semântica. Algumas tabelas que compõem o dicionário de dados padrão SQL também são utilizadas, mas, por si só, elas não seriam suficientes. As seguintes tabelas formam a extensão do dicionário S-SQL:

- **Surrogate** – Tabela que contém um único registro que sempre guarda o último *surrogate* utilizado.

- **Categoria** – Tabela que descreve cada categoria existente no esquema. Um dos campos descreve a variação da especialização empregada, enquanto o outro, a forma de especialização: simples, múltipla, derivada com base em um predicado ou em um atributo. O atributo `NCat`, é uma chave artificial, criada com o intuito de identificar cada categoria.

- **Super** – Tabela que associa as superclasses às categorias. O campo `Preced` indica a precedência da superclasse na definição de uma herança múltipla.

- **Sub** – Tabela que associa as subclasses às categorias.
- **P-Condição** – Tabela que guarda a condição de pertinência das subclasses derivadas com base em um predicado.
- **P-Atributo** – Tabela que armazena atributo e classe relacionados às subclasses derivadas com base em um atributo.
- **Chave** – Tabela que armazena as chaves definidas para uma relação. Outra vez, tornou-se necessário criar uma chave artificial.
- **Ch-Atributo** – Tabela que mantém, para cada chave, os seus atributos componentes.
- **Referência** – Tabela que armazena os atributos que, apesar de fazerem referência a outras classes, não fazem parte da chave da relação a qual pertecem.
- **Multivalorado** – Tabela que armazena os atributos multivalorados, com os respectivos domínios.

Para um melhor entendimento, a figura 5.20 exibe parte do conteúdo do dicionário de dados que corresponde ao esquema da figura 4.1.

Surrogate

Surrogate#
00101

Categoria

NCat	Variacao	Tipo
00001	Covering	S
00002	Partial	M
00003	Overlaping	S
00004	Derived	P
00005	Derived	A
⋮	⋮	⋮

Super

NCat	Super	Preced
00001	Pessoa	0001
00002	Funcionario	0001
00002	Aluno	0002
⋮	⋮	⋮

Sub

NCat	Sub
00001	Aluno
00001	Funcionario
00002	Monitor
⋮	⋮

P-Condicao

NCat	Condicao
00004	Media > 9.5
⋮	⋮

P-Atributo

NCat	Atributo	Classe
00005	Coordenador	Pesquisa
⋮	⋮	⋮

Chave

NChave	Classe
00012	Pessoa
00013	Aluno
00014	Funcionario
00015	OFerta
00016	Matricula
⋮	⋮

Ch-Atributo

NChave	Atributo	Dominio
00012	RG	Char
00013	RA	Char
00014	Matricula	Char
00015	Disciplina	Disciplina
00015	Turma	Int
⋮	⋮	⋮

Referencia

Classe	Atributo	Dominio
Aluno	Curso	Curso
Funcionario	Cargo	Cargo
Monitor	Disciplina	Disciplina
⋮	⋮	⋮

Multivalorado

Classe	Atributo	Dominio
Aluno	Esportes	Char
⋮	⋮	⋮

Figura 5.20: Dicionário de dados S-SQL

5.3.2 Interpretação de Comandos S-SQL

Esse módulo foi implementado através dos utilitários LEX e YACC que, a partir de uma especificação gramatical, geram automaticamente procedimentos para o reconhecimento léxico e sintático, respectivamente. Objetivando simplificar a implementação, além de evitar a redundância na interpretação de comandos, a gramática definida no apêndice B, somente reconhece as extensões que formam as LDD e LMD S-SQL.

5.3.3 Criação e Manutenção de Esquemas

O módulo de criação e manutenção de esquemas é o responsável por manter a correspondência entre a intenção, representada pelo dicionário de dados, e a extensão, representada pelos valores contidos nas tabelas e atributos do banco de dados.

A proibição de categorias múltiplas, ou seja, a não permissão da especialização de dois ou mais conjuntos de subclasses a partir de um mesmo grupo de superclasses, foi uma restrição imposta na definição das redes de generalizações, com o objetivo de simplificar não só a implementação da interface, bem como, o projeto da própria LMD. Esta restrição impede a ocorrência de conflitos na definição dos tipos de especializações. Por exemplo, com as extensões oferecidas pela LMD S-SQL, duas categorias oriundas de um mesmo conjunto de superclasses e, ambas, definidas por uma especialização particionada, como poderia se dar a inclusão de instâncias nas subclasses?

Alguns comandos oferecidos pela LDD S-SQL, como a exclusão de um atributo, não possuem contrapartida na LDD SQL, e a sua implementação, deve ser feita através de uma seqüência de comandos da LDD e LMD SQL. No protótipo, entretanto, não foi implementado a parte deste módulo referente a manutenção de esquemas.

5.3.4 Tratamento de Transações

O processo de transformação de transações já foi abordado na subseção 5.2.2. Porém, este módulo não se resume à implementação deste processo e, por efetivar

toda a validação e parte da manutenção de integridade do banco de dados, ele pode ser considerado a parte central da ferramenta. É claro que todos os procedimentos efetuados por esse módulo, baseiam-se nas informações contidas na extensão do dicionário de dados.

Em geral, validar uma transação significa verificar a sua semântica. Por exemplo, nas inclusões de instâncias em uma agregação, verificar para cada predicado a qualificação de apenas uma instância componente. Contudo, em algumas ocasiões, a validação de transações provoca a geração automática de atualizações. Um bom exemplo são as exclusões automáticas de instâncias das classes derivadas que não mais satisfazem o predicado de pertinência.

Apenas parte deste módulo foi implementado, como algumas extensões dos comandos `Insert` e `Select`.

5.3.5 Propagação de Exclusões

Apesar deste módulo não ter sido implementado, discute-se nesta seção alguns do seus aspectos. O módulo de propagação de exclusões é o responsável por disparar as exclusões de instâncias nas redes de generalizações e agregações, que são provocadas pelas transações de retiradas. Como em [CASA88], o módulo de tratamento de transações não instrui o SGBD a executar as transações de exclusão. Quando essas transações são solicitadas, ou mesmo geradas por esse módulo, ele simplesmente as transforma em transações de consulta que retornam os *surrogates* das instâncias a serem excluídas. A figura 5.21 mostra este processo.

TRANSAÇÃO RECEBIDA

```
Delete
From Aluno
Where  (Aluno.Nome = "João")
```

TRANSAÇÃO MODIFICADA

```
Insert into AlunoSur#(Aluno#)
Select (Aluno#)
From Aluno, Pessoa
Where Aluno# = Pessoa# and
      Nome = "João"
```

Figura 5.21: Transformação de uma transação de exclusão S-SQL

Para cada tabela definida em um esquema, a interface cria automaticamente a tabela <nome-da-tabela>Sur#, que irá conter um único atributo, destinado a guardar os *surrogates* gerados pelas transações de retiradas.

Grande parte das propagações a serem efetuadas se devem à manutenção da integridade referencial entre as diversas tabelas representadas em um esquema S-SQL. Agregações são intimamente relacionadas ao mecanismo de integridade de referências. Idem para os atributos multivalorados.

Uma subclasse S derivada com base em um atributo, implica em uma dependência de inclusão entre o atributo S# e o atributo base de derivação C_n de uma classe C[CASA88b].

Não é surpreendente notar que pela maneira como foi mapeada, parte da integridade semântica a ser mantida em uma rede de generalização também pode ser representada através de integridade referencial: uma subclasse faz referência aos atributos da superclasse e, dessa forma, quando instâncias de uma superclasse são retiradas, não fazem mais sentido as referências feitas a essas instâncias, e, portanto, podem ser automaticamente retiradas.

Todavia, a propagação no sentido contrário, ou seja, das subclasses para as superclasses, requer uma maior necessidade de informações, pois dependerá da forma de especialização envolvida. Por exemplo, para se propagar a retirada de instâncias de uma subclasse derivada através de uma especialização coberta, é necessário verificar quais, dos registros excluídos, se encontram instanciados nas outras subclasses da categoria. A figura 5.22 exibe a propagação, para a superclasse Pessoa, de uma transação de exclusão iniciada na subclasse Aluno.

```
Insert into PessoaSur#(Pessoa#)
Select Aluno#
From AlunoSur
Where not (Aluno# in (Select Funcionário# from Funcionário))
```

Figura 5.22: Propagação de exclusão para uma superclasse

O leitor deve notar que a ferramenta não retira imediatamente os registros, pois cada exclusão pode propagar-se hierarquia acima, ou mesmo, pelas

agregações. A retirada de registros dar-se-á, efetivamente, quando não é mais possível haver propagações.

Alguns SGBDs[IBM88] oferecem algum suporte à manutenção de integridade referencial. Dessa forma, o emprego dessa característica pode facilitar sobremaneira a implementação deste módulo. Esta opção, entretanto, sacrifica o aspecto de portabilidade da ferramenta. Uma opção válida seria a de reescrevê-lo para os SGBDs que provêem esse tipo de suporte.

Para finalizar, vale lembrar que esse é o módulo responsável também pela propagação do valor nulo aos atributos que, apesar de fazerem referência à entidades, não fazem parte da chave da relação à qual pertence.

5.3.6 Execução da SQL Dinâmica

Este módulo é responsável pela execução das transações de recuperações, bem como, a impressão dos resultados destas. Vale comentar que a linguagem SQL fornece um suporte bastante razoável à execução de transações dinâmicas de atualização (`Insert`, `Delete`, `Update`) que requisitam simples modificações no estado do banco de dados. Já as operações `Select`, pela sua imprevisibilidade, requisitam uma execução mais elaborada, pois não se sabe de antemão nem quantos nem quais atributos serão recuperados. Para a sua efetiva implementação, tornou-se necessário um estudo mais profundo da linguagem SQL, em especial, o uso de descritores[ORAC87B].

Capítulo 6

Considerações Finais

6.1 Introdução

Parte da comunidade acadêmica de banco de dados partilha a idéia da qual os modelos semânticos foram uma tentativa inglória de substituição do modelo relacional, e que a possibilidade atual de supremacia encontra-se nos modelos orientados a objetos.

Outros apostam em um desenvolvimento mais suave e gradativo, ressaltam as virtudes do modelo relacional, criticam as implementações atuais por não suportarem por completo as características do modelo e propõem extensões para o mesmo[STON90]. Este trabalho pode ser enquadrado neste grupo. A motivação para o desenvolvimento desta dissertação, baseou-se no propalado princípio de sedimentação e conhecimento da tecnologia de SGBDs relacionais, princípio este, que também está sendo seguido por alguns pesquisadores, no intuito de oferecer uma interface orientada a objetos, a partir de um SGBD relacional[PREM90, ROWE86].

Inúmeras extensões ao modelo e a SGBDs relacionais[CODD79, ZANI83, STON84, ROW87, PIST86, CAMM89, VOSS88, KUNG90] foram propostas e, na sua maioria, com abordagens diversas. A abordagem seguida por este trabalho, ou seja, a de prover um modelo semântico e uma linguagem de manipulação associada, transformando todas as transações requisitadas em transações de um SGBD relacional, não é inédita[ZANI83, NOGU89, KUNG90]. Contudo, se comparado ao GEM, a interface S-SQL permite algumas características não suportadas pelo mesmo, como a flexibilização do mecanismo de especialização, herança múltipla e derivação de classes com base em predicados.

6.2 Sugestões de Trabalhos Futuros e Extensões

O comportamento imposto na manutenção da integridade semântica da abstração de agregação foi extremamente simples: bloqueia-se as inclusões de instâncias na classe composta caso inexista alguma instância componente e propaga-se as exclusões de instâncias da classe componente para as classes compostas. Baseado nas propostas de Date[DATE81] e Casanova[CASA88b], o modelo poderia ser estendido a fim de permitir o bloqueio de exclusões e a propagação de inclusões de instâncias nas classes componentes.

Uma característica bastante interessante da interface S-SQL é a sua adequada representação gráfica. Ao mesmo tempo, apesar de fornecer uma linguagem de consulta mais acessível que a SQL para usuários inexperientes, um ambiente gráfico ainda é muito atrativo. Desenvolver um ambiente em que, a partir da representação gráfica S-SQL, seja possível mapear a definição de esquemas e recuperação de informações em comandos S-SQL, pode se tornar um excelente tópico de investigação.

O trabalho proposto, não aborda, em momento algum, a integração da ferramenta com uma linguagem de propósito geral. Este não foi um dos seus objetivos. Este módulo poderia ser desenvolvido sem maiores problemas como um pré-compilador, pois, toda a parte de tradução de transações e manutenção de integridade semântica, já se encontria na interface.

Por se tratar de um modelo que já oferece rede de generalização e *surrogates*, a interface S-SQL pode candidatar-se a servir de base de dados numa implementação de persistência em uma linguagem de programação orientada a objetos. Nessa direção, um módulo capaz de gerar, a partir das informações contidas no dicionário de dados, as declarações de tipos que devem ser importados pelos programas aplicativos e um conjunto de rotinas, do tipo um registro por vez, que permitem o acesso e a atualização dos dados, devem ser desenvolvidos.

Ainda na área de linguagens, um projeto mais ambicioso, seria a proposição de uma linguagem de programação de banco de dados baseada no modelo.

Mesmo sendo voltada aos SGBDs SQL, a interface S-SQL poderia ser perfeitamente implementada em outros SGBDs que não provessem tal linguagem. Sendo assim, o modelo e a linguagem poderiam ser empregados para o desenvolvimento de uma interface única em um ambiente heterogêneo.

Não houve preocupações na formalização e validação do processo de tradução de consultas. Um estudo nesse sentido seria extremamente válido. Este é também um comentário adequado para a validação de esquemas.

6.3 Conclusões

O trabalho consegue mostrar que, com o suporte disponível em um SGBD relacional, é possível implementar facilmente uma interface capaz de capturar uma gama maior de semântica da aplicação, outrora, no modelo relacional, restrita a simples relacionamentos entre tabelas ou a restrições singelas.

Nenhuma das abstrações implementadas requisitou estruturas auxiliares ou mapeamentos mirabolantes. As extensões sintáticas feitas à SQL foram mínimas, e, de forma alguma, desfigurou essa linguagem. Através da notação de ponto, da transparência no acesso aos atributos de superclasse e dos operadores de conjunto, a interface S-SQL, sem grandes problemas, propicia a definição de consultas que em SQL seriam consideravelmente mais complexas.

Como já foi visto, inúmeras críticas e sugestões foram e estão sendo feitas à SQL[DATE84, STON90]. O comitê ANSI trabalha no aperfeiçoamento e padronização desta linguagem, e, sem dúvida alguma, qualquer desenvovimento nessa direção, pode beneficiar diretamente a interface S-SQL, seja tornando disponível uma linguagem de manipulação mais natural ou pela facilitação do processo de tradução.

Espera-se que os mecanismos oferecidos para modificação de esquemas, embora simples, sejam satisfatórios no acompanhamento da evolução das aplicações.

A abordagem de construção de uma interface semântica sobre um SGBD SQL, mostrou-se oportuna por alguns motivos:

- Facilidade e racionalidade na implementação. O aproveitamento do já existente, evitou a implementação a partir do nada.

- Disponibilidade de SGBDs SQL, o que possibilita a migração desta interface por diversas plataformas.

Contudo, não se pode negar que a arquitetura proposta também possui os seus inconvenientes. A convivência com aplicativos SQL, por exemplo, passa a ser extremamente difícil. Essas aplicações não reconhecem a semântica agora existente no banco de dados, e são fontes de atualizações desencontradas e errôneas.

O mecanismo de *surrogates* foi de fundamental importância no processo de implementação de abstrações. Contudo, traz consigo a virtude e, ao mesmo tempo, o defeito, de não possuir conteúdo semântico. Mesmo sem motivo para tal, fazer acesso a um banco de dados S-SQL a partir da linguagem SQL, pode parecer um tanto estranho a usuários inexperientes. Ademais, qualquer seleção de objetos compostos, feitas com base em atributos chaves de objetos componentes, deve, necessariamente, invocar uma junção.

Aspectos de eficiência, como o agrupamento[18] de tabelas, não foram abordados. Na verdade, este é um problema que deve ser particularizado, de acordo com as facilidades disponíveis por cada SGBD utilizado.

Nesse contexto, vale ressaltar, que a manutenção da integridade semântica é bastante custosa, embora deva sempre ser feita, quer pelo usuário, quer pelo SGBD. Portanto, senão mais eficiente, é muito mais cômodo e seguro que este processo seja feito pelo SGBD. Ao contrário de outros trabalhos [CAMM89, KUNG90], preferiu-se por não retardar as propagações das atualizações, pois, sendo a interface S-SQL um ambiente semântico, é quase que inadmissível supor que sejam efetuadas consultas contra um banco de dados não confiável.

Ao final do trabalho, vale salientar que as ausências de duas extensões foram particulamente sentidas: TAD e semântica temporal. Existe farta literatura a

---

18. *Clustering*.

respeito de ambas como extensões ao modelo relacional[STON86, LUMM84]. Porém, devido a arquitetura proposta, tornaram-se impraticáveis as suas implementações.

A tendência atual da maioria das extensões feitas a SGBDs relacionais é o de oferecer suporte adequado às aplicações ditas não convencionais. A interface S-SQL é uma proposta mais singela. Ela está restrita à captação semântica das aplicações tradicionais. Isto não configura demérito algum para esta proposta, pois a mesma se apresenta como uma alternativa de baixo custo e rápido desenvolvimento para inúmeros usuários ávidos por obterem maiores facilidades de um SGBD.

Apêndice A

Aplicação Exemplo

Uma hipotética universidade, deseja implantar no seu sistema de computação, sistemas de informações para o registro acadêmico, de pessoal e controle de publicações disponíveis. Nesta universidade, em torno de quarenta cursos são oferecidos, entre graduação e pós-graduação, nas mais diversas áreas. O quadro funcional tem em torno de mil funcionários. O corpo docente é composto por dois mil professores e o corpo discente por cerca de dez mil estudantes.

Devido ao grande volume de informações armazenadas, o registro manual das informações de funcionários e alunos tornou-se extremamente ineficiente e propenso a erros.

O setor de pessoal arquiva, separadamente, as fichas dos funcionários técnico-administrativos e dos professores. Ambas, contêm dados pessoais, como nome, R.G. e endereço, bem como, dados funcionais, como número de matrícula, cargo e sálario atual.

Atendendo às particularidades de cada grupo, para os funcionários técnico-administrativos, a ficha detalha ainda o valor de vantagens incorporadas ao salário e o órgão de lotação. Já para os professores, são arquivados a sua titulação e o departamento de lotação. Nada impede que um funcionário ocupe um cargo técnico-admnistrativo e lecione em um departamento.

A universidade utiliza-se do expediente de contratação de mão-de-obra temporária, destinada a efetuar serviços especializados. Os prestadores de serviços, entretanto, não são qualificados como técnicos-administrativos, e nem podem receber qualquer tipo de vantagem.

O setor acadêmico, da mesma forma, arquiva dados relativos aos alunos. Além dos dados pessoais, outros, como número do registro acadêmico, curso, média geral e esportes que praticam, são também mantidos. A universidade mantém um sistema de distribuição de bolsas de estudos que é controlado por este setor. O critério de distribuição é bastante simples: tem direito a bolsa qualquer aluno que mantiver média geral maior que nove e meio. O setor acadêmico, junto ao setor de pessoal, também controla o registro dos alunos monitores.

Anualmente, o setor acadêmico produz um catálogo contendo informações sobre as pesquisas atualmente desenvolvidas pela universidade. A universidade tenta estimular o desenvolvimento de pesquisas oferecendo um adicional aos professores pesquisadores.

Cada unidade de ensino é subdividida em departamentos afins. Por exemplo, o instituto de matemática agrupa os departamentos de estatística, matemática e ciência da computação. Os departamentos possuem a responsabilidade direta pela administração dos cursos e disciplinas oferecidas.

A matrícula dos alunos nas disciplinas oferecidas pelos departamentos é feita semestralmente. O setor de matrícula prepara uma lista, dirigida a cada curso, contendo a disciplina, o número da turma e a quantidade de vagas disponíveis para o curso referido.

A biblioteca dispõe de um grande número de publicações para os professores, alunos e funcionários. Basicamente, as publicações são classificadas como livros ou periódicos e agrupadas por área. Há um grande interesse em ter-se um maior controle no empréstimo das publicações, bem como, em um sistema que proporcione um fácil acesso às informações bibliográficas.

Apêndice B

Sintaxe da LDD S-SQL

```
<comando_esquema> ::=
      <decl_tabela>
    | <decl_subclasse>

<decl_tabela> ::=
      <decl_criação>
    | <decl_exclusão>
    | <decl_alteração>

<decl_criação> ::=
      CREATE CLASS <ident> ( <lista_elem_tabela> ) [ <decl_chaves> ]

<decl_chaves> ::=
      KEY ( <lista_ident> ) { , KEY ( <lista_ident> ) }

<decl_exclusão> ::=
      DROP CLASS <ident>

<decl_alteração> ::=
      ALTER CLASS <ident> <ação_alteração>

<ação_alteração> ::=
      ADD ( <elem_tabela> )
    | DROP ( <ident> )
    | ADD KEY ( <lista_ident> )
    | DROP KEY ( <lista_ident> )

<decl_subclasse> ::=
      <decl_categoria>
    | <decl_derivação>
    | <decl_inclusão_cat>

<decl_categoria> ::=
      <tipo_pertinência> SUBCLASS OF <lista_ident> IS <ident>
    | <tipo_generalização> SUBCLASSES OF <lista_ident>
                          ARE <lista_ident>
```

```
<decl_derivação> ::=
     DERIVED SUBCLASS OF <ident> IS <ident> WHERE <tipo_derivação>

<tipo_derivação> ::=
       IS A VALUE OF <ident> FROM <ident>
     | <predicado>

<decl_inclusão_cat> ::=
       INCLUDE <ident> AS <lista_ident> SUBCLASS

<tipo_generalização> :=
       COVERING
     | OVERLAPPING
     | DISJOINT
     | PARTITIONING

<tipo_pertinência> ::=
       TOTAL
     | PARTIAL

<lista_elem_tabela> ::=
       <elem_tabela> { , <elem_tabela> }

<elem_tabela> ::=
       <ident> <domínio>

<domínio> :: =
       <tipo_base> [ NOT NULL ]
     | { <tipo_base> }
     | <ident>

<tipo_base> ::=
       CHAR [ (<cint>) ]
     | INT
     | FLOAT

<lista_ident> ::=
       <ident> { , <ident> }

<ident> ::=
       <letra> { <ialfanum> }

<alfanum> ::=
       <letra>
     | <dígito>
     | <símbolos>

<ialfanum> ::=
       <letra>
     | <dígito>
     | <isímbolos>
```

```
<letra> ::=
      A .. Z
    | a .. z

<dígito> ::= 1 .. 9

<símbolos> ::=
        ../
    |   :..@

<isímbolos> ::=

    | _
    | $
    | #
```

Apêndice C

Sintaxe da LMD S-SQL

```
<transação_ssql> ::=
      <transação_consulta>
    | <transação_inclusão>
    | <transação_exclusão>
    | <transação_alteraçao>

<transação_consulta> ::=
      SELECT <lista_ident_est> FROM <lista_var>
             [ WHERE <predicado> ]

<transação_inclusão> ::=
      INSERT INTO <ident> ( lista_ident )  VALUES ( <lista_val> )
                  [ SURROGATE FROM <ident> WHERE <predicado> ]

<transação_exclusão> ::=
      DELETE FROM <ident> [ WHERE <predicado> ]

<transação_alteraçao> ::=
      UPDATE <ident> SET ( <lista_alt> ) [ WHERE <predicado> ]

<predicado> ::=
      <predicado> OR <predicado_and>
    | <predicado_and>

<predicado_and> ::=
      <clausula>
    | <predicado_and> AND <clausula>

<clausula> ::=
      <expressao>
    | NOT <expressao>

<expressao> ::=
      <comparacao>
    | ( <predicado> )
<comparacao> ::=
          <operando> <op_rel> <operando>
    | <constante> IN <conjunto>
```

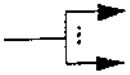
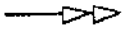
```
    | <constante> IN <ident_est>
    | <ident_est> <op_ident> <ident>

<operando> ::=
      <constante>
    | <ident_est>
    | <conjunto>
    | <funcao> ( <ident_est> )

<conjunto> ::=
      { <ident_est> [ GROUP BY <ident_est> ] [ WHERE <predicado> ] }
    | <cconjunto>

<op_rel> ::=
      =
    | >
    | <
    | >=
    | <=
    | !=

<op_ident> ::=
      IS-A
    | IS-NOT-A

<função> ::=
      MIN
    | MAX
    | AVG
    | COUNT
    | SUM

<lista_var> ::=
      <variável> {, <variável> }

<variável> ::=
      <ident> [<ident>]

<lista_val> ::=
      <valor> {, <valor> }

<valor> ::=
      <constante>
    | <cconjunto>
    | <predicado>
    | NULL

<lista_alt> ::=
      <elem_alt> {, <elem_alt> }

<elem_alt> ::=
```

```
      <ident> = <constante>
    | <ident> = <predicado>
    | <ident> = <sinal> <cconjunto>

<lista_ident_est> ::=
      <ident_est> {, <ident_est> }

<ident_est> ::=
      <ident> { . <ident> }

<lista_ident> ::=
      <ident> { , <ident> }

<ident> ::=
      <letra> { <ialfanum> }

<cconjunto> ::=
      { <constante> { , <constante> } }

<constante> ::=
      <cfloat>
    | <cint>
    | <cstring>

<cfloat> ::=
    [ <sinal> ] { <dígito>} . { <dígito> }

<cint> ::= {<dígito>}

<cstring> ::= '{<alfanum>}'

<alfanum> ::=
      <letra>
    | <dígito>
    | <símbolos>

<ialfanum> ::=
      <letra>
    | <dígito>
    | <isímbolos>

<letra> ::=
      A .. Z
    | a .. z

<dígito> ::= 1 .. 9

<símbolos> ::=
      ../
    | :..@
```

```
<isímbolos> ::=
      _
    | $
    | #

<sinal> ::=
      +
    | -
```

Apêndice D

Representação Gráfica

Simbolo	Significado
	Entidade
	Dominio Simples
	Referencia (Atributo chave)
	Referencia (Atributo chave composto / Agregacao)
	Referencia
	Referencia (Atributo multi-valorado)
	Heranca de atributos

Bibliografia

[AGRA89] Agrawal, R.; Gehani, N. H. ODE: the language and the data model. *Proceedings of the 1989 ACM SIGMOD Conference 18(2)*, Junho 1989.

[BANC88] Bancilhon, François. Object-oriented databases systems. *Rapport Technique Altair 16-88*, Janeiro 1988.

[BARN87] Barnerjee, J. et. al. Data model issues for object-oriented applications. *ACM Transactions on Office Information Systems 5(1)*, 1987, pp. 3-26.

[BLAH88] Blaha, M. R.; Premerlani, W. J.; Rumbaugh, J. E. Relational database design using an object-oriented methodology. *Communications of the ACM (31)4*, Abril 1988, pp. 414-427.

[BOOC86] Booch, Grady. Object-oriented development. *IEEE Transactions on Software Enginering 12(2)*. Fevereiro 1986, pp. 211-221.

[BORG83] Borgida, A. Features of languages for the development of information systems at the conceptual level. *Technical Report LCSR-TR-52, Laboratory of Computer Science Research, Rutgers University*, Dezembro 1983.

[BROD81a] Brodie, M. L. Data abstraction for design database-intensive applications. *Proceedings Workshop on Data Abstraction, Databases and Conceptual Modelling, SIGPLAN Notices 16(1)*, Janeiro 1981, pp. 101-103.

[BROD81b] Brodie, M. L. On modelling behavioural semantics of databases. *Proceedings of the 7th International Conference on Very Large Databases*, Setembro 1981, pp. 32-42.

[BROD83] Brodie, M. L. Association: a database abstraction for semantic modelling. *Entity-Relationship Approach to Information Modelling and Analysis*, P. Chen, Elsevier Science Publishers, 1983, pp. 577-602.

[BROD84] Brodie, M. L. On the development of data models. *Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming Languages. M. L. Brodie, J. Mylopoulos and J. W. Schmidt*, Eds. Springer Verlag, 1984, pp. 19-48.

[CAMM89] Cammarata, S.; Ramachandra, P.; Shane, D. Extending relational database with deffered referential integrity checking and intelligent joins. *Proceedings of the ACM SIGMOD Conference*, 1989, pp. 88-97.

[CASA88a] Casanova, M. A. et. al. A monitor enforcing referential integrity. *Anais do III Simposio Brasileiro de Banco de Dados*, 1-16, 1988.

[CASA88b] Casanova, M. A. et. al. Enforcing inclusion dependencies and referential integrity, *Technical Report CCR052*, Rio Scientific Center, Rio de Janeiro, Fevereiro de 1988.

[CHEN76] Chen, P. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems 1(1)*, Março 1976, pp. 9-36.

[CODD70] Codd, E. F. A relational model of data for large shared data banks. *Communications of the Acm 13(4)*, Junho 1970, pp. 337-387.

[CODD79] Codd, E. F. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems 4(4)*, Dezembro 1979, pp. 397-494.

[COPE84] Copeland, G.; Maier, D. Making smalltalk a database system. *Proceedings of the ACM SIGMOD Conference*, Junho 1984, pp. 316-325.

[DATE81] Date, Chris. Referential integrity. *Proceedings of the 7th International Conference on Very Large Databases*, Cannes, France, Setembro 1981, pp. 2-12.

[DATE84] Date, Chris. A critique of the SQL database language. *ACM SIGMOD Record 14(3)*, Novembro, 1984.

[DATE89] Date, Chris. *Guia para o padrão SQL*. Editora Campus, 1989.

[DITT86] Dittrich, Klaus R. Object-oriented systems: a workshop report. *Proceedings of the 5th. E-R Conference*, 1986.

[HAMM81] Hammer, M.; McLeod, D. Database description with SDM: a semantic database model. *ACM Transactions on Database Systems 6(3)*, Setembro 1981, pp. 351-386.

[HULL87] Hull, R.; King, R. Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys 19(3)*, Setembro 1987, pp. 201-260.

[IBM88] *SQL/DS Versão 2 Release 2, Guia de Utilização de Integridade Referencial, IBM Brasil*, 1988.

[JAGA88] Jagannatham, D. et. al. SIM: a databse system based on the semantic data model, *Proceedings of the ACM SIGMOD Conference*, Junho 1988, pp. 46-55.

[JOHN79] Johnson, S. C. *YACC – yet another compiler-compiler, Technical Report*, Murray Hill, AT&T Bell Laboratories, 1979.

[KENT79] Kent, W. Limitations of record based information models. *ACM Transactions on Database Systems 4(1)*, Março 1979, pp. 107-131.

[KENT82] Kent, W. A simple guide to five normal forms in relational database theory. *Communications of the ACM 26(2)*, Fevereiro 1982, pp. 120-125.

[KERN78] Kernighan, Brian W.; Ritchie, D. W. *C a linguagem de programação*. Editora Campus, 1988.

[KUNG90] Kung, Chenko. Object subclass hierarchy in SQL: a simple approach. *Communications of the ACM (33)7*, Julho 1990, pp. 117-225.

[LUM84] Lum, V.; Dadam, P. et. al. Designing DBMS support for the temporal dimension. *In Proceedings of the ACM SIGMOD Conference*, Junho 1984, pp. 115-130.

[MACH90] Machado, Javam de Castro. *Um modelo de dados para ambientes de projeto. Dissertação de Mestrado em Ciência da Computação*, UFRGS-CPGCC, Porto Alegre, 1990.

[MICH76] Michaels, Ann S.; Mittman, B.; Carlson, C. R. A comparison of the relational and CODASYL approaches to database management. *Computing Surveys, 8(1)*, Março 1976.

[MYLO80] Mylopoulos, J.; Bernstein, P. A.; Wong, H. K. A language facility for design database-intensive applications. *ACM Transactions on Database Systems 5(2)*, Junho 1980, pp. 185-207.

[NOGU89] Nogueira, Mônica de Lima. *Um sistema para manuseio de objetos entidade-relacionamento no modelo relacional. Dissertação de Mestrado em Ciência da Computação*, DCC-IMECC-UNICAMP, Campinas, 1989.

[OLIV89] Oliveira, Ricardo. *Conceitos Básicos de Programação e Bancos de Dados Orientados a Objetos, Estudo Dirigido em Banco de Dados*, DCC-IMECC-UNICAMP, Campinas, 1989.

[ORAC87A] *SQL Plus User's Guide, Ver. 2.0, Oracle Corporation*, Belmont, California, 1987.

[ORAC87B] *PRO*C Users's Guide, Ver. 1.1*, Oracle Corporation, Belmont, California, 1987.

[PECK88] Peckham, J.; Maryanski, F. Semantic data models. *ACM Computing Surveys 20(3)*, Setembro 1988, pp. 153-189.

[PIST86] Pistor, P.; Anderson, F. Designing a generalized NF2 model with an SQL-type language interface. *Proceedings of the 12th International Conference on Very Large Databases*, Agosto 1986, pp. 278-285.

[PREM90] Premerlani, W. J.; Blaha, M. et. al. An object-oriented relational database. *Communications of the ACM 33(11)*, Novembro 1990, pp. 99-109.

[RENT82] Rentsch, Tim. Object-oriented programming. *SIGPLAN Notices*, Setembro 1982, pp. 51-57.

[ROWE86] Rowe, L. A shared object hierarchy. *IEEE Proceedings of the International Object-Oriented WorkShop*, California, 1986, pp. 160-170.

[ROWE87] Rowe L.; Stonebraker, M. The POSTGRES data model. *Readings in Object-Oriented Database Systems, Cap. 7.2, Stanley B. Zdonik and David Maier, Morgan Kaufmann Publishers*, San Mateo, California, pp. 461-473.

[ROGE87] Rogers, T. R.; Cattel, R. G. G. Entity-Relationship database user interfaces. *Readings in Database Systems, Cap. 5.4, M. Stonebraker, Morgan Kaufmann Publishers*, San Mateo, California, pp. 359-373.

[SCHM77] Schmidt, J. W. Some high level construct for data of type relation. *ACM Transactions on Databases Systems 2(3)*, Setembro 1977, pp. 247-281.

[SHIP81] Shipman, D. The functional data model and the data language DAPLEX. *ACM Transactions on Database Systems 6(1)*, Março 1981, pp. 140-173.

[SMIT77a] Smith, J. M.; Smith, D. C. P. Database abstractions: aggregation, *Communications of the ACM 20(6)*, Junho 1977, pp. 405-413.

[SMIT77b] Smith, J. M.; Smith, D. C. P. Database abstractions: aggregation and generalization, *ACM Transactions on Database Systems (2)2*, Junho 1977, pp. 105-133.

[SNOD85] Snodgrass, R.; Ahn., I. A taxonomy of time in databases, *Proceedings of the ACM SIGMOD Conference*, 1985, pp. 236-246.

[STEF84] Stefik, Mark; Bobrow, Daniel. Object-oriented Programming: themes and variations, *The AI Magazine, Vol 6(4)*, pp. 40-60.

[STON83] Stonebraker, M.; Rubenstein, B.; Guttman, Antonin. Application of abstract data types and abstract indices to CAD data, *Proceedings of the Annual Meeting Database Week*, 1983, pp. 107-115.

[STON84] Stonebraker, M. QUEL as a data type. *Proceedings of the ACM SIGMOD Conference*, Junho 1984.

[STON86] Stonebraker, Michael. Inclusion of new types in relational data bases systems. *Proceedings of the Second International Conference on Data Base Engineering pp. 262-269*, Los Angeles, California, 1986.

[STON88a] Stonebraker, M. The Roots (Introduction). *Readings in Database Systems, M. Stonebraker, Morgan Kaufmann Publishers*, Abril 1988, pp. 1-4.

[STON88b] Stonebraker, M. New data models-Introduction. *Readings in Databases Systems, Cap 6.1, M. Stonebraker, Morgan Kaufmann Publishers*, Abril 1988, pp. 369-373.

[STON90] Stonebraker, M., et. al. Third-generation data base system manifesto, *ACM SIGMOD Record 19(3)*, Setembro 1990, pp. 31-44.

[TAYL76] Taylor, R. W.; Frank, R. L. CODASYL database management systems, *ACM Computing Surveys 8(1)*, Março 1976, pp. 67-103.

[TEOR86] Teorey, T. J.; Yang, D.; Fry, J. P. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys 18(2)*, Junho 1986, pp. 197-222.

[TSIC76] Tsichritzis, D.; Lochovsky, F. Hierarchical databases management systems: a Survey, *ACM Computing Surveys 8(1)*, Março 1976, pp. 105-123.

[TSUR84] Tsur, Shalon; Zaniolo, Carlo. An implementation of GEM – supporting a semantic data model on a relational back-end. *Proceedings of the ACM SIGMOD Conference 14(2)*, 1984, pp. 286-295.

[VELO86] Veloso, Paulo. *Tipos abstratos de dados*. V Escola de Computação, Belo Horizonte, Julho 1986.

[VIES89] Viescas, John. *SQL: a linguagem padrão de banco de dados relacionais! Quick Reference*. Editora Campus, 1989.

[VOSS88] Vossen, G.; Yacabucci, Jim. An extension of the database language SQL to capture more relational concepts. *ACM SIGMOD Record 17(4)*, Dezembro 1988, pp. 70-78.

[WAGN87] Wagner, C. F. e Medeiros, C. B. Um modelo binário estendido para entidade-relacionamento. *Anais do VII Congresso da SBC*, Salvador, 1987.

[WAGN88] Wagner, C. F. Implementing abstraction hierarchies, *Proceedings of the 7th International Conference on Entity-Relationship Approach*, Roma, 1988.

[ZANI83] Zaniolo, C. The database language GEM. *Readings in object-oriented database systems, Cap 7.1, Stanley B. Zdonik and David Maier, Morgan Kaufmann Publishers*, pp. 449-460.

[ZDON90] Zdonik, Stanley B.; Maier, David. Fundamental of object-oriented databases. *Readings in object-oriented database systems, Stanley B. Zdonik and David Maier, Morgan Kaufmann Publishers*, Janeiro 1990, pp. 1-32.