Atualização de Bancos de Dados Objeto Relacionais através de Visões de Objetos

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Juliana Pontes da Costa e aprovada pela Banca Examinadora.

Fortaleza, 19 de Dezembro de 2002.

Profa. Dra. Vânia Maria Ponte Vidal
(Orientadora)

Dissertação apresentada ao Mestrado em Ciência da Computação, UFC, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Mestrado em Ciência da Computação
Universidade Federal do Ceará

Atualização de Bancos de Dados Objeto Relacionais através de Visões de Objetos

Juliana Pontes da Costa[1]

19 de Dezembro de 2002

Banca Examinadora:

- Profa. Dra. Vânia Maria Ponte Vidal
- Prof. Dr. Marco Antônio Casanova
- Prof. Dr. Angelo Roncalli Alencar Brayner

[1]**Bolsista da FUNCAP**

Resumo

Na arquitetura de três níveis de esquema, as visões constituem interfaces através das quais os usuários acessam e atualizam o banco de dados. Os Bancos de Dados Objeto-Relacional estendem o mecanismo básico de visões relacionais e proporcionam visões de objetos. As visões de objeto tornam mais fácil introduzir aplicações orientadas a objetos para dados relacionais já existentes sem ter que fazer mudanças drásticas de um paradigma para o outro. Essas visões provêem ainda a flexibilidade de ver o mesmo dado (relacional ou orientado a objetos) de várias maneiras.

Uma atualização de visão é simplesmente uma atualização que é especificada em uma visão, mas que deve ser traduzida em uma seqüência de atualizações no banco de dados. O problema de Tradução de Atualização de Visão (TAV) refere-se à questão de definir traduções corretas de atualizações de visões. Neste trabalho desenvolvemos algoritmos para gerar tradutores para as operações básicas de atualização de visões de objetos. Um tradutor é uma função que recebe como entrada um pedido de atualização de visão e gera a tradução para esta atualização. No nosso enfoque, os tradutores são definidos em tempo de projeto e armazenados juntamente com as definições do esquema da visão de objetos. Assim, uma vez definido o tradutor, o usuário especifica atualizações através da visão e o tradutor as traduz em atualizações no banco de dados, sem a necessidade de qualquer diálogo adicional.

Nos algoritmos propostos, os tradutores são gerados a partir das assertivas de correspondências da visão de objetos, as quais especificam formalmente o relacionamento do esquema da visão e o esquema do banco de dados, o qual pode ser um esquema relacional ou objeto-relacional. As vantagens do uso desse formalismo é que nos permite especificar as condições em que é possível definir um tradutor em tempo de projeto, assim como calcular a seqüência de atualizações necessárias no banco de dados para realizar uma atualização solicitada na visão.

Agradecimentos

De formas variadas e em momentos diferentes, muitas pessoas direta ou indiretamente, ajudaram no desenvolvimento deste trabalho. A cada uma delas, sinceramente, agradeço.

Agradeço a Deus, por toda a força que Ele me deu para seguir nesta caminhada.

Agradeço aos meus pais Eduardo e Jane, presenças tão importantes também na minha formação profissional.

Agradeço ao meu marido Júnior pelo amor, paciência e incentivos dedicados a mim.

Agradeço aos meus irmãos Felipe e Flávio, por terem aguentado o meu mau humor nos momentos difíceis e por compartilharem dos momentos de felicidade.

Agradeço a todos meus parentes pelo apoio que me deram.

Agradeço a minha orientadora Vânia por toda a paciência e orientação tão importante no desenvolvimento deste trabalho.

Agradeço aos meus amigos presenças essenciais no cotidiano de minha atividade. Em especial aos amigos Wamberg, Renata, Danielle, Fabiana, Lineu, Valdiana e Sabrina.

Agradeço aos professores e funcionários que contribuiram de alguma forma nesta dissertação. Em especial aos professores que participaram da banca.

Agradeço ao Departamento de Computação da UFC e a Funcap por tornarem possível a realização deste mestrado.

Sumário

Lista de Tabelas

Lista de Figuras

1

Introdução

Os Bancos de Dados Objeto-Relacionais (BDOR) surgiram com o intuito de atender as necessidades das aplicações atuais tais como manipular e modelar tipos de dados complexos e suportar dados de grandes tamanhos. Esses bancos de dados estendem o modelo relacional para suportar o modelo de objetos, expandindo o espaço de alternativas disponíveis para o projeto do esquema. Nos BDOR os dados existentes no formato relacional podem continuar existindo sem a necessidade de alterações. Os BDOR suportam extensão dos tipos de dados, objetos complexos, herança, regras e têm um poderoso mecanismo de visões de objetos.

Na arquitetura de três-níveis de esquema, as visões ou esquemas externos permitem ao usuário ignorar os dados que não são relevantes para a sua aplicação. As visões além de protegerem o acesso aos dados, ajudam a alcançar um certo grau de independência lógica, uma vez que é possível alterar o esquema do banco de dados sem alterar uma visão. Os usuários interagem com uma visão para consultar e atualizar o banco de dados.

Assim como uma visão relacional é uma tabela virtual, um visão de objetos é uma tabela virtual de objetos. Além das vantagens bastante difundidas das visões, as visões de objetos proporcionam novas vantagens como: (i) permitir ao usuário utilizar as técnicas de programação orientada a objetos sem ter que converter as tabelas existentes para esta tecnologia; (ii) converter os dados gradualmente de tabelas relacionais para tabelas objeto-relacionais; (iii) fornecer a flexibilidade para acessar o mesmo dado relacional ou objeto de mais de uma forma, ou seja, você pode usar diferentes representações de objetos em memória para diferentes aplicações sem mudar a maneira que o dado está armazenado no banco de dados.

Como as visões de objetos são apenas interfaces, consultas e atualizações especificadas nas visões devem ser traduzidas em consultas e atualizações a serem executadas no banco de dados. Uma definição de visão consiste de: (i) o mapeador de instâncias que especifica como o estado do banco de dados em um determinado instante é mapeado no estado da visão correspondente; (ii) o mapeador de atualizações que especifica como atualizações

especificadas na visão são traduzidas em uma seqüência de atualizações especificadas no banco de dados.

A atualização de Banco de Dados Objeto-Relacionais através de visões de objetos é um problema que ainda não foi tratado na literatura. A maioria das pesquisas têm focalizado no problema de atualização de bancos de dados relacionais através de visões. Apesar do grande número de pesquisas que tratam a atualização de bancos de dados relacionais através de visões relacionais [6, 15, 17, 20], este problema ainda não foi resolvido satisfatoriamente. Isto acontece porque a maioria dos enfoques propostos não usa um formalismo para representar os relacionamentos entre o esquema da visão e o esquema do banco de dados. Com essa falta de formalismo não é possível garantir que as traduções geradas pelos algoritmos propostos nesses enfoques são corretas.

Neste trabalho, tratamos o problema da atualização de bancos de dados objeto-relacionais através de visões de objetos. O esquema do BDOR pode ser um esquema relacional ou objeto-relacional. O problema de atualização de BDOR através de visões de objetos é bem mais complexo, pois, diferentemente do modelo relacional, o modelo objeto-relacional permite o desenvolvimento de estruturas complexas, com atributos multivalorados, estruturados e de referência, além das estruturas relacionais.

No nosso enfoque, o mapeador de atualizações de uma visão consiste de um conjunto de tradutores, um para cada uma das operações de atualização de visão que forem permitidas. Um tradutor é uma função que recebe como entrada um pedido de atualização e gera a tradução para esta atualização. Os tradutores são definidos em tempo de projeto. Uma vez definido o tradutor, o usuário especifica atualizações através da visão e o tradutor as traduz em atualizações no banco de dados, sem a necessidade de qualquer diálogo adicional.

Neste trabalho, desenvolvemos algoritmos que geram tradutores para os tipos básicos de operações de atualização de visões de objetos: inserção e remoção de objetos, modificação dos valores de atributos monovalorados e inserção e remoção de objetos em coleções aninhadas. Os algoritmos recebem como entrada o esquema da visão de objetos, o esquema do banco de dados e as assertivas de correspondências (ACs) do esquema da visão com o esquema do banco de dados. As ACs da visão especificam formalmente o relacionamento entre o esquema da visão e o esquema do banco de dados. Nosso formalismo permite identificar precisamente as situações em que as ambigüidades podem ser resolvidas em tempo de definição da visão e calcular a seqüência de atualizações necessárias no banco de dados para realizar uma atualização solicitada na visão. Nesta dissertação, só tratamos de visões perservadoras de objetos. Uma visão preserva os objetos, quando cada objeto da visão é semanticamente equivalente a um objeto de uma tabela base [1].

[1]Dois objetos $\mathbf{o}_1$ e $\mathbf{o}_2$ são semanticamente equivalentes ($\mathbf{o}_1 \equiv \mathbf{o}_2$) sss representam uma mesma entidade do mundo real.

Uma das contribuições deste trabalho é a definição dos vários tipos de Assertivas de Correspondência (AC) para especificar correspondênias entre esquemas objeto-relacionais. O formalismo proposto permite especificar várias formas de correspondências, inclusive situações onde os esquemas são estruturados de formas diferentes (heterogeneidade estrutural) . A vantagem deste formalismo é que permite especificar de forma precisa as correspondências do esquema da visão com o esquema do banco de dados. Discutimos também os passos do processo para geração das assertivas de correspondências de uma visão.

1.1. Trabalhos relacionados

O problema de definir traduções para atualizações de visões tem sido considerado por muitos pesquisadores, incluindo [3, 6, 8, 22, 15]. O enfoque de definir procedimentos gerais de tradução [3], [6] somente pode ser aplicado a uma classe restrita de atualizações de visões, porque alguns tipos de atualizações de visões requerem que mais semântica seja fornecida pelo usuário, para eliminar ambigüidades nas traduções das atualizações em visões [22], [15].

O enfoque de Keller propõe que a semântica necessária para remover ambigüidades na tradução das atualizações seja obtida através de diálogo com o usuário na hora da definição da visão. Nesse enfoque as visões são definidas pelo administrador do banco de dados(DBA) que fornece a semântica necessária para escolher um tradutor, a semântica é meramente a sequência de decisões feitas pelo DBA. As questões são apresentadas para o DBA, sendo fornecida assim uma enumeração completa de todas as traduções possíveis para as atualizações da visão, baseado no esquema da visão, no esquema do banco de dados e nas respostas de questões anteriores. Nesta seção, discutimos os principais enfoques que tratam da atualização de base de dados através de visões.

O projeto Penguim [11, 36] propõe um framework unificado para o desenvolvimento de sistemas que compartilham informações. Esse framework armazena os dados em bancos de dados relacionais distribuídos e os apresentam para a aplicação na forma de objetos através das visões de objetos. Visões de objetos projetadas sobre banco de dados relacionais permitem que cada aplicação tenha seu próprio esquema de objetos ao invés de requerer que todas as aplicações compartilhem o mesmo esquema. Nesse projeto, as relações do banco de dados são mapeadas em templates de objetos através de um diálogo com o usuário, onde cada template pode ser uma combinação complexa de operações de junções e projeções das relações bases. Para garantir a correta definição dos templates a partir do esquema do banco de dados é usado o modelo estrutural proposto em Penguim [4]. Cada template é ancorado numa relação pivô e a chave desta relação serve como um identificador de objeto quando os objetos são instanciados. Os templates de objetos são arranjados em uma rede de objetos que coletivamente formam o esquema de objetos. Esse processo é guiado pelo conhecimento do usuário, através do uso da semântica da estrutura

do banco de dados. Uma vez criada a definição da visão de objetos, o sistema Penguim fornece módulos de instanciação e decomposição de objetos, para suportar as operações de recuperação e armazenamento de objetos. A instanciação é feita a partir dos dados das relações do banco de dados e a decomposição consiste em distribuir as atualizações sobre as relações bases. O Penguim gera e manipula instâncias de objetos temporários através da ligação das relações bases com os templates de objetos pré-definidos. Os usuários atualizam os objetos e finalmente o resultado deve ser colocado a disposição de outros usuários, traduzindo as atualizações para as tabelas do banco de dados relacional. O módulo de decomposição de objetos faz essa tarefa e mapeia as instancias de objetos de volta para o banco de dados. Esse módulo é chamado quando alguma mudança, feita nas instancias de objetos, deve ser tornadas persistentes no nível do banco de dados. Em tempo de geração do template, todas as possíveis ambigüidades são enumeradas e resolvidas. Quando operações de atualização são executadas nas instâncias, estas atualizações devem ser traduzidas e movidas da representação de objetos para as relações base do banco de dados. Em [4] é apresentado um esquema para tratar operações de atualização em objetos de visão. Atualizações realizadas sobre objetos da visão devem ser traduzidas em operações válidas nas tabelas base. A consistência do banco de dados, definida pelas regras de integridade do modelo estrutural, é mantida para prevenir ou corrigir atualizações ilegais.

Recentemente, o modelo XML (eXtended Markup Language) vem se tornando um padrão para troca e integração de dados na WEB. Isto cria a necessidade de publicar dados, os quais estão armazenados em bases convencionais, no formato XML. A publicação desses dados é feita através do uso de visões XML. Sendo assim, temos que XML está sendo usado como uma camada intermediária entre aplicações e banco de dados [54]. O enfoque propõe algoritmos que possibilitam a atualização de banco de dados relacional através de atualizações XQuery na visão XML. Além disso, apresenta algumas estratégias de tradução de atualizações XML em atualizações SQL, tais como: Triggers (Per-Tuple e Per-Stratement) e o método Access suppport relations (ASR), o qual é um método para otimizar a validação de expressões de caminho em banco de dados orientados a objetos e semi-estruturados. Por último, o enfoque faz uma análise da performance de diferentes estratégias de atualização.

O nosso enfoque se diferencia dos demais, pois utilizamos o modelo objeto-relacional. O problema de atualização de banco de dados objeto-relacional (BDOR) através de visões de objetos é bem mais complexo que o relacional, pois, além de estruturas relacionais, o modelo objeto-relacional permite o uso de tabela de objetos, os quais possuem estruturas complexas,como atributos multivalorados (coleções aninhadas), atributos estruturados e de referência.

A dissertação é dividida em 6 capítulos como se segue. No Capítulo 2, apresentamos o modelo objeto-relacional, que é utilizado para representar o esquema das visões de objetos e o esquema do banco de dados, o qual pode ser um esquema relacional ou objeto-relacional. No Capítulo 3, apresentamos as assertivas de correspondências que us-

amos para especificar os relacionamentos entre esquemas objeto-relacionais. No Capítulo 4, apresentamos o nosso enfoque para atualização de banco de dados objeto-relacional através de visões de objetos. No Capítulo 5 descrevemos os algoritmos que geram tradutores para as operações de: (i) Adição de um objeto em uma coleção aninhada da visão; (ii) Remoção de um objeto de uma coleção aninhada da visão; (iii) Modificação de atributos monovalorados de um objeto da visão; (iv) Adição de um objeto em uma visão; e (v) Remoção de um objeto de uma visão. Finalmente no Capítulo 6, são mostradas as conclusões, contribuições e sugestões para trabalhos futuros.

2

Modelo Objeto-Relacional

Neste capítulo apresentamos o modelo objeto-relacional baseado no modelo do Oracle 8i. O modelo objeto-relacional é resultado da extensão do modelo relacional para suportar os conceitos de orientação a objetos. Vale ressaltar que um esquema objeto-relacional pode ser puramente relacional ou objeto-relacional. Uma das vantagens do modelo objeto-relacional é o suporte a dados complexos e a visões de objetos.

Este capítulo é organizado como se segue. Na Seção 2.1, discutimos as limitações da representação relacional e apresentamos as características do modelo objeto-relacional. Nas Seções 2.2 e 2.3, apresentamos os conceitos básicos do modelo Objeto-Relacional. Na Seção 2.4, apresentamos a representação gráfica usada neste trabalho e na Seção 2.5, apresentamos o mecanismo das visões de objetos proposto no Oracle 8i.

2.1. Introdução

Apesar da popularidade do modelo relacional e da infraestrutura em termos de banco de dados comerciais que foram projetados para suportá-lo, o modelo relacional básico não é suficiente para modelar as exigências de aplicações como sistemas de informação geográficas, bibliotecas digitais, engenharia mecânica e elétrica (CAD/CAM), engenharia de software (ferramentas CASE), aplicações médicas e científicas (química, genética e geoprocessamento), as quais possuem tipos de dados muito mais complexos que os dados tradicionalmente tratados em banco de dados relacionais. O modelo relacional apresenta várias limitações como: (i) os atributos compostos não podem ser modelados diretamente no banco de dados, (ii) os atributos multivalorados devem ser diferenciados através de valores únicos e representados em uma outra estrutura relacional, (iii) as agregações e as especializações requerem esquemas de relações individuais equipadas com restrições de integridade especiais e (iv) é necessário a introdução de chaves artificiais se os atributos não forem suficientes para obter uma identificação única.

Com o surgimento das aplicações avançadas de banco de dados descobriu-se a necessidade de mecanismos para: definição e manipulação de tipos de dados complexos, facilidades para modelar o comportamento de objetos complexos através da definição de operações, suporte para itens de dados de grandes tamanho e gerenciamento de versões.

Para dar suporte a essa classe mais ampla de aplicações surgiram, no final dos anos 80, os banco de dados orientados a objetos e os bancos de dados objeto-relacionais. O modelo objeto-relacional pode ser visto como uma extensão do modelo relacional para suportar a capacidade de modelagem do modelo orientado a objetos. A extensão inclui mecanismos para permitir aos usuários estender o banco de dados com tipos e funções especificas da aplicação. O modelo Objeto-Relacional fornece suporte a consultas complexas sobre dados complexos e atende aos requisitos das novas gerações de aplicações de negócio. As principais características do modelo Objeto-Relacional são: (i) permite especificar e utilizar tipos abstratos de dados da mesma forma que os tipos de dados pré-definidos, (ii) estende a tabela convencional para permitir a referência de objetos, (iii) permite definir tipos abstratos de dados e valores alfanuméricos como domínio de coluna, (iv) utiliza referências para representar conexões inter-objetos tornando as consultas baseadas em caminhos de referência mais compactas do que as consultas feitas com junção, (v) implementa herança organizando todos os tipos em hierarquias e (vi) utiliza os construtores: set, list, multiset ou array para organizar coleções de objetos. Desta forma o modelo objeto-relacional aumenta indefinidamente o conjunto de tipos e funções fornecidas pelo SGBD e possibilita o desenvolvimento de aplicações de forma simplificada através do reuso de código e definição de padrões.

Vimos que atualmente muitas aplicações requerem técnicas de banco de dados para modelar e gerenciar objetos complexos e, ao mesmo tempo, suportar o compartilhamento de dados entre as aplicações. Porém, armazenar informações através de objetos inibe o compartilhamento, já que os objetos são configurados para uma determinada visão da aplicação e, portanto, não podem servir a uma variedade de propósitos [4]. Essa restrição aliada a familiarização dos profissionais com a tecnologia relacional tem dificultado a entrada dos banco de dados orientados a objetos no meio industrial. Com isso, apesar da explosão do modelo de objetos, a tendência é que o meio industrial continue a utilizar banco de dados relacional e suas extensões em suas aplicações, já que esses banco de dados atendem ao principal incentivo para explorar SGBDs, a capacidade de compartilhamento de dados.

Como podemos notar, toda essa realidade impulsionou o aparecimento, de um novo tipo de sistema, onde aplicações orientadas a objetos acessam bancos de dados relacionais. Entretanto, a coexistência entre as duas tecnologias gera, o que muitos autores chamam de "object-relational impedance-mismatch"[9, 2]. [9] declara que "A modelagem objeto descreve um sistema através de objetos e que cada objeto possui uma identidade, comportamento e estados encapsulados. Por outro lado, o modelo relacional descreve um sistema através de informações". Em [2] o autor cita que "O paradigma objeto é baseado em objetos que possuem dados e comportamentos enquanto que o paradigma relacional é baseado

em armazenagem de dados". Essa divergência, acontece porque o objetivo da modelagem do esquema relacional do banco de dados é a normalização dos dados, enquanto que o objetivo dos modelos objetos utilizados para o desenvolvimento das aplicações é modelar o domínio do problema como objetos do mundo real. Como veremos na Seção 2.5, uma forma de tratar a integração de aplicações orientadas a objetos com banco de dados é através do uso de visões de objetos.

Nos últimos anos a Oracle Corporation trabalhou na extensão do seu Sistema de Banco de Dados Relacional com o objetivo de transformá-lo em um Sistema de Banco de Dados Objeto-Relacional adicionando suporte a extensão de tipos, armazenamento de objetos, controle de cache de objetos, extensão de consultas, suporte a tipos de dados multimídia, visões de objetos, entre outras características. O modelo objeto-relacional utilizado neste trabalho é baseado no modelo do banco de dados Oracle 8i.

2.2. Objetos e Tabelas

O paradigma orientado a objetos é baseado no conceito de objetos. Os objetos representam entidades do mundo real e possuem um identificador único, chamado "Object-Identifier"(OID). Os OIDS são gerados pelo sistema e nunca mudam de valor mesmo quando o conteúdo do objeto é modificado. Os OIDs permitem que os objetos sejam referenciados.

Neste trabalho, assim como no modelo de objetos da ODMG-97 [44], é feita a distinção entre objetos e literais. Um literal é um tipo especial de objeto que não possui um identificador. Portanto, ao contrário dos objetos com indentificadores, os literais não podem ser referenciados por outros objetos.

Os objetos encapsulam estrutura e comportamento. A estrutura de um objeto é definida por um conjunto de atributos. Essa estrutura representa o estado interno do objeto e é formada por um conjunto de valores dos atributos. O comportamento de um objeto é definido por um conjunto de operações que podem ser executadas nos objetos. As operações são invocadas através do envio de mensagem.

Todo objeto tem um tipo que serve como molde para a criação das instâncias desse tipo. Todos os objetos de um tipo tem estrutura e comportamento comum. Considere, por exemplo, o esquema objeto-relacional COMPANHIA mostrado na Figura 2.1. EMPREGADO é um tipo que possui os atributos **eno**, **enome** e **esalario**.

Um objeto persistente é um objeto que é colecionado no banco de dados, diferentemente dos objetos transientes, os quais só existem localmente nos blocos PL/SQL. Objetos transientes, assim como variáveis locais PL/SQL, são desalocados quando termina o escopo. Por outro lado, objetos persistentes são colecionados em tabelas do banco de dados e ficam disponíveis para serem acessados, modificados e deletados. No modelo objeto-

```
CREATE TYPE EMPREGADO AS
OBJECT(
  eno        NUMBER,
  enome      VARCHAR2(20),
  esalario   NUMBER,
  dept_ref   REF DEPT,
  eendereco ENDERECO,
  efones       VARRAY(3) OF varchar(11)    );

CREATE TYPE ENDERECO AS
OBJECT (
  rua        VARCHAR2(20),
  cidade     VARCHAR2(10),
  estado     CHAR(2),
  cep        VARCHAR2(10)   );

CREATE TYPE EMPREGADO_LISTA
AS TABLE OF REF EMPREGADO;
```

```
CREATE TYPE DEPT AS
OBJECT (
  dno       NUMBER,
  dnome     VARCHAR2(20),
  emplist   EMPREGADO_LISTA   );

CREATE TABLE EMPREGADOS OF
EMPREGADO (
PRIMARY KEY(eno),
FOREING KEY(dept_ref) references DEPTS );

CREATE TABLE DEPTS OF DEPT(
PRIMARY KEY(dno),
NESTED TABLE emplist STORE AS
EMPLIST_NTAB);

ALTER TABLE EMPLIST_NTAB
ADD ( SCOPE FOR(emplist) is
EMPREGADOS);
```

Figura 2.1. *Esquema objeto-relacional* COMPANHIA

relacional existem dois tipos de tabelas: *tabelas de objetos* e *tabela de tuplas*, as quais são definidas a seguir.

- **Tabela de objetos:** é uma tabela que coleciona objetos com identificadores. Assim sendo, os objetos desta tabela podem ser referenciados. Toda tabela de objetos tem um tipo associado, que especifica o tipo dos seus objetos. No esquema objeto-relacional COMPANHIA apresentado na Figura 2.1, EMPREGADOS é uma tabela da tabela de objetos, que contém um conjunto de objetos do tipo EMPREGADO.

- **Tabela de tuplas:** é a tabela do Modelo Relacional. As tabelas de tuplas colecionam um conjunto de tuplas, que são objetos sem OIDS. Esses objetos não podem ser referenciados. Em uma única declaração define-se a estrutura das tuplas (tipo das tuplas da tabela) e cria-se a tabela que colecionará um conjunto de tuplas. Assim sendo, o tipo das tuplas de uma tabela relacional $\mathbf{R}_1$ é implicitamente definido na criação da tabela $\mathbf{R}_1$. Na Figura 2.2, mostramos a definição de uma tabela de tuplas EMPS_REL, que possui os atributos **empno**, **empnome** e **empsalario**.

2.3. Tipo Abstrato de Dados (TAD)

Uma das principais características do uso de objetos é que além do usuário poder usar os tipos pré-definidos pelo banco de dados eles podem também construir tipos próprios para sua aplicação. Esses tipos são chamados de Tipos Abstratos de Dados (TAD) que podem ser usados da mesma forma que são usados os tipos pré-definidos.

```
CREATE TABLE EMPS_REL
(
  empno    NUMBER PRIMARY KEY,
  empnome  VARCHAR2(20),
  empsalario   NUMBER
);
```

Figura 2.2. *Tabela de Tuplas*

Um tipo abstrato de dado define a estrutura de dados (atributos) e as operações (métodos), funciona como um molde para a criação de objetos através da atribuição de valores a essa estrutura de dados. Em alguns modelos orientados a objetos um TAD é chamado de *classe*.

Os atributos de um TAD podem ser classificados em *monovalorados* ou *multivalorados*:

1. Atributos Monovalorados

   O valor de um atributo monovalorado é um objeto o qual pode ser:

   - um literal (objeto sem identificador). Neste caso denominamos de "atributo monovalorado de valor", o qual pode ser um literal *atômico*, ou *estruturado*.
   - uma referência para um objeto. Neste caso denominamos de "atributo monovalorado de referência". No modelo objeto-relacional os atributos de referência são usados para representar relacionamentos entre objetos. O valor de um atributo monovalorado de referência é uma referência para um objeto. Por exemplo, o valor do atributo **dept_ref** do tipo EMPREGADO, definido na Figura 2.1, é uma referência a um objeto do tipo DEPT.

2. Atributos Multivalorados

   O valor de um atributo multivalorado pode ser:

   - uma coleção de literais (coleção de objetos sem identificadores). Neste caso denominamos de "atributo multivalorado de valor", o qual pode ser uma coleção de literais *atômicos*, ou *estruturados*.
   - uma coleção de referências para objetos. Neste caso denominamos de "atributo multivalorado de referência".

O Oracle disponibiliza duas maneiras para implementar atributos multivalorados: *Nested Tables* (tabelas aninhadas) e *Varrays*.

- *Nested Tables* são coleções não limitadas de elementos homogêneos. Uma Nested Table é um tabela que é representada como uma coluna dentro de outra tabela.

Ela é inicialmente densa, mas pode se tornar esparsa devido a exclusão de elementos. Os dados das nested tables são armazenadas em tabelas chamadas tabelas de armazenamento, onde cada elemento da tabela principal é mapeado na tabela de armazenamento. O atributo **emp_list** do tipo DEPT, definido na Figura 2.1, é um exemplo de atributo multivalorado que foi implementado no Oracle usando *nested table*. Note que o tipo da *nested table*, no nosso exemplo EMPREGADO_LISTA, tem que ser declarado separadamente.

- *Varrays* são coleções ordenadas e limitadas de elementos homogêneos e nunca são esparsas. Os varrays são armazenados como objetos contínuos. O atributo **efones** do tipo EMPREGADO, definido na Figura 2.1, é um exemplo de atributo multivalorado que foi implementado usando *varray*.

2.4. Esquema Objeto-Relacional e Representação Gráfica

A Unified Modeling Language (UML) tem se tornado um padrão para modelagem de objetos durante as fases de análise e projeto do desenvolvimento de softwares. Neste trabalho, propomos uma extensão da UML para modelar, além de tipo de objetos (chamado de classe na UML), também tabelas de objetos, tabelas relacionais e visões de objetos. A adaptação da UML feita neste trabalho é baseada em [60]. Desta forma, nós estendemos a UML através do uso de estereotipos. Um estereotipo é uma frase delimitada por "≺≺ ≻≻"que você utiliza como um símbolo para representar uma extensão "oficial"para a semântica da UML [Smarties].

Considere, por exemplo, o esquema objeto-relacional COMPANHIA apresentado na figura 2.1, composto pelos tipos EMPREGADO, DEPT e ENDERECO e pelas tabelas EMPREGADOS e DEPTS.

Na Figura 2.3 apresentamos a representação gráfica do esquema objeto-relacional COMPANHIA. Na notação gráfica os retângulos com o estereotipo ≺≺Tipo de Objeto≻≻ representam um TAD, os retângulos cujo estereotipos são ≺≺Visão de Objeto≻≻ representam uma tabela virtual de objetos e os retângulos com o estereotipo ≺≺Tabela≻≻ representam uma tabela de tuplas ou de objetos. Os atributos cujos tipos são pré-definidos são escritos dentro dos retângulos. Os demais atributos cujos tipos são definidos pelo usuário são representados por: seta simples para atributo monovalorado e seta dupla para atributo multivalorado, com o nome do atributo escrito próximo ao tipo que o define. Maiores detalhes sobre a notação gráfica serão apresentados à medida que novos conceitos forem sendo introduzidos.

Um esquema objeto-relacional é uma tripla $\mathbf{S = (T, R, I)}$, onde $\mathbf{T}$ é um conjunto de definições de tipos (TADs), $\mathbf{R}$ é um conjunto de tabelas e $\mathbf{I}$ é um conjunto de restrições de integridade. No restante desta seção considere o esquema $\mathbf{S = (T, R, I)}$ onde $\mathbf{T}_1$, $\mathbf{T}_2$,..., $\mathbf{T}_m$ são tipos de $\mathbf{T}$; $\mathbf{R}_1$, $\mathbf{R}_2$,..., $\mathbf{R}_n$ são tabelas de $\mathbf{R}$ e $\mathbf{I}$ são restrições de integridade de $\mathbf{S}$.

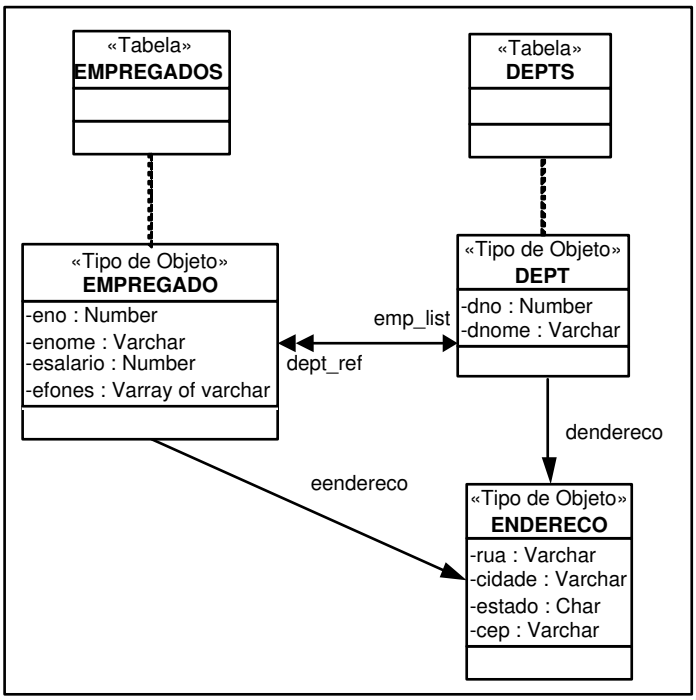

Figura 2.3. *Representação gráfica do esquema objeto-relacional* COMPANHIA

Um banco de dados muda ao longo do tempo por meio das informações que nele são inseridas ou excluídas. O conjunto de informações contidas em determinado banco de dados, em um dado momento, é chamado *estado* ou *instância* do banco de dados. A seguir definimos formalmente estado de um banco de dados objeto-relacional.

Definição 2.4.1. *(Estado de um Esquema Objeto-Relacional): Um estado (ou instância) $\mathcal{D}$ do esquema $\boldsymbol{S}$, em um dado instante, é uma função definida como se segue:*

- *Para qualquer tabela $\mathbf{R}_i$, $\mathcal{D}(\boldsymbol{R}_i)$ (lê-se extensão da tabela $\mathbf{R}_i$ no estado $\mathcal{D}$) é o conjunto de todos objetos* [1] *que são membros de $\mathbf{R}_i$ no estado $\mathcal{D}$.*

- *Para qualquer* atributo monovalorado de valor $\boldsymbol{a}$ *de um tipo $\boldsymbol{T}_{R_i}$, cujo valor é um objeto do tipo $\mathbf{T}_a$, $\mathcal{D}(\boldsymbol{a})$ atribui para cada objeto $\boldsymbol{o}_i$ de $\mathbf{T}_{R_i}$ um objeto $\boldsymbol{o}_a$ instância de $\mathbf{T}_a$. A notação $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$ indica o valor do atributo $\boldsymbol{a}$ para o objeto $\boldsymbol{o}_i$ no estado $\mathcal{D}$. Considere $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a}) = \boldsymbol{o}_a$, dizemos que $\boldsymbol{o}_i$ está relacionado com $\boldsymbol{o}_a$ através do atributo $\boldsymbol{a}$.*

- *Para qualquer* atributo monovalorado de referência $\boldsymbol{a}$ *de um tipo $\boldsymbol{T}_{R_i}$, cujo valor é uma referência a um objeto do tipo $\mathbf{T}_a$, $\mathcal{D}(\boldsymbol{a})$ atribui para cada objeto $\boldsymbol{o}_i$ de $\mathbf{T}_{R_i}$ uma referência a um objeto $\boldsymbol{o}_a$ instância de $\mathbf{T}_a$. A notação $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$ indica o valor*

[1] *Note que um objeto pode ser uma tupla quando $\boldsymbol{R}_i$ é uma* tabela de tuplas *ou um objeto com identificador quando $\boldsymbol{R}_i$ é uma* tabela de objetos.

do atributo $\boldsymbol{a}$ *para o objeto* $\boldsymbol{o}_i$ *no estado* $\mathcal{D}$. *Considere* $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a}) = \boldsymbol{o}_a$, *dizemos que* $\boldsymbol{o}_i$ *está relacionado com* $\boldsymbol{o}_a$ *através do atributo* $\boldsymbol{a}$.

- *Para qualquer* atributo multivalorado de valor $\boldsymbol{a}$ *de um tipo* $\boldsymbol{T}_{R_i}$, *cujo valor é uma coleção de objetos do tipo* $\mathbf{T}_a$, $\mathcal{D}(\boldsymbol{a})$ *atribui para cada objeto* $\boldsymbol{o}_i$ *de* $\boldsymbol{T}_{R_i}$ *uma coleção de objetos cujos membros são instâncias de* $\mathbf{T}_a$. *A notação* $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$ *indica o valor do atributo* $\boldsymbol{a}$ *para o objeto* $\mathbf{o}_i$ *no estado* $\mathcal{D}$. *Suponha que o objeto* $\boldsymbol{o}_a \in \boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$, *dizemos que* $\mathbf{o}_i$ *está relacionado com* $\mathbf{o}_a$ *através do atributo* $\boldsymbol{a}$.

- *Para qualquer* atributo multivalorado de referência $\boldsymbol{a}$ *de um tipo* $\boldsymbol{T}_{R_i}$, *cujo valor é uma coleção de referências para objetos do tipo* $\mathbf{T}_a$, $\mathcal{D}(\boldsymbol{a})$ *atribui para cada objeto* $\boldsymbol{o}_i$ *de* $\boldsymbol{T}_{R_i}$ *uma coleção de objetos cujos membros são referências a instânias de* $\mathbf{T}_a$. *A notação* $\boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$ *indica o valor do atributo* $\boldsymbol{a}$ *para o objeto* $\mathbf{o}_i$ *no estado* $\mathcal{D}$. *Suponha que o objeto* $\boldsymbol{o}_a \in \boldsymbol{o}_i \bullet \mathcal{D}(\boldsymbol{a})$, *dizemos que* $\mathbf{o}_i$ *está relacionado com* $\mathbf{o}_a$ *através do atributo* $\boldsymbol{a}$.

Para simplificar, as referências ao estado corrente $\mathcal{D}$ serão omitidas quando possível. Por exemplo, suponha o objeto $\mathbf{o}$, ao invés de $\mathbf{o} \bullet \mathcal{D}(\boldsymbol{a})$, será usado $\mathbf{o} \bullet \boldsymbol{a}$.

2.5. Visões de Objetos

Um banco de dados geralmente é compartilhado por uma variedade de usuários e deve atender aos diferentes requisitos desses usuários. Dessa forma, muitas vezes, os bancos de dados tornam-se grandes e complexos, dificultando assim a sua manipulação. Para facilitar a manipulação dos dados, devem ser fornecidas interfaces, que apresentem somente as informações relevantes para cada grupo de usuários. Isto é feito através da definição de visões, que representam modelos simplificados do banco de dados, através dos quais os usuários podem expressar consultas e atualizações. As visões além de protegerem o acesso aos dados, ajudam a alcançar um certo grau de independência lógica, uma vez que é possível alterar o esquema do banco de dados sem alterar uma visão.

Em bancos de dados relacionais, as visões relacionais são tabelas virtuais que derivam seus dados das tabelas relacionais nas quais estão baseadas, chamadas tabelas base da visão. Além das características das visões tradicionais, as visões de objetos possuem as seguintes características:

- Habilidade para navegar usando referências

  Os objetos das visões possuem identificadores, que fornecem a capacidade para referenciar objetos e serem referenciados por outros objetos. Utilizar referências para representar conexões inter-objetos tornam as consultas baseadas em caminhos de referência mais fáceis e compactas do que as consultas feitas com junção.

- Facilidade para evolução de esquemas

  As visões de objeto fornecem a flexibilidade de ver o mesmo dado relacional ou orientado a objetos de mais de uma maneira. Assim você pode ter mais de uma representação de objeto para diferentes aplicações sem ter que mudar os dados no banco de dados.

- Consistência com novas aplicações baseadas em objetos

  Se você necessita estender o projeto de um banco de dados legado, os novos componentes podem ser implementados em tabelas de objetos. As novas aplicações orientadas a objetos, que requerem acesso para aplicações orientadas a dados existentes, podem empregar um modelo de programação consistente. As aplicações legadas podem continuar a existir sem modificação.

- Integração de aplicações OO com banco de dados

  Uma forma de tratar a integração de aplicações orientadas a objetos com banco de dados é através do uso de visões de objetos. As principais vantagens da abordagem de se utilizar visões de objetos para essa integração é permitir o compartilhamento de informações e a escalabilidade e acesso a dados legados. As visões de objetos permitem que cada aplicação tenha seu próprio esquema de objetos, ao invés de todas as aplicações compartilharem o mesmo esquema de objetos fornecendo assim o compartilhamento das informações. Desta forma, novos objetos podem ser criados para satisfazer as novas visões dos usuários. As visões de objetos possibilitam a coexistência de aplicações relacionais e orientadas a objetos, o que torna fácil a introdução de aplicações orientadas a objetos para dados relacionais já existentes sem provocar uma mudança drástica de um paradigma para o outro.

Para criar uma visão de objetos temos primeiro que criar os tipos da visão. A Figura 2.4 contém a definição dos tipos da visão **Pedidos_v**. Na Figura 2.5 mostramos a representação gráfica do esquema da visão de objetos **Pedidos_v**.

A Figura 2.6 contém a definição da visão **Pedidos_v** baseada no esquema do banco de dados relacional PEDIDO_REL, apresentado na Figura 2.7. O esquema PEDIDO_REL é composto pelas tabelas **Clientes_rel**, **Pedidos_rel**, **Itens_rel** e **Produtos_rel**. A visão **Pedidos_v** contém um conjunto de objetos do tipo **Tpedido_v**, os quais são sintetizados a partir de objetos[2] das tabelas raízes.

[2]Note que objetos podem ser tuplas quando as tabelas raízes são *tabelas de tuplas* ou objetos quando as tabelas raízes são *tabelas de objetos*.

```
CREATE TYPE Tendereco_v AS OBJECT
(     rua        VARCHAR2(20),
      cidade   VARCHAR2(20),
      estado   CHAR(2),
      cep        VARCHAR2(10)      );

CREATE TYPE Tfone_v  AS OBJECT
(     numero  VARCHAR2(11)   );

CREATE TYPE Tfone_lista_v AS  TABLE
OF Tfone_v;

CREATE TYPE Tcliente_v AS OBJECT
(     codigo          NUMBER,
      nome            VARCHAR2(200),
      endereco      Tendereco_v,
      listaFone     Tfone_lista_v      );
```

```
CREATE TYPE Tpedido_v AS OBJECT
(     codigo              NUMBER,
      cliente_ref         REF Tcliente_v,
      data                 DATE,
      dataEntrega         DATE,
      lista                Titem_lista_v,
      enderEntrega        Tendereco_v      );
CREATE TYPE Tproduto_v AS OBJECT
(     codigo     NUMBER,
      preco      NUMBER,
      taxa        NUMBER     ) ;

CREATE TYPE Titem_v AS OBJECT
(     codigo          NUMBER,
      produto_ref  REF Tproduto_v,
      quantidade   NUMBER,
      desconto       NUMBER      );

CREATE TYPE Titem_lista_v AS TABLE
OF Titem_v;
```

Figura 2.4. *Esquema da visão* ***Pedidos_v***

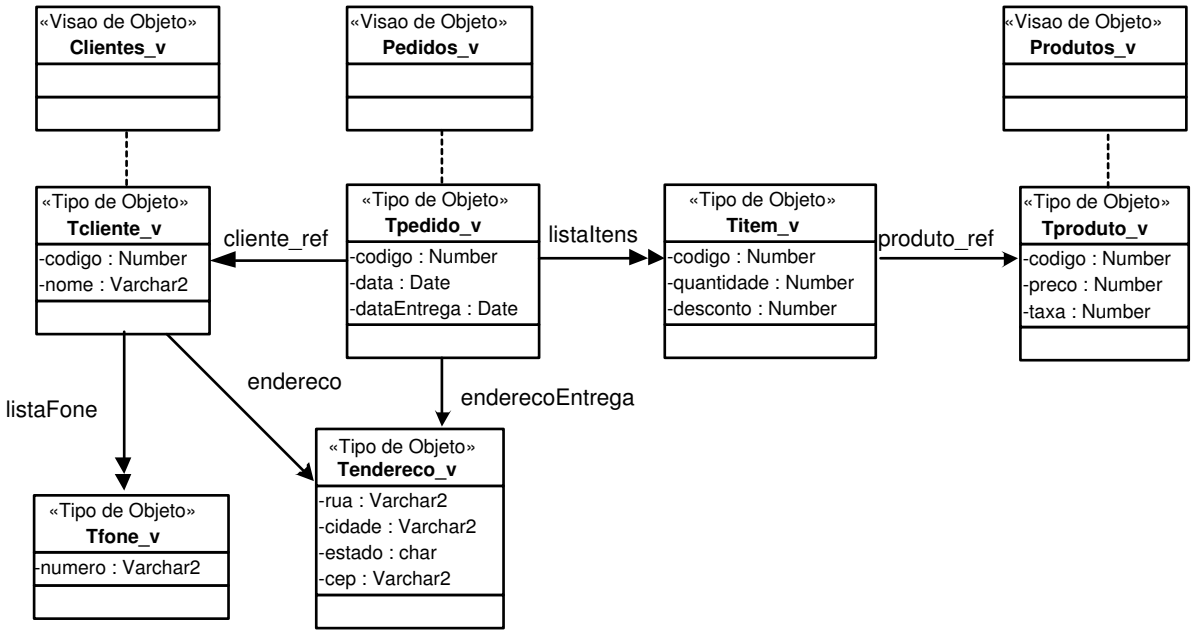

Figura 2.5. *Representação gráfica do esquema da visão* ***Pedidos_v***

```
CREATE VIEW Pedidos_v OF Tpedido_v
WITH OBJECT IDENTIFIER(codigo) AS
    SELECT P.Pcodigo,
        MAKE_REF(Clientes_v, P.pcliente), P.pdata, P.pdataEntrega,
        CAST( MULTISET(  SELECT Titem_v (I.icodigo,
                                         MAKE_REF(Produtos_v, I.iproduto),
                                         I.iquantidade, I.idesconto)
                         FROM Itens_rel I
                             WHERE I.ipedido = P.pcodigo)
            AS Titem_lista_v),
        Tendereco_v (P.prua,P.pcidade, P.pestado, P.pcep)
    FROM Pedidos_rel P;
```

* O operador MAKE_REF cria uma referência da visão Pedidos_v para a visão Clientes_V.
* O operador CAST converte o resultado para um tipo apropriado, no nosso exemplo para o tipo coleção Titem_lista_v.
* A palavra-chave MULTISET indica que o resultado é um conjunto de valores.

Figura 2.6. *Definição da visão* ***Pedidos_v***

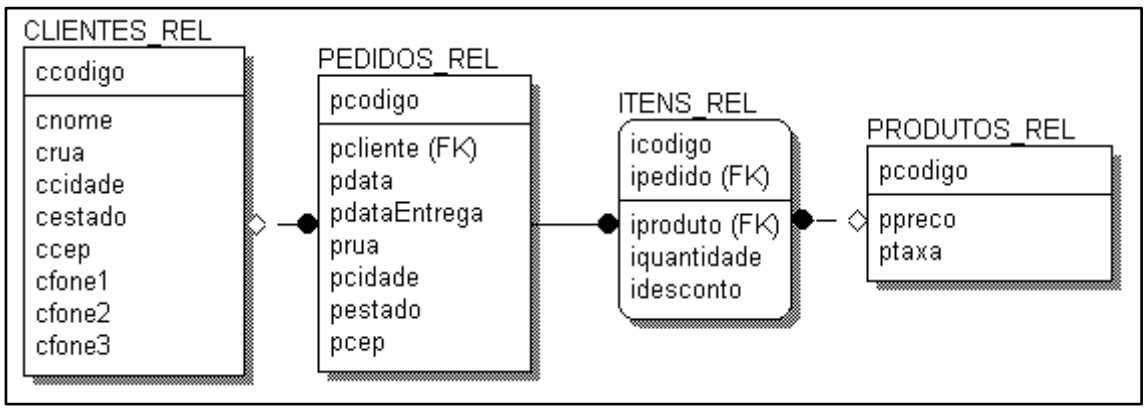

Figura 2.7. *Esquema do banco de dados* PEDIDO_REL

3

Assertivas de Correspondência no Modelo Objeto Relacional

Neste trabalho usamos assertivas de correspondências para especificar formalmente a correspondência entre o esquema da visão de objetos e o esquema objeto-relacional do banco de dados. Consideramos apenas as visões que preservam objetos. Uma visão preserva os objetos quando cada objeto da visão é semanticamente equivalente a um objeto de uma tabela base. Os objetos $\mathbf{o}_1$ e $\mathbf{o}_2$ são semanticamente equivalentes ($\mathbf{o}_1 \equiv \mathbf{o}_2$) sss representam uma mesma entidade do mundo real.

Neste capítulo, definimos as assertivas de correspondências no modelo Objeto-Relacional, as quais são classificadas em: assertivas de correspondência de extensão, assertivas de correspondência de objetos, assertivas de correspondência de caminho. O capítulo é organizado como se segue. Na Seção 3.1, definimos os conceitos preliminares que são necessários para a definição formal dos vários tipos de correspondências entre os esquemas. Na Seção 3.2, definimos as assertivas de correspondências de extensão. Na Seção 3.3, definimos as assertivas de correspondências de caminho. Na Seção 3.4, definimos as assertivas de correspondências de objetos. Na Seção 3.5, apresentamos o processo de geração das assertivas de correspondências de uma visão.

3.1. Terminologias

Nesta seção, apresentamos algumas definições necessárias para a definição formal das assertivas de correspondências formalizadas neste capítulo. Para tratarmos de forma uniforme tabelas de objetos e tabelas de tuplas, assumimos que, dada uma tabela de tuplas $\mathbf{R}$, $\mathbf{T}_R$ é o tipo das tuplas de $\mathbf{R}$. Este tipo é definido implicitamente na própria criação da tabela $\mathbf{R}$.

Definição 3.1.1. *(Navegando através de chave estrangeira): Suponha a chave estrangeira*

FK = $\boldsymbol{R}_1[\boldsymbol{a}_1, ..., \boldsymbol{a}_n] \subset \boldsymbol{R}_2[\boldsymbol{b}_1, ..., \boldsymbol{b}_n]$. *Dada uma tupla* $\boldsymbol{t}_1$ *de* $\boldsymbol{R}_1$ *a expressão* $\boldsymbol{t}_1 \bullet$FK *retorna a tupla* $\boldsymbol{t}_2$ *de* $\boldsymbol{R}_2$ *tal que* $\boldsymbol{t}_1 \bullet \boldsymbol{a}_i = \boldsymbol{t}_2 \bullet \boldsymbol{b}_i$, *para* $1 \leq i \leq n$. *Dado uma tupla* $\boldsymbol{t}_2$ *de* $\boldsymbol{R}_2$ *a expressão* $\boldsymbol{t}_2 \bullet \text{FK}^{-1}$ *retorna todas as tuplas* $\boldsymbol{t}_1$ *de* $\boldsymbol{R}_1$ *tal que* $\boldsymbol{t}_1 \bullet \boldsymbol{a}_i = \boldsymbol{t}_2 \bullet \boldsymbol{b}_i$, *para* $1 \leq i \leq n$. *Assim temos que* $\boldsymbol{T}_{R_1}$ *referencia* $\boldsymbol{T}_{R_2}$ *através de* FK *e* $\boldsymbol{T}_{R_2}$ *referencia* $\boldsymbol{T}_{R_1}$ *através da inversa de* FK (FK^{-1}).

Definição 3.1.2. *(Ligação de Tipos): Usamos o conceito de ligação para representar o relacionamente entre tipos. Considere* $\boldsymbol{T}_1$ *e* $\boldsymbol{T}_2$ *tipos de um esquema. Existe uma ligação de* $\boldsymbol{T}_1$ *para* $\boldsymbol{T}_2$ *nas seguintes situações:*
(i) (Ligação de atributo de valor) $\boldsymbol{T}_1$ *contém um atributo* $\boldsymbol{a}$ *cujo tipo é* $\boldsymbol{T}_2$ *ou coleção de objetos do tipo* $\boldsymbol{T}_2$. *Assim,* $\boldsymbol{a}$: $\boldsymbol{T}_1 \rightarrow \boldsymbol{T}_2$ *é uma ligação de* $\boldsymbol{T}_1$ *para* $\boldsymbol{T}_2$;
(ii) (Ligação de atributo de referência) $\boldsymbol{T}_1$ *contém um atributo* $\boldsymbol{a}$ *cujo tipo é uma referência ou uma coleção de referências para objetos do tipo* $\boldsymbol{T}_2$. *Assim,* $\boldsymbol{a}$: $\boldsymbol{T}_1 \rightarrow \boldsymbol{T}_2$ *é uma ligação de* $\boldsymbol{T}_1$ *para* $\boldsymbol{T}_2$;
(iii) (Ligação de chave estrangeira) Existe uma chave estrangeira FK *tal que* $\boldsymbol{T}_1$ *referencia* $\boldsymbol{T}_2$ *através de* FK. *Assim,* FK: $\boldsymbol{T}_1 \rightarrow \boldsymbol{T}_2$ *é uma ligação de* $\boldsymbol{T}_1$ *para* $\boldsymbol{T}_2$;
(iv) (Inversa de ligação) $\boldsymbol{T}_2$ *tem uma ligação* ℓ: $\boldsymbol{T}_2 \rightarrow \boldsymbol{T}_1$ *tal que* ℓ *é uma ligação de atributo de valor, de atributo de referência ou de restrição referencial. Então, a inversa da ligação* ℓ, *dada por* ℓ^{-1}: $\boldsymbol{T}_1 \rightarrow \boldsymbol{T}_2$ *é uma ligação de* $\boldsymbol{T}_1$ *para* $\boldsymbol{T}_2$.

As ligações (i), (ii) e (iii) são chamadas de "ligações diretas". Uma ligação é considerada "virtual"quando a mesma é definida a partir de uma ligação inversa. Uma ligação é considerada "de referência"quando:(i) é uma ligação de atributo de referência ou (ii) é a inversa de uma ligação de atributo de referência ou (iii) é uma ligação de chave estrangeira ou (iv) é a inversa de uma ligação de chave estrangeira.

Definição 3.1.3. *(Caminho): Considere as ligações* ℓ_1: $\boldsymbol{T}_1 \rightarrow \boldsymbol{T}_2$, ℓ_2: $\boldsymbol{T}_2 \rightarrow \boldsymbol{T}_3$, ... , ℓ_{n-1}: $\boldsymbol{T}_{n-1} \rightarrow \boldsymbol{T}_n$, *onde* $\boldsymbol{T}_1$, $\boldsymbol{T}_2$, ..., $\boldsymbol{T}_n$ *são tipos de um esquema objeto-relacional. Então,* $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ *é um caminho de* $\boldsymbol{T}_1$. *Isso significa que as instâncias de* $\boldsymbol{T}_1$ *podem estar relacionadas com as instâncias de* T_n *através do caminho* φ. *Dada uma instância* $\boldsymbol{t}_1$ *de* $\boldsymbol{T}_1$ *a expressão* $\boldsymbol{t}_1 \bullet \varphi$ *retorna um conjunto de tuplas* $\boldsymbol{t}_n$ *de* $\boldsymbol{T}_n$ *tal que existem as instâncias* $\boldsymbol{t}_2$, $\boldsymbol{t}_3$, ..., $\boldsymbol{t}_{n-1}$ *onde* $\boldsymbol{t}_i$ *está relacionada com* $\boldsymbol{t}_{i+1}$ *através da ligação* ℓ_i, *para* $1 \leq i \leq n$. *Assim, o tipo do caminho* φ *é* $\boldsymbol{T}_n$ ($\boldsymbol{T}_\varphi = \boldsymbol{T}_n$). *Se as ligações* $\ell_1, \ell_2, ..., \ell_{n-1}$ *são monovaloradas, então* φ *é um "caminho monovalorado", caso contrário* φ *é um "caminho multivalorado". Se* ℓ_{n-1} *é uma ligação de referência então* φ *é um "caminho de referência", caso contrário* φ *é um "caminho de valor".*

3.2. Assertivas de Correspondência de Extensão

As Assertivas de Correspondência de Extensão (ACE) especificam a correspondência existente entre a extensão da visão com as extensões das tabelas bases. A Tabela 3.1 contém a definição das ACs de Extensão. Existe uma assertiva de correspondência de extensão

entre uma visão **V** e uma tabela **R**, quando **V** e **R** têm objetos em comum (objetos semanticamente equivalentes). Neste caso, existe uma função de mapeamento **f** entre os objetos de **V** e os objetos de **R**. Se um objeto **v** de **V** é mapeado no objeto **r** de **R** (**f**(**v**)=**r**), então **v** e **r** são "semanticamente equivalentes"(**v**≡**r**), isto é, correspondem ao mesmo objeto no mundo real. As assertivas de correspondência de objetos definidas na Seção 3.4 são usadas para especificar as funções de mapeamento. Em geral **V** e **R** têm um identificador comum, o qual é usado como função de mapeamento.

Nós definimos *tabelas raízes* de uma visão **V** como todas as tabelas que estão relacionadas a **V** através de alguma ACE, o que significa dizer que existem objetos nas tabelas raízes que são semanticamente equivalentes aos objetos da visão.

Assertivas de Correspondência de Extensão		
Relação	**Notação**	**Definição**
Subconjunto	$\mathbf{V} \subset \mathbf{R}$	Existe uma função injetiva f: $\mathcal{D}(\mathbf{V}) \rightarrow \mathcal{D}(\mathbf{R})$
Equivalência	$\mathbf{V} \equiv \mathbf{R}$	Existe uma função bijetiva f: $\mathcal{D}(\mathbf{V}) \rightarrow \mathcal{D}(\mathbf{R})$
Diferença	$\mathbf{V} \equiv \mathbf{R}_1$ - $\mathbf{R}_2$	Existe uma função bijetiva f: $\mathcal{D}(\mathbf{V}) \rightarrow \mathcal{D}(\mathbf{R}_1)$ - $\mathcal{D}(\mathbf{R}_2)$
Intersecção	$\mathbf{V} \equiv \cap_{i=1}^{n} \mathbf{R}_i$	Existe uma função bijetiva f: $\mathcal{D}(\mathbf{V}) \rightarrow \cap_{i=1}^{n} \mathcal{D}(\mathbf{R}_i)$

Tabela 3.1. *Assertivas de Correspondência de Extensão*

Exemplo 3.1

Nos exemplos deste capítulo, usaremos a visão de objetos **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e os esquemas do banco de dados BD_1 e BD_2 mostrados nas Figuras 3.2 e 3.3 respectivamente. A visão **Pedidos_v** está relacionada com o esquema do banco de dados BD_1 através da seguinte ACE ψ:**Pedidos_v**≡**Pedidos**. ψ especifica que a visão **Pedidos_v** e a tabela **Pedidos** são equivalentes, isto é, para cada objeto de **Pedidos** existe um objeto semanticamente equivalente em **Pedidos_v**. Neste caso, **Pedidos** é a tabela raiz da visão **Pedidos_v**.

```
CREATE OR REPLACE TYPE Tendereco_v AS OBJECT
(     rua       VARCHAR2(20),
      cidade    VARCHAR2(20),
      estado    CHAR(2),
      cep       VARCHAR2(10)      );

CREATE OR REPLACE TYPE Titem_v AS OBJECT
(     codigo            NUMBER,
      quantidade        NUMBER,
      desconto          NUMBER,
      produto_codigo NUMBER     );

CREATE OR REPLACE TYPE Titem_lista_v AS TABLE
OF Titem_v;
```

```
CREATE OR REPLACE TYPE Tfone_lista_v AS
VARRAY(10) of VARCHAR2(20);

CREATE OR REPLACE TYPE Tpedido_v AS OBJECT
(     codigo            NUMBER,
      data              DATE,
      dataEntrega       DATE,
      cliente_codigo    NUMBER,
      cliente_nome      VARCHAR2(20),
      listaFone         Tfone_lista_v,
      listaItensTitem_lista_v,
      enderecoEntrega Tendereco_v      );
```

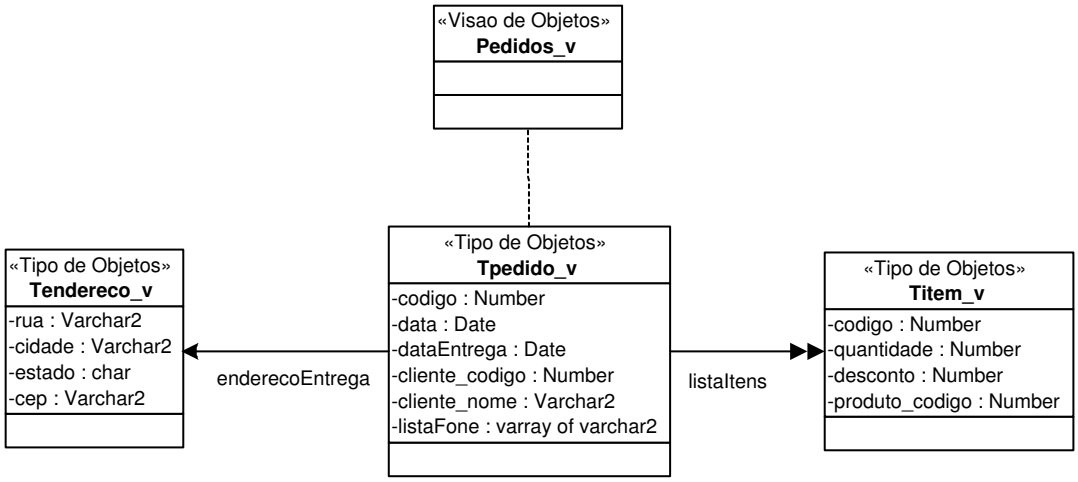

Figura 3.1. *Esquema da visão de objetos* ***Pedidos_v***

3.3. Assertivas de Correspondência de Caminhos

As Assertivas de Correspondência de Caminhos (ACC) especificam relacionamentos entre caminhos dos tipos do esquema da visão com caminhos dos tipos das tabelas bases. No resto desta seção, considere $\mathbf{T}_v$ um tipo do esquema da visão $\mathbf{V}$ e $\mathbf{T}_1$ um tipo de uma tabela base, onde $\mathbf{T}_v$ e $\mathbf{T}_1$ são semanticamente relacionados; isto é, suas instâncias podem representar objetos semanticamente equivalentes. Mais formalmente, $\mathbf{T}_v$ e $\mathbf{T}_1$ são semanticamente relacionados sss (i) $\mathbf{T}_v$ é o tipo de $\mathbf{V}$, $\mathbf{T}_1$ é o tipo de $\mathbf{R}_1$ e existe uma ACE relacionando a visão $\mathbf{V}$ e a tabela $\mathbf{R}_1$; ou (ii) Existem caminhos φ_v e φ_1, tais que $\mathbf{T}_{\varphi_v}$=$\mathbf{T}_v$ e $\mathbf{T}_{\varphi_1}$=$\mathbf{T}_1$ e φ_v e φ_1 estão relacionados através de uma ACC.

As ACs de Caminhos são classificadas em ACC Monovalorados e ACC Multivalorados, as quais são definidas a seguir.

```
CREATE OR REPLACE TYPE Tendereco AS
OBJECT
(     rua        VARCHAR2(20),
      cidade   VARCHAR2(20),
      estado   CHAR(2),
      cep        VARCHAR2(10)      );

CREATE OR REPLACE TYPE Tfone_lista AS  TABLE
OF Tfone;

CREATE OR REPLACE TYPE Tfone  AS OBJECT
(     numero  VARCHAR2(11)   );

CREATE OR REPLACE TYPE Tcliente AS OBJECT
(     codigo          NUMBER,
      nome            VARCHAR2(200),
      endereco      Tendereco,
      listaFone     Tfone_lista      );

CREATE OR REPLACE TYPE Tproduto AS OBJECT
(     codigo     NUMBER,
      preco      NUMBER,
      taxa        NUMBER     ) ;

CREATE OR REPLACE TYPE Titem AS OBJECT
(     codigo          NUMBER,
      produto_ref  REF Tproduto,
      quantidade   NUMBER,
      desconto      NUMBER      );
```

```
CREATE OR REPLACE TYPE Titem_lista AS TABLE OF
Titem;

CREATE OR REPLACE TYPE Tpedido AS OBJECT
(     codigo              NUMBER,
      cliente_ref         REF Tcliente,
      data                 DATE,
      dataEntrega       DATE,
      lista                Titem_lista,
      enderecoEntrega    Tendereco      );

CREATE TABLE Pedidos of Tpedido (
  PRIMARY KEY (codigo),
  Foreign Key (cliente_ref) references Clientes)
  OBJECT ID PRIMARY KEY
  NESTED TABLE Lista STORE AS  POLine_ntab
((PRIMARY KEY(NESTED_TABLE_ID, codigo))
    ORGANIZATION INDEX COMPRESS)
   RETURN AS LOCATOR;

ALTER TABLE PoLine_ntab
  ADD (SCOPE FOR (produto_ref) IS Produtos);

CREATE Table Clientes of Tcliente (codigo Primary Key)
Object Id Primary Key;

CREATE Table Produtos of Tproduto (codigo Primary Key)
Object Id Primary Key;
```

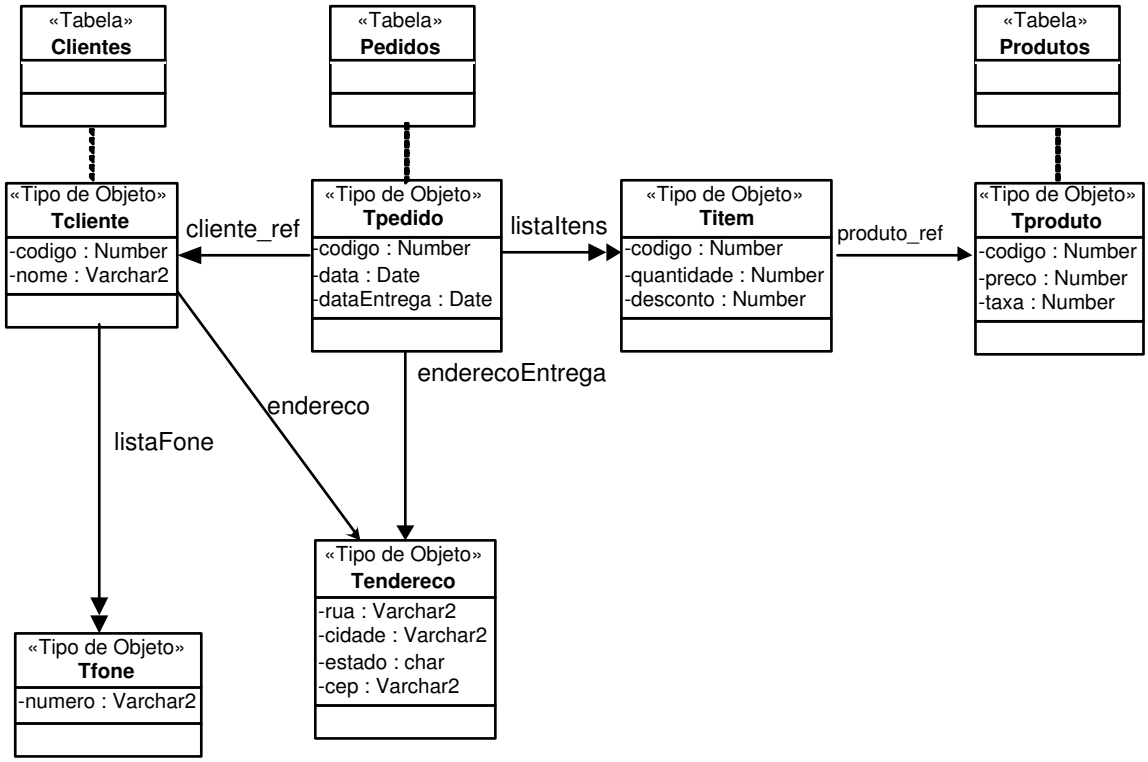

Figura 3.2. *Esquema do banco de dados* BD_1

ACs de Caminhos Monovalorados

As ACs de caminhos monovalorados podem ser de dois tipos:

Definição 3.3.1. *Sejam φ_v e φ_1 caminhos monovalorados de $\boldsymbol{T}_v$ e $\boldsymbol{T}_1$ respectivamente. A ACC $\boldsymbol{T}_v\bullet\varphi_v \equiv \boldsymbol{T}_1\bullet\varphi_1$ especifica que para quaisquer instâncias $\boldsymbol{t}_v$ de $\boldsymbol{T}_v$ e $\boldsymbol{t}_1$ de $\boldsymbol{T}_1$, se $\boldsymbol{t}_v \equiv \boldsymbol{t}_1$ então $\boldsymbol{t}_v\bullet\varphi_v \equiv \boldsymbol{t}_1\bullet\varphi_1$.*

Exemplo 3.2

Considere a visão **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e o esquema do banco de dados BD_1 apresentado na Figura 3.2.

A ACC **Tpedido_v•data ≡ Tpedido•data** especifica que para quaisquer instâncias $\mathbf{t}_v$ de **Tpedido_v** e $\mathbf{t}_1$ de **Tpedido**, se $\mathbf{t}_v\equiv \mathbf{t}_1$, então $\mathbf{t}_v$**•data** ≡ $\mathbf{t}_1$**•data**. Dado que o valor do atributo **data** é atômico então $\mathbf{t}_v$**•data** ≡ $\mathbf{t}_1$**•data** sss $\mathbf{t}_v$**•data** = $\mathbf{t}_1$**•data**.

A ACC **Tpedido_v•enderecoEntrega ≡ Tpedido•enderecoEntrega** especifica que dada uma instância $\mathbf{t}_v$ de **Tpedido_v**, se existir uma instância $\mathbf{t}_1$ de **Tpedido** tal que $\mathbf{t}_v\equiv \mathbf{t}_1$, então $\mathbf{t}_v$**•enderecoEntrega** ≡ $\mathbf{t}_1$**•enderecoEntrega**. Note que como o valor de **enderecoEntrega** é um objeto complexo, então devem ser geradas as ACC do tipo **Tendereco_v** com o tipo **Tendereco**. Na Seção 3.5, apresentamos o processo de geração das ACs que utiliza o algoritmo *Compara_Tipos* para gerar todas as ACCs que são necessárias.

A ACC **Tpedido_v•cliente_nome** ≡ **Tpedido•**FK_1**•cnome** especifica que dada uma instância $\mathbf{t}_v$ de **Tpedido_v**, se existir uma instância $\mathbf{t}_1$ de **Tpedido** tal que $\mathbf{t}_v\equiv \mathbf{t}_1$, então $\mathbf{t}_v$**•cliente_nome** = $\mathbf{t}_1$**•**FK_1**•cnome**.

Definição 3.3.2. *Suponha $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, ..., $\boldsymbol{a}_n$ atributos atômicos de $\boldsymbol{T}_1$ e $\boldsymbol{b}$ um atributo de $\boldsymbol{T}_v$, cujo tipo $\boldsymbol{T}_b$ contém os atributos atômicos $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, ..., $\boldsymbol{b}_n$. A ACC $[\boldsymbol{T}_v\bullet\boldsymbol{b}, \{ \boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_n\}] \equiv [\boldsymbol{T}_1, \{ \boldsymbol{a}_1, \boldsymbol{a}_2, ..., \boldsymbol{a}_n\}]$ especifica que para quaisquer instâncias $\boldsymbol{t}_v$ de $\boldsymbol{T}_v$ e $\boldsymbol{t}_1$ de $\boldsymbol{T}_1$, se $\boldsymbol{t}_v \equiv \boldsymbol{t}_1$ então $\boldsymbol{t}_v\bullet\boldsymbol{b}\bullet\boldsymbol{b}_i = \boldsymbol{t}_1\bullet\boldsymbol{a}_i$, para $1\leq i \leq n$.*

Exemplo 3.3

Considere a visão **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e o esquema do banco de dados BD_2 apresentado na Figura 3.3.

A ACC [**Tpedido_v•enderecoEntrega**, {**rua**, **cidade**, **estado**, **cep** }] ≡ [**Tpedido_rel**, { **prua**, **pcidade**, **pestado**, **pcep** }] especifica que para quaisquer instâncias $\mathbf{t}_v$ de **Tpedido_v** e $\mathbf{t}_1$ de **Tpedido_rel**, se $\mathbf{t}_v\equiv \mathbf{t}_1$, então $\mathbf{t}_v$**•endereco•rua** = $\mathbf{t}_1$**•prua**, $\mathbf{t}_v$**•endereco•cidade** = $\mathbf{t}_1$**•pcidade**, $\mathbf{t}_v$**•endereco•estado** = $\mathbf{t}_1$**•pestado**, $\mathbf{t}_v$**•endereco •cep** = $\mathbf{t}_1$**•pcep**.

ACs de Caminhos Multivalorados

As ACs de caminhos multivalorados podem ser de dois tipos, os quais são apresentados a seguir.

Definição 3.3.3. *Sejam φ_v e φ_1 caminhos multivalorados de $\boldsymbol{T}_v$ e $\boldsymbol{T}_1$ respectivamente. A ACC $\boldsymbol{T}_v \bullet \varphi_v \equiv \boldsymbol{T}_1 \bullet \varphi_1$ especifica que para quaisquer instâncias $\boldsymbol{t}_v$ de $\boldsymbol{T}_v$ e $\boldsymbol{t}_1$ de $\boldsymbol{T}_1$, se $\boldsymbol{t}_v \equiv \boldsymbol{t}_1$ então $\boldsymbol{t}_v \bullet \varphi_v \equiv \boldsymbol{t}_1 \bullet \varphi_1$. Dado que $\boldsymbol{t}_v \bullet \varphi_v$ e $\boldsymbol{t}_1 \bullet \varphi_1$ são coleções (de objetos atômicos, estruturados ou de referências) temos que $\boldsymbol{o}_v$ pertente a $\boldsymbol{t}_v \bullet \varphi_v$ sss existe $\boldsymbol{o}_1$ em $\boldsymbol{t}_1 \bullet \varphi_1$ tal que $\boldsymbol{o}_v \equiv \boldsymbol{o}_1$. No caso em que o tipo de $\boldsymbol{o}_v$ é um tipo complexo, então a ACO é utilizada para definir a correspondência entre objetos.*

Exemplo 3.4

Considere a visão **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e o esquema do banco de dados BD_1 apresentado na Figura 3.2.

A ACC **Tpedido_v•listaItens** ≡ **Tpedido•lista** especifica que para quaisquer instâncias $\mathbf{t}_v$ de **Tpedido_v** e $\mathbf{t}_1$ de **Tpedido**, se $\mathbf{t}_v \equiv \mathbf{t}_1$, então $\mathbf{o}_v$ pertence a $\mathbf{t}_v$•**listaItens** sss existe um objeto $\mathbf{o}_1$ em $\mathbf{t}_1$•**lista** tal que $\mathbf{o}_v \equiv \mathbf{o}_1$.

Definição 3.3.4. *Seja $\boldsymbol{b}$ um atributo atômico multivalorado de $\boldsymbol{T}_v$ e φ_1 um caminho monovalorado de $\boldsymbol{T}_1$ (cujo tipo é $\boldsymbol{T}_{\varphi_1}$). Sejam $\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_n$ atributos atômicos monovalorados de $\boldsymbol{T}_{\varphi_1}$. A ACC $\boldsymbol{T}_v \bullet \boldsymbol{b} \equiv [\ T_1 \bullet \varphi_1, \{\ a_1, a_2, ..., a_n\ \}\ ]$ especifica que para quaisquer instâncias $\boldsymbol{t}_v$ de $\boldsymbol{T}_v$ e $\boldsymbol{t}_1$ de $\boldsymbol{T}_1$, se $\boldsymbol{t}_v \equiv \boldsymbol{t}_1$ então $\boldsymbol{o}_v$ pertence a $t_v \bullet \boldsymbol{b}$ sss existe um objeto o_1 em $\{t_1 \bullet \varphi_1 \bullet a_1, t_1 \bullet \varphi_1 \bullet a_2, ..., t_1 \bullet \varphi_1 \bullet a_n\}$, tal que $o_v = o_1$.*

Exemplo 3.5

Considere a visão **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e o esquema do banco de dados BD_2 apresentado na Figura 3.3.

A ACC **Tpedido_v•listaFone** ≡ [**Tpedido_rel**•FK_1, {**pfone1**, **pfone2**, **pfone3** }] especifica que para quaisquer instâncias $\mathbf{t}_v$ de **Tpedido_v** e $\mathbf{t}_1$ de **Tpedido_rel**, se $\mathbf{t}_v \equiv \mathbf{t}_1$, então $\mathbf{t}_v$•**listaFone** ≡ {$\mathbf{t}_1$•FK_1•**pfone1**, $\mathbf{t}_1$•FK_1•**pfone2**, $\mathbf{t}_1$•FK_1•**pfone3**}.

3.4. Assertivas de Correspondência de Objetos

As Assertivas de Correspondência de Objetos (ACO) especificam sob que condições dois objetos, os quais são instâncias de tipos semanticamente relacionados, representam o mesmo objeto do mundo real, ou seja, são semanticamente equivalentes.

Definição 3.4.1. *Considere $\boldsymbol{T}_v$ um tipo do esquema da visão e $\boldsymbol{T}_1$ um tipo de uma tabela base, os quais são semanticamente relacionados. Sejam $\mathbf{a}_1$, ..., $\mathbf{a}_n$ atributos monovalorados*

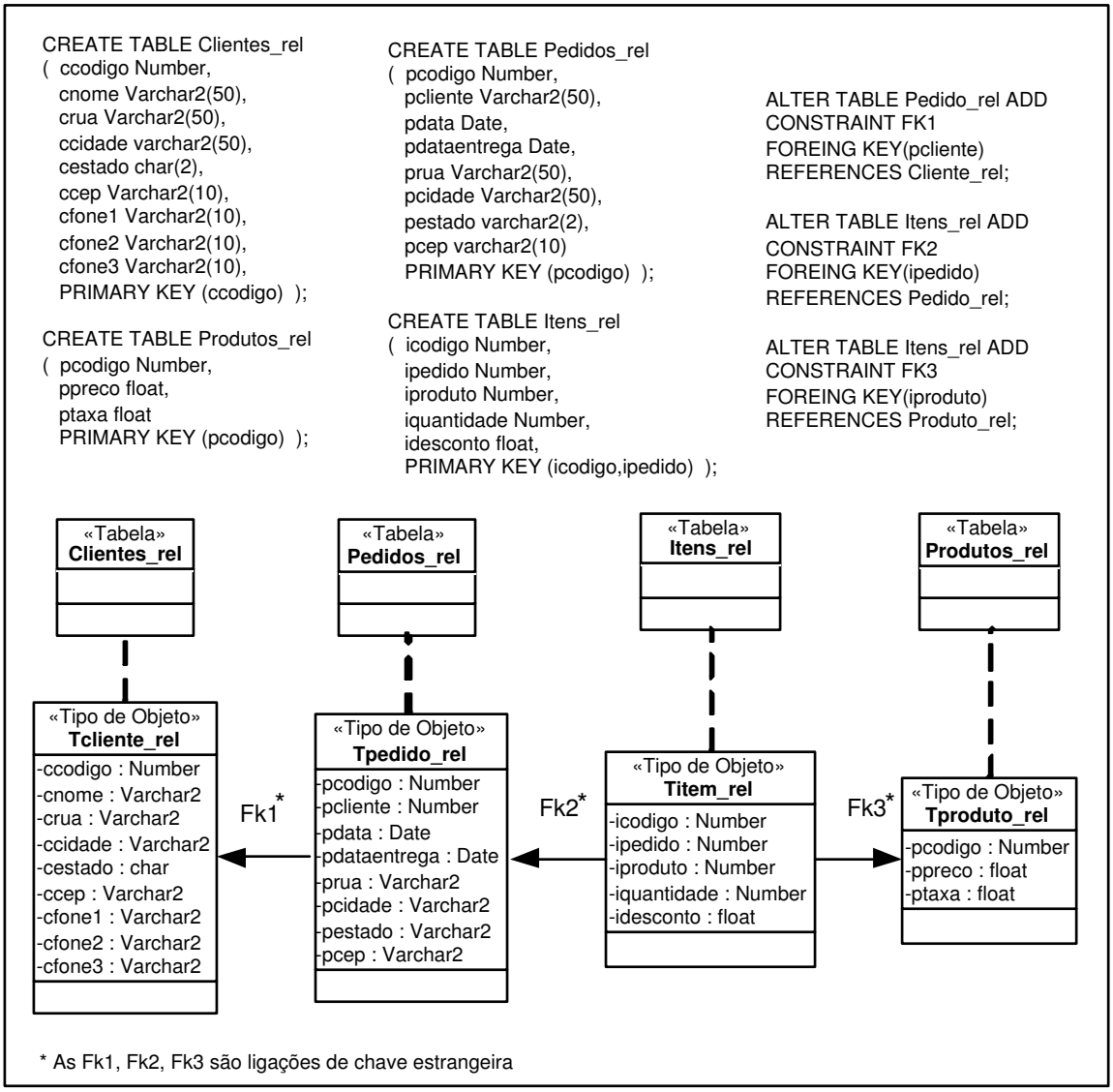

Figura 3.3. *Esquema do banco de dados* BD_2

de $\boldsymbol{T}_v$ *e* $\boldsymbol{b}_1$, ..., $\boldsymbol{b}_n$ *atributos de* $\boldsymbol{T}_1$. *A ACO* ψ: *[*$\boldsymbol{T}_v$, { $\mathbf{a}_1$, ..., $\mathbf{a}_n$}*]* $\equiv$ *[*$\boldsymbol{T}_1$, { $\boldsymbol{b}_1$, ..., $\boldsymbol{b}_n$}*]* *especifica que para qualquer instância* $\boldsymbol{t}_v$ *de* $\boldsymbol{T}_v$ *e* $\boldsymbol{t}_1$ *de* $\boldsymbol{T}_1$, *se* $\boldsymbol{t}_v \bullet \mathbf{a}_i = \boldsymbol{t}_1 \bullet \boldsymbol{b}_i$, $1 \leq i \leq n$, *então* $\boldsymbol{t}_v \equiv \boldsymbol{t}_1$.

Exemplo 3.6

Considere a visão **Pedidos_v** cujo esquema é apresentado na Figura 3.1 e o esquema do banco de dados BD_1 apresentado na Figura 3.2.

A ACO ψ: [**Tpedido_v**, {**codigo** }] $\equiv$ [**Tpedido_rel**, {**pcodigo** }] especifica que para quaisquer instâncias $\mathbf{t}_v$ de **Tpedido_v** e $\mathbf{t}_1$ de **Tpedido**, se $\mathbf{t}_v \bullet$**codigo** $= \mathbf{t}_1 \bullet$**codigo**, então $\mathbf{t}_v \equiv \mathbf{t}_1$.

3.5. Gerando as Assertivas de Correspondência de uma Visão de Objetos

O processo de geração das asssertivas de correspondência de uma visão de objetos **V** do tipo $\mathbf{T}_v$ consiste dos seguintes passos:

Passo 1: Especificação da ACE
Neste passo, deve ser especificada a ACE da visão **V** com as tabelas raízes[1].

Passo 2: Especificação das ACO e ACCs
Para cada tabela raiz **R** de **V**, onde **R** é do tipo $\mathbf{T}_R$, faça:

Passo 2.1: Especificação da ACO
Neste passo, as chaves de **V** e **R** são comparadas para serem identificadas as ACs dos objetos de $\mathbf{T}_v$ com os objetos de $\mathbf{T}_R$ (função de matching entre $\mathbf{T}_v$ e $\mathbf{T}_R$).

Passo 2.2: Especificação das ACCs
Neste passo, os tipos $\mathbf{T}_v$ e $\mathbf{T}_R$ são comparados, usando o algoritmo *Compara_tipos* da Figura 3.4, de forma a gerar as ACCs entre $\mathbf{T}_v$ e $\mathbf{T}_R$. Note que o algoritmo *Compara_tipos* é recursivo.

Compara_Tipos(T_1, T_2)

Se os tipos T_1 e T_2 não foram comparados

1.Identifica ACs relacionando caminhos de T_1 com caminhos de T_2.

2.Para cada ACC ψ identificada no passo 1, onde ψ relaciona o caminho φ_1 de T_1 com o caminho φ_2 de T_2 ($T_1 \bullet \varphi_1 = T_2 \bullet \varphi_2$) tal que T_{φ_1} e T_{φ_2} são tipos complexos ou referências para tipos complexos, faça:

2.1.{Identifica a ACO de T_{φ_1} com T_{φ_2}}

2.2 Compara_Tipos(T_{φ_1}, T_{φ_2})

Figura 3.4. *Algoritmo* Compara_Tipos

A seguir, usamos o processo descrito acima para geramos as ACs entre o esquema da visão **Pedidos_v** e o esquema do banco de dados BD_2.

[1]Como definido na Seção 3.2, as *tabelas raízes* de uma visão **V** são todas as tabelas que estão relacionadas a **V** através de alguma ACE.

Passo 1: A visão **Pedidos_v** está relacionada com a tabela **Pedidos_rel** através da ACE ψ_1: **Pedidos_v** $\equiv$ **Pedidos_rel**. ψ_1 especifica que para cada objeto $\mathbf{o}_1$ em **Pedidos_v**, existe um objeto $\mathbf{o}_2$ em **Pedidos_rel** tal que $\mathbf{o}_1 \equiv \mathbf{o}_2$ e vice-versa.

Passo 2: De ψ_1 temos que **Pedidos_rel** é a tabela raiz de **Pedidos_v**. Neste passo, deve-se especificar as ACO e AC de caminhos de **Tpedido_rel** com **Tpedido_v**.

Passo 2.1: A ACO que indica as condições de matching entre as instâncias de **Tpedido_v** e **Tpedido_rel** é dada por:
ψ_2: [**Tpedido_v**, {**codigo** }]$\equiv$[**Tpedido_rel**, {**pcodigo**}].

Passo 2.2: As ACs de caminhos entre **Tpedido_v** e **Tpedido_rel** são obtidas através da comparação dos tipos **Tpedido_v** e **Tpedido_rel** de acordo com o algoritmo da Figura 3.4. Comparando estes tipos são identificadas 7 ACC:
ψ_3: **Tpedido_v•codigo** $\equiv$ **Tpedido_rel•pcodigo**,
ψ_4: **Tpedido_v•data** $\equiv$ **Tpedido_rel•pdata**,
ψ_5: **Tpedido_v•dataEntrega** $\equiv$ **Tpedido_rel•pdataEntrega**,
ψ_6: [**Tpedido_v•enderecoEntrega**, {**rua, cidade, estado, cep** }] $\equiv$ [**Tpedido_rel**, {**prua, pcidade, pestado, pcep** }]
ψ_7: **Tpedido_v•cliente_codigo** $\equiv$ T**pedido_rel•pcliente**
ψ_8: **Tpedido_v•cliente_nome** $\equiv$ **Tpedido_rel•**FK_1**•cnome**
ψ_9: **Tpedido_v•ListaItens** $\equiv$ **Tpedido_rel•**$\mathrm{FK}_2{}^{-1}$

De acordo com o algoritmo *Compara_Tipos*, como na ACC ψ_9 o tipo de ListaItens é complexo, deve-se identificar as ACC e ACO entre **Titem_v** e **Titem_rel**. As seguintes assertivas são obtidas através da comparação entre os tipos **Titem_v** e **Titem_rel** de acordo com o algoritmo da Figura 3.4:

ψ_{10}: [**Titem_v**, {**codigo** }] $\equiv$ [**Titem_rel**, {**icodigo** }]
ψ_{11}: **Titem_v•codigo** $\equiv$ **Titem_rel•icodigo**
ψ_{12}: **Titem_v•quantidade** $\equiv$ **Titem_rel•iquantidade**
ψ_{13}: **Titem_v•desconto** $\equiv$ **Titem_rel•idesconto**
ψ_{14}: **Titem_v•produto_codigo** $\equiv$ **Titem_rel•iproduto**,

4

Definindo Tradutores para Atualizações em Visões de Objetos

Neste capítulo, discutimos o nosso enfoque para a atualização de bancos de dados objeto-relacionais através de visões de objetos. Na seção 4.1 abordamos o problema de atualização de visões e descrevemos o processo de tradução de atualização de visões. Na seção 4.2 apresentamos o nosso enfoque para definir tradutores para atualização de visões de objetos.

4.1. Atualização de Banco de Dados Através de Visões

Na arquitetura de três níveis de esquemas, as visões constituem interfaces através das quais os usuários acessam e atualizam o banco de dados, consultas e atualizações especificadas nas visões devem ser traduzidas em consultas e atualizações a serem executadas no banco de dados. Uma definição de visão consiste de: (i) o mapeador de instâncias que especifica como o estado do banco de dados em um determinado instante é mapeado no estado da visão correspondente; (ii) o mapeador de atualizações que especifica como atualizações especificadas na visão são traduzidas em uma seqüência de atualizações especificadas no banco de dados.

Uma atualização de visão é simplesmente uma atualização que é especificada em uma visão, mas que deve ser traduzida em uma seqüência de atualizações no banco de dados. Chamamos esta sequência de atualizações do banco de dados de tradução da atualização da visão requisitada. O processo de tradução de atualização de visão (TAV) é descrito no diagrama da Figura 4.1 . O estado inicial do banco de dados D é mapeado pelo mapeador de instâncias σ_v no estado da visão σ_v(D). O usuário especifica a atualização u sobre o estado da visão. A atualização de visão u deve ser traduzida em uma seqüência de atualizações sobre o banco de dados $\tau(u)$. $\tau(u)$ é executada no estado do banco de dados D para obter o novo estado do banco de dados D'=$\tau(u)$(D). Com o novo estado do banco

de dados D', obtemos o novo estado da visão correspondente σ_v(D').

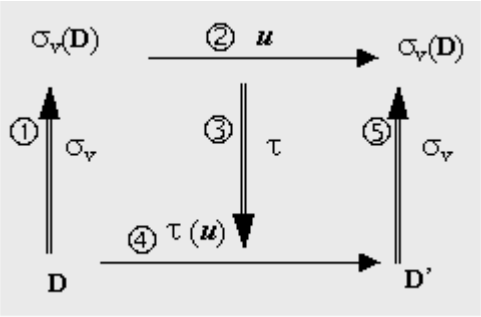

Figura 4.1. *Tradução de atualização através de visão (TAV)*

Algumas vezes, uma tradução de uma atualização na visão u pode causar mudanças adicionais no banco de dados além dos efeitos diretos desejados por u. Estas mudanças adicionais são chamadas "efeitos colaterais". Além disso, como sabemos, uma atualização em um banco de dados pode disparar outras atualizações, as quais são necessárias para corrigir violações das restrições de integridade do esquema do banco de dados. Assim sendo, o novo estado de uma visão vai depender do estado do banco de dados e de suas restrições de integridade.

Um pedido de atualização submetido a visão pode ter nenhuma, uma, ou múltiplas traduções. Uma das dificuldades associada ao problema de TAV surge quando existe mais de uma tradução possível, e apenas uma deve ser escolhida. Existem algumas formas de ambigüidades que podem ser resolvidas em "tempo de definição"da visão. É o que chamamos de ambigüidades a "nível de esquema", uma vez que estas são causadas pela existência de mais de um componente do esquema onde se pode realizar a atualização. Nesse caso, a escolha do componente mais apropriado pode ser feita em tempo de definição da visão. No nosso enfoque, a preferência é dada para a tradução que cause o mínimo de efeitos colaterais na visão. No caso das ambigüidades a "nível de dados", que são causadas pela existência de mais de uma instância onde se pode realizar a atualização, não podemos resolve-las em tempo de projeto. Estas formas de ambigüidades só podem ser resolvidas em tempo de atualização, e para resolvê-las se faz necessário um diálogo com o usuário (em tempo de atualização) para que ele escolha a instância mais apropriada.

4.2. Usando as ACs na Geração dos Tradutores de Atualização da Visão

No nosso enfoque, o tradutor de atualizações de uma visão consiste de um conjunto de tradutores, um para cada uma das operações de atualização de visão de objetos que forem permitidas. Um tradutor é uma função que recebe como entrada um pedido de atualização u e gera a tradução para u, a qual consiste de uma seqüência de atualizações que devem

ser realizadas no banco de dados, de maneira que este fique consistente com o novo estado da visão, supondo-se que u fosse realizado diretamente na visão. Os tradutores são definidos em tempo de projeto. Uma vez definido o tradutor, o usuário especifica atualizações através da visão e o tradutor as traduz em atualizações no banco de dados, sem a necessidade de qualquer diálogo adicional.

O nosso enfoque só pode ser aplicado a uma classe limitada de visões denominadas "independentes dos dados", que são estas visões cujas atualizações podem ser definidas antecipadamente na definição da visão sem nenhuma consideração dos dados. No caso de ambigüidades a nível de esquema deve-se escolher o componente do esquema que cause o mínimo de efeitos colaterais no banco de dados.

Neste trabalho, desenvolvemos algoritmos que recebem como entrada: o esquema da visão, o esquema do banco de dados e as ACs da visão; e geram tradutores para as operações básicas de atualização de bancos de dados objeto-relacionais. No Apêndice A são apresentados os seguintes algoritmos:

- Algoritmo C1 - Gera os tradutores para adição de um objeto em uma coleção aninhada da visão
- Algoritmo C2 - Gera os tradutores para remoção de um objeto de uma coleção aninhada da visão
- Algoritmo V3 - Gera os tradutores para modificação de atributos monovalorados de visão
- Algoritmo V1 - Gera os tradutores para adição de um objeto em uma visão
- Algoritmo V2 - Gera os tradutores para remoção de um objeto de uma visão

Como discutido no Capítulo 3, as ACs da visão especificam formalmente o relacionamento do esquema da visão e o esquema do banco de dados. No nosso enfoque, as ACs são tratadas como restrições de integridade que devem ser preservadas pelas as operações de atualização na visão. Desta forma, o problema de definição do tradutor para uma dada operação de atualização da visão de objetos, consiste em determinar a seqüência de atualizações no banco de dados que são requeridas para a manutenção das assertivas das correspondências "relevantes"para esta operação, isto é as ACs que podem ser violadas por essa operação. As vantagens do uso das ACs é que permitem especificar as condições em que é possível definir um tradutor em tempo de projeto, isto é, não existe ambigüidade a nível de dados, e permitem determinar a seqüência de atualizações necessárias no banco de dados para realizar uma atualização solicitada na visão de objetos.

No banco de dados Oracle 8i os tradutores de atualizações nas visões de objetos são implmentados utilizando os *instead of triggers*. O *instead of trigger* é usado para traduzir a atualização sobre a visão de objetos em atualizações nas tabelas bases. O código do

instead of trigger é executado quando adições, atualizações ou remoções são solicitadas na visão.

A Figura 4.2 mostra, a título de ilustração, o *instead of trigger* "Adiciona Pedido"o qual traduz a adição de um pedido na visão **Pedidos_v** em uma adição na tabela base **Pedidos**, cujos esquemas estão definidos nas Figura 3.1 e 3.3 do Capítulo 3, respectivamente. O tradutor é gerado a partir das ACs da visão **Pedidos_v** definidas na Seção 3.5. No trigger da Figura 4.2 mostramos as ACs que geraram cada linha de comando do trigger.

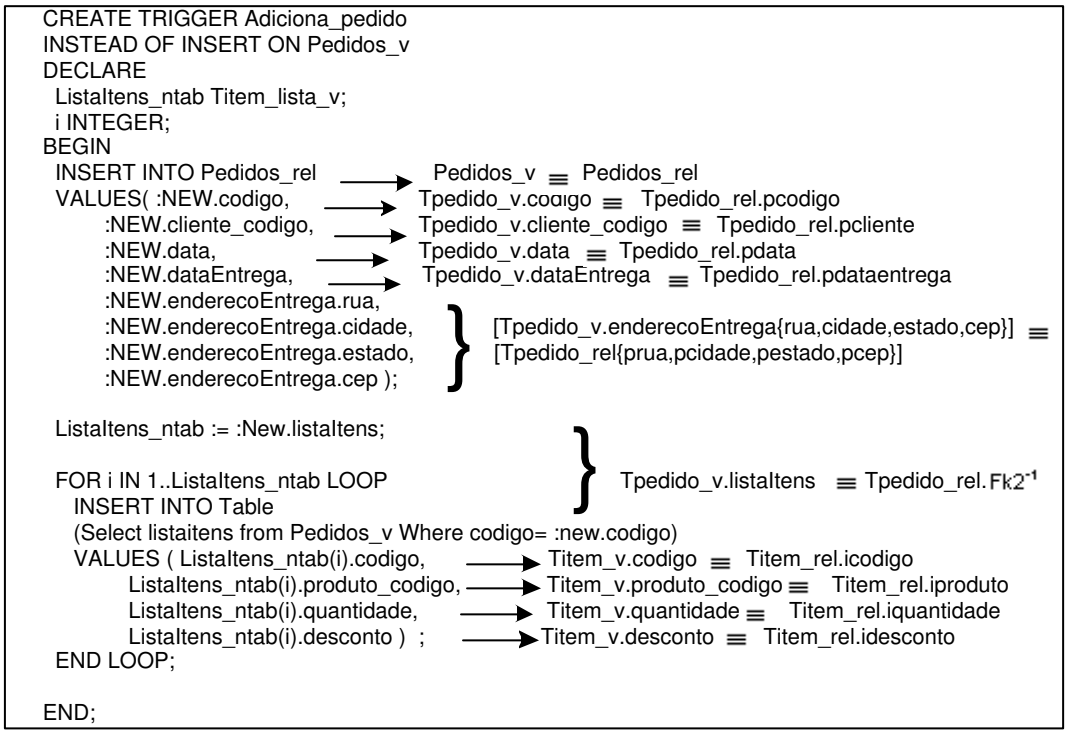

Figura 4.2. *Instead of Trigger Adiciona_pedido*

5

Algoritmos para Geração de Tradutores para Atualizações de Visões de Objetos

Neste capítulo, descrevemos os algoritmos que geram tradutores para as operações de atualização de visões de objetos em banco de dados objeto-relacional. Os algoritmos recebem como entrada: o esquema da visão, o esquema do banco de dados, as assertivas de correspondências da visão e a semântica de atualização que é fornecida pelo projetista da visão; e geram tradutores para as operações básicas de atualização de bancos de dados objeto-relacionais.

Este capítulo está organizado com se segue. Na seção 5.1 apresentamos o algoritmo que gera tradutores para as operações de adição de objetos em coleções aninhadas. O algoritmo que gera tradutores para as operações de remoção de objetos de coleções aninhadas é definido na seção 5.2. Na seção 5.3 descrevemos o algoritmo que gera tradutores para as operações de modificação de atributos monovalorados de objetos de visões. Na seção 5.4 apresentamos o algoritmo que gera tradutores para as operações de adição de objetos em visões e finalmente na seção 5.5 definimos o algoritmo que gera tradutores para as operações de remoção de objetos de visões.

5.1. Definindo Tradutores para Operações de Adição em Coleções Aninhadas

Nesta seção, considere a visão $\mathbf{V}$ cujos objetos são do tipo $\mathbf{T}_v$ e $\mathbf{lista\_T}_c$ um atributo multivalorado (coleção aninhada) de $\mathbf{T}_v$, cujos objetos são do tipo $\mathbf{T}_c$, como mostrado na Figura 5.1. Suponha $\mathbf{o}_v$ um objeto da visão $\mathbf{V}$. Considere um pedido de adição do objeto $\mathbf{o}_c$ do tipo $\mathbf{T}_c$ na coleção $\mathbf{lista\_T}_c$ do objeto $\mathbf{o}_v$. Esse pedido de atualização no nosso formalismo é definido por:
"Adicione $\mathbf{o}_c$ em $\mathbf{o}_v \bullet \mathbf{lista\_T}_c$"

O algoritmo C1, apresentado no Apêndice A, gera tradutores para adição na coleção

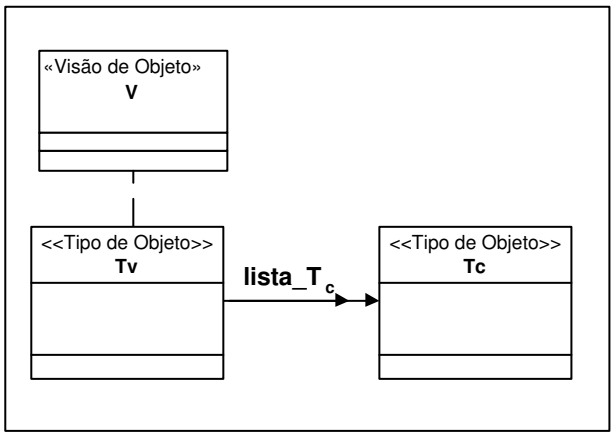

Figura 5.1. *Visão* $\boldsymbol{V}$ *do tipo* $\boldsymbol{T}_v$ *com atributo multivalorado* $\boldsymbol{lista\_T}_c$

aninhada $\mathbf{lista\_T}_c$ da visão $\mathbf{V}$. No resto dessa seção, considere $\mathbf{T}_v\bullet\mathbf{lista\_T}_c \equiv \mathbf{T}_{R_1}\bullet\varphi$ a assertiva de correspondencia de $\mathbf{lista\_T}_c$, onde $\mathbf{T}_{R_1}$ é o tipo da tabela base $\mathbf{R}_1$ e φ é um caminho de $\mathbf{T}_{R_1}$. O algoritmo C1 gera tradutores de atualização para os dois casos descritos a seguir. Em ambos os casos, só poderá existir uma única ligação multivalorada no caminho de derivação de $\mathbf{lista\_T}_c$ (φ), uma vez que na existência de mais de uma ligação ocorrerá ambigüidade da nível de dados. O Caso 1 trata a situação em que a ligação multivalorada é a ultima ligação de φ, e o Caso 2 a situação em que a ligação multivalorada não é a ultima. Mostraremos que nestes casos não existe ambigüidade a nível de dados e a tradução pode ser gerada em tempo de projeto.

5.1.1. Caso 1: No caminho de derivação de Lista_$\mathbf{T}_c$ a ligação multivalorada é a última

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ é de um dos tipos:
   $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1$, $\mathbf{a}_2$, ... , $\mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1$, $\mathbf{b}_2$, ... , $\mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência de $\mathbf{lista\_T}_c$ é dada por:
   $\mathbf{T}_v\bullet\mathbf{lista\_T}_c \equiv \mathbf{T}_{R_1}\bullet\varphi$, onde:

   3.1. $\varphi = \ell_1\bullet\ell_2\bullet$ ... $\bullet\ell_{n-1}$ é um caminho de $\mathbf{T}_{R_1}$ e ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_{R_i}\rightarrow\mathbf{T}_{R_{i+1}}$, para 1≤i≤n-1, como apresentado na Figura 5.2;

3.2. As ligações ℓ_1, ..., ℓ_{n-2} e suas inversas são ligações monovaloradas e a ligação ℓ_{n-1} é multivalorada. Desta forma a ligação multivalorada é a última ligação do caminho φ e cada objeto de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{n-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{n-2}$ e vice-versa.

4. As assertivas de correspondência de caminhos monovalorados entre $\mathbf{T}_c$ e $\mathbf{T}_{R_n}$ podem ser de um dos tipos definidos abaixo:

4.1. $\mathbf{T}_c \bullet \mathbf{a}_c \equiv \mathbf{T}_{R_n} \bullet \mathbf{c}$, onde $\mathbf{a}_c$ é um atributo de $\mathbf{T}_c$ e $\mathbf{c}$ é um atributo de $\mathbf{T}_{R_n}$;

4.2. $\mathbf{T}_c \bullet \mathbf{a}_c \equiv \mathbf{T}_{R_n} \bullet \varphi$**'**, onde $\mathbf{a}_c$ é um atributo de $\mathbf{T}_c$, $\varphi' = \ell'_1 \bullet \ell'_2 \bullet ... \bullet \ell'_{k-1} \bullet \mathbf{c}$ é um caminho de $\mathbf{T}_{R_n}$ e ℓ'_i é uma ligação definida por ℓ'_i: $\mathbf{T}_{R'_i} \rightarrow \mathbf{T}_{R'_{i+1}}$, 1≤i≤k-1, tal que:

4.2.1. ℓ'_1 é a ligação monovalorada direta cuja inversa pode ser multivalorada ou monovalorada. As ligações ℓ'_2, ..., ℓ'_{k-1} e suas inversas são monovaloradas. Desta forma, temos que cada objeto de $\mathbf{R'}_2$ está relacionado com um único objeto de $\mathbf{R'}_k$ e vice-versa;

4.2.2. $\mathbf{c}$ é um atributo de $\mathbf{T}_{R'_k}$, tal que $\mathbf{c}$ é um identificador de $\mathbf{R'}_k$. Esta condição garante que com o valor de $\mathbf{c}$ iremos recuperar apenas uma instância de $\mathbf{R'}_k$;

Estas condições garantem que não existe ambigüidade a nível de dados. No caso de existir mais de uma ligação multivalorada no caminho φ ou se a ligação multivalorada não é a última ligação de φ, então existe ambigüidade a nível de dados.

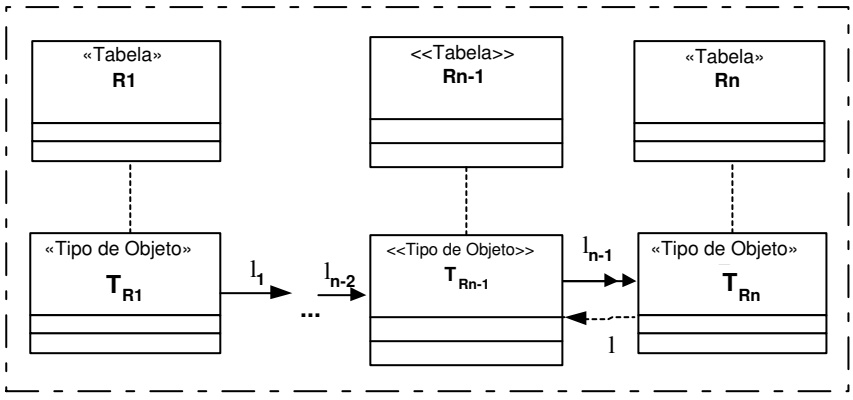

Figura 5.2. *Caminho* $\varphi = \ell_1 \bullet ... \bullet \ell_{n-2} \bullet \ell_{n-1}$ *do tipo base* $\boldsymbol{T}_{R_1}$

De acordo com o Algoritmo C1, o Caso 1 possui três subcasos que precisam ser tratados separadamente:

- O **Caso 1.1** trata a situação em que a ligação multivalorada ℓ_{n-1} é direta; [1]
- O **Caso 1.2** trata a situação em que a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação multivalorada;
- O **Caso 1.3** trata a situação em que a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação monovalorada.

É importante lembrar que os casos 1.1 e 1.2 só se aplicam para banco de dados objeto-relacionais uma vez que não existe atributo multivalorado em bancos de dados relacionais. A seguir discutimos cada caso do algoritmo.

[1] Como visto no Capítulo 2, ℓ_{n-1} é uma ligação direta se: (i) ℓ_{n-1} é uma ligação de atributo de valor; ou (ii) ℓ_{n-1} é uma ligação de atributo de referência; ou (iii) ℓ_{n-1} é uma ligação de chave estrangeira.

5.1.1.1. Caso 1.1 - a ligação multivalorada ℓ_{n-1} é direta

Neste caso, o pedido de atualização, *"Adicione* $\boldsymbol{o_c}$ *em* $\boldsymbol{o_v \bullet lista\_T_c}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o_n}$ *em* $\boldsymbol{o_{n-1} \bullet \ell_{n-1}}$*"*, onde:

- $\mathbf{o}_n$ é uma instância de $\mathbf{T}_{R_n}$ semanticamente equivalente (SE) a $\mathbf{o}_c$;
- $\mathbf{o}_{n-1}$ é uma instância de $\mathbf{R}_{n-1}$, que está relacionada com a instância $\mathbf{o}_1$ de $\mathbf{R}_1$ ($\mathbf{o}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{n-2}$ ($o_1 \bullet \ell_1 \bullet ... \bullet \ell_{n-2} = o_{n-1}$).

A Figura 5.3 mostra o trecho do Algoritmo C1 que trata o Caso 1.1. A seguir descrevemos cada linha do algoritmo.

Caso 1.1: a ligação multivalorada ℓ_{n-1} é direta.

```
 /* Seleciona ou cria o objeto o_n */
1.      Se ℓ_{n-1} é uma coleção aninhada de referência
2.        τ:=τ ∪{≺o_n:=(Selecione Referência(r_n) de R_n r_n onde r_n≡o_c);≻}
         /* o_n é uma referência a uma instância de R_n tal que r_n≡o_c */
3.      senão Se ℓ_{n-1} é uma coleção aninhada de valor estruturado
4.        Construtor_o_n : = Cria_Construtor_Objeto_Coleção(T_{R_n}, T_c);
         /* Retorna um construtor de objeto do tipo T_{R_n}*/
5.        τ := τ ∪ { ≺ o_n : = Construtor_o_n; ≻ }
6.      senão /* ℓ_{n-1} é uma coleção aninhada de valor atômico */
7.        τ := τ ∪ { ≺ o_n : = o_c; ≻ }

 /* Seleciona o objeto o_{n-1} */
8.      Se n=2 /* o caminho φ é composto de apenas uma ligação */
9.        τ := τ ∪ {≺ o_{n-1} := (Selecione r_1 de R_1 onde r_1≡o_v); ≻}
10.     senão /* n>2 */
11.       τ:=τ∪ {≺ o_{n-1} := (Selecione r_1•ℓ_1•...•ℓ_{n-2} de R_1 onde r_1≡o_v);≻}
         /* o_{n-1} é uma instância de R_{n-1}, tal que o_{n-1} está relacionado
          com r_1 através do caminho ℓ_1•ℓ_2•...  •ℓ_{n-2} */

 /* Gera a atualização */
12.      τ := τ ∪ { ≺ Adicione o_n em o_{n-1}•ℓ_{n-1} ; ≻ }
```

Figura 5.3. *Caso 1.1 do Algoritmo C1*

Linhas 1-7: Seleciona ou cria o objeto $\mathbf{o}_n$.

Linhas 1-2: Se a ligação ℓ_{n-1} é uma coleção aninhada de referência então $\mathbf{o}_n$ recebe a referência do objeto r_n de R_n tal que r_n é SE ao objeto o_c. Neste caso o objeto referenciado r_n já deve existir, pois o mesmo deve ser selecionado e não criado.

Linhas 3-5: Se ℓ_{n-1} é uma coleção de valor estruturado então $\mathbf{o}_n$ recebe o objeto criado pelo construtor de objetos **Construtor_o**$_n$ gerado na linha 4, o qual cria um objeto do tipo $\mathbf{T}_{R_n}$ SE ao objeto o_c (adicionado em **lista_T**$_c$). O construtor de objetos **Construtor_o**$_n$ do tipo $\mathbf{T}_{R_n}$ foi gerado pelo procedimento Cria_Construtor_Objeto _Coleção, apresentado no Apêndice A.

Este procedimento recebe como entrada os tipos $\mathbf{T}_1$ e $\mathbf{T}_c$, onde $\mathbf{T}_1$ é um tipo base e $\mathbf{T}_c$ é o tipo de uma coleção aninhada, e gera o construtor de objetos do tipo $\mathbf{T}_1$, que cria um objeto de $\mathbf{T}_1$ semanticamente equivalente a um dado objeto $\mathbf{o}_c$ de $\mathbf{T}_c$. Para gerar um construtor de objetos, que cria um objeto $\mathbf{o}_1$ de $\mathbf{T}_1$ a partir de um objeto $\mathbf{o}_c$ de $\mathbf{T}_c$, deve-se determinar valores para os atributos de $\mathbf{o}_1$ a partir de $\mathbf{o}_c$. Como mostrado no algoritmo Cria_Construtor_Objeto_Coleção, os valores dos atributos de $\mathbf{o}_1$ são determinados com base nas assertivas de correspondência entre os tipos $\mathbf{T}_1$ e $\mathbf{T}_c$ ($\mathbf{T}_1$ e $\mathbf{T}_c$ são semanticamente relacionados). Este formalismo permite definir o mapeamento entre os objetos de tipos com estruturas diferentes. Maiores detalhes deste procedimento serão apresentados durante as discussões dos exemplos.

Linhas 6-7: Se ℓ_{n-1} é uma coleção de valor atômico então $\mathbf{o}_n$ recebe o valor $\mathbf{o}_c$.

Linhas 8-11: Seleciona o objeto $\mathbf{o}_{n-1}$.

Linhas 8-9: Se n=2 (o caminho φ é composto de apenas uma ligação) então $\mathbf{o}_{n-1}$ recebe o objeto r_1 de R_1 tal que r_1 é SE ao objeto o_v, "pai"do objeto o_c (adicionado em **lista_T**$_c$).

Linhas 10-11: Se n>2 (o caminho φ é composto de mais de uma ligação), então $\mathbf{o}_{n-1}$ recebe o objeto r_{n-1} de R_{n-1} tal que r_{n-1} está relacionado com o objeto r_1 de R_1, onde r_1 é SE a o_v, através do caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$. De acordo com a condição 3.2 do Caso 1, o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ é monovalorado, bem como sua inversa. Assim, temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{n-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{n-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto possível definir o tradutor em tempo de projeto.

Linha 12: É requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_n$ na coleção aninhada ℓ_{n-1} do objeto $\mathbf{o}_{n-1}$ da tabela $\mathbf{R}_{n-1}$.

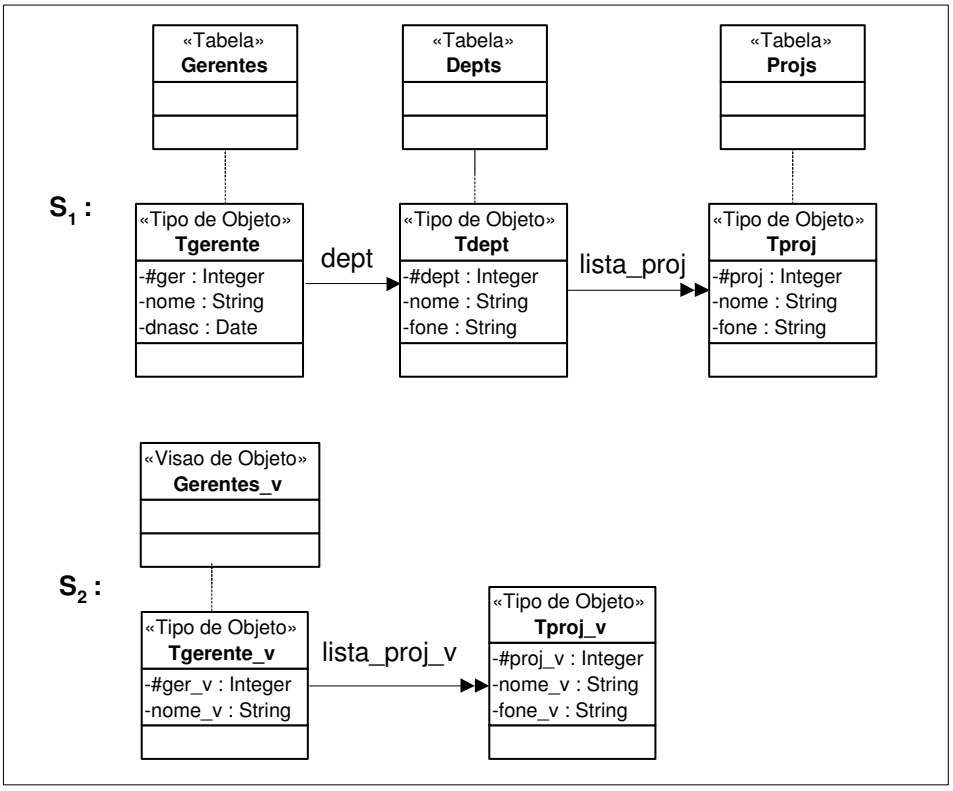

Figura 5.4. *Esquema do banco de dados* $\boldsymbol{S}_1$ *e esquema da visão de objetos* $\boldsymbol{S}_2$

Exemplo 5.1

Sejam $\mathbf{S}_1$, o esquema do banco de dados, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v**, apresentados na Figura 5.4. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_1$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v**≡**Gerentes**

- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
  ψ_3: [**Tproj_v**, {♯**proj_v**}] ≡ [**Tproj**, {♯**proj**}]

- Assertivas de Correspondência de Caminhos entre os tipos **Tgerente_v** e **Tgerente**
  ψ_4: **Tgerente_v•lista_proj_v** ≡ **Tgerente•dept•lista_proj**
  ψ_5: **Tgerente_v•♯ger_v** ≡ **Tgerente•♯ger**
  ψ_6: **Tgerente_v•nome_v** ≡ **Tgerente•nome**

- Assertivas de Correspondência de Caminhos entre os tipos **Tproj_v** e **Tproj**
  ψ_7: **Tproj_v•♯proj_v** ≡ **Tproj•♯proj**
  ψ_8: **Tproj_v•nome_v** ≡ **Tproj•nome**
  ψ_9: **Tproj_v•fone_v** ≡ **Tproj•fone**

A Figura 5.5 apresenta o tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$1', gerado pelo algoritmo C1, o qual é usado para traduzir a adição de um objeto na coleção aninhada **lista_proj_v** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualização na tabela base.

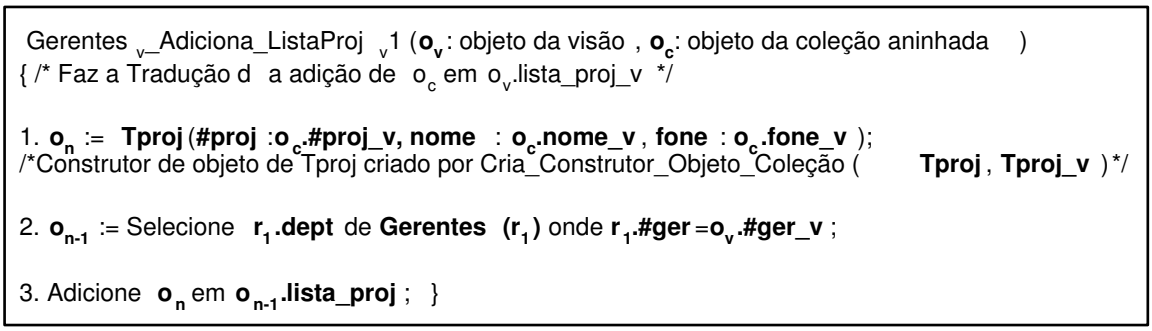

Figura 5.5. *Tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$1'*

De acordo com o algoritmo C1, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{n-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.4, a ligação multivalorada **lista_proj** é um atributo de **Tdept**, logo é uma ligação direta. Desta forma aplica-se o Caso 1.1 do algoritmo C1.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C1.

Linha 1: Dado que **lista_proj** não é uma coleção de referência, o_n recebe o objeto criado pelo construtor de objetos: ***Tproj(♯proj: o_c•♯proj_v, nome: o_c•nome_v, fone: o_c•fone_v)***, o qual cria um objeto do tipo **Tproj** sendo este semanticamente equivalente ao objeto o_c (adicionado em **lista_proj_v**). Este construtor de objetos foi gerado pelo procedimento Cria_Construtor_Objeto_Coleção (**Tproj**, **Tproj_v**) baseado nas ACs ψ_7, ψ_8 e ψ_9 que especificam formalmente o relacionamento entre **Tproj** e **Tproj_v**. Este formalismo permite ao construtor definir automaticamente o mapeamento entre objetos que possuem estruturas diferentes.

Linha 2: De ψ_4 temos que o caminho de derivação de **lista_proj_v** (**dept•lista_proj**) é composto de mais de uma ligação. Assim $\mathbf{o}_{n-1}$ recebe o objeto $\mathbf{r}_1$•**dept** de **Depts**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r}_1 \equiv \mathbf{o}_v$) sss ($\mathbf{r}_1$•♯**ger**= $\mathbf{o}_v$•♯**ger_v**). No algoritmo C1, assim como nos demais algoritmos, quando for solicitada uma instância através de um comando "selecione"e esta instância não existir no banco de dados, então trataremos esse erro como uma exceção. O uso das exceções permite-nos algumas ações corretivas caso os erros ocorram.

Linha 3: É gerada a atualização requerida; a qual consiste em adicionar o objeto $\mathbf{o}_n$ (instância de **Tproj** criada na linha 1) na coleção **lista_proj** do objeto $\mathbf{o}_{n-1}$ (instância de **Depts** selecionada na linha 2).

```
CREATE OR REPLACE TRIGGER Adiciona_Projeto1_Oracle
   INSTEAD OF INSERT ON NESTED TABLE lista_proj_v OF Gerentes_V
BEGIN
1.  Insert Into  TABLE(Select g.dept.lista_proj From Gerentes g
                        Where g.#ger= :PARENT.#ger_v)
     ( #proj, nome, fone )
2.  values  ( :new. #proj_v, :new.nome_v, :new.fone_v );
END;
```

Figura 5.6. *Tradutor 'Adiciona_Projeto1_Oracle'*

Na Figura 5.6, mostramos a título de ilustração o trigger 'Adiciona_Projeto1_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$1' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma adição na coleção **lista_proj_v** do objeto $\mathbf{o}_v$, este trigger é disparado. O valor *:new* refere-se ao objeto inserido, no nosso caso $\mathbf{o}_c$. O valor *:parent* refere-se ao objeto "pai"do objeto inserido, no nosso caso $\mathbf{o}_v$.

```
CREATE OR REPLACE TRIGGER Adiciona_Projeto2_Oracle
   INSTEAD OF INSERT ON NESTED TABLE lista_proj_v OF Gerentes_V
BEGIN
1.    Insert Into  TABLE(Select g.dept.lista_proj From Gerentes g
                        Where g.#ger= :PARENT.#ger_v)
2.    Select   Ref(p)  From  Projs p Where p.#proj= :new.#proj_v    ;
END;
```

Figura 5.7. *Tradutor 'Adiciona_Projeto2_Oracle'*

No caso de **lista_proj** ser de referência, o procedimento é similar. Na Figura 5.7, mostramos o tradutor gerado quando **lista_proj** é uma ligação de referencia. Verifique que a única mudança ocorre na linha 2.

5.1.1.2. Caso 1.2 - a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação multivalorada de referência.

Neste caso, o pedido de atualização, *"Adicione* $\boldsymbol{o}_c$ *em* $\boldsymbol{o}_v\bullet\boldsymbol{lista\_T}_c$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o}_{n-1}$ *em* $\boldsymbol{o}_n\bullet\ell$*"*, onde:

- $\mathbf{o}_n$ é uma instância de $\mathbf{R}_n$ (neste caso $\mathbf{o}_n$ já existe em $\mathbf{R}_n$) SE a $\mathbf{o}_c$;
- $\mathbf{o}_{n-1}$ é uma referência de uma instância de $\mathbf{R}_{n-1}$ que está relacionada com a instância $\mathbf{o}_1$ de $\mathbf{R}_1$ ($\mathbf{o}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1\bullet\ell_2\bullet$ ... $\bullet\ell_{n-2}$ ($o_1\bullet\ell_1\bullet ... \bullet\ell_{n-2} = o_{n-1}$).

A Figura 5.8 mostra o trecho do Algoritmo C1 que trata o Caso 1.2. A seguir descrevemos cada linha do algoritmo.

Caso 1.2: a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa de uma ligação multivalorada ℓ: $T_{R_n} \rightarrow T_{R_{n-1}}$.

```
  /* Seleciona o objeto o_{n-1} */
1.      Se n=2 /* o caminho φ é composto de apenas uma ligação */
2.        τ := τ ∪ { ≺ o_{n-1} := (Selecione Ref(r_1) de R_1 onde r_1≡o_v) ≻ };
3.      senão /* n>2 */
4.        τ:=τ ∪{≺ o_{n-1} :=(Selecione r_1•ℓ_1•...•ℓ_{n-2} de R_1 onde r_1≡o_v);≻}
          /* o_{n-1} é uma referência a uma instância de R_{n-1} tal que o_{n-1}
          está relacionada com r_1 através do caminho ℓ_1•ℓ_2•...  •ℓ_{n-2} */

  /* Seleciona o objeto o_n */
5.      τ := τ ∪ {≺ o_n := (Selecione r_n de R_n as r_n onde r_n≡o_c);≻ }
        /* o_n é uma instância de R_n tal que o_n≡o_c */

  /* Gera a atualização */
6.      τ := τ ∪ { ≺ Adicione o_{n-1} em o_n•ℓ ; ≻ }
```

Figura 5.8. *Caso 1.2 do Algoritmo C1*

Linhas 1-4: Seleciona o objeto $\mathbf{o}_{n-1}$.

Linhas 1-2: Se n=2 (o caminho φ é composto de apenas uma ligação), então $\mathbf{o}_{n-1}$ recebe a referência do objeto r_1 de R_1 tal que r_1 é SE ao objeto o_v, "pai"do objeto o_c (adicionado em $\mathbf{lista\_T}_c$).

Linhas 3-4: Se n>2 (o caminho φ é composto de mais de uma ligação), então $\mathbf{o}_{n-1}$ recebe a referência do objeto r_{n-1} de R_{n-1} tal que r_{n-1} está relacionado com o objeto r_1 de R_1, onde r_1 é SE a o_v, através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-2}$. Note que, de maneira análoga ao Caso 1.1, o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-2}$ é monovalorado (conforme condição 3.2 do Caso 1).

Linha 5: Seleciona o objeto $\mathbf{o}_n$. o_n recebe o objeto r_n de R_n tal que r_n é SE ao objeto o_c, adicionado na coleção $\mathbf{lista\_T}_c$. Neste caso, o objeto r_n já deve existir em R_n, pois o mesmo deve ser selecionado e não criado.

Linha 6: É requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_{n-1}$ de $\mathbf{T}_{R_{n-1}}$ na coleção aninhada ℓ do objeto $\mathbf{o}_n$ da tabela $\mathbf{R}_n$.

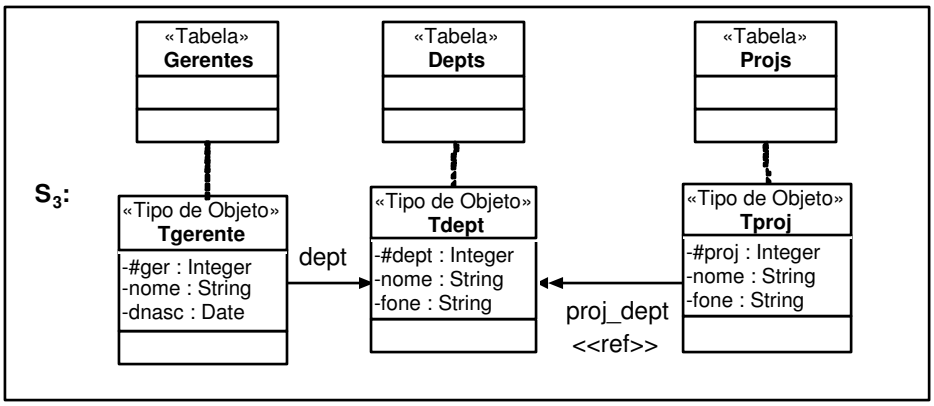

Figura 5.9. *Esquema do banco de dados S_3*

Exemplo 5.2

Considere $\mathbf{S}_3$, o esquema do banco de dados apresentado na Figura 5.9, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v** apresentado na Figura 5.4. Neste exemplo as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_3$ são semelhantes as especificadas para o Caso 1.1, com exceção da AC do atributo multivalorado **lista_proj_v** dada a seguir:

- Assertivas de Correspondência do Atributo **lista_proj_v**
  ψ_{10}: $\mathbf{Tgerente\_v \bullet lista\_proj\_v \equiv Tgerente \bullet dept \bullet proj\_dept^{-1}}$

A Figura 5.10 apresenta o tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$3', gerado pelo algoritmo C1, o qual é usado para traduzir a adição de um objeto na coleção aninhada **lista_proj_v** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualizações na tabela base.

De acordo com o algoritmo C1, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{n-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.9, a ligação multivalorada **lista_proj** é virtual, obtida da inversa da ligação

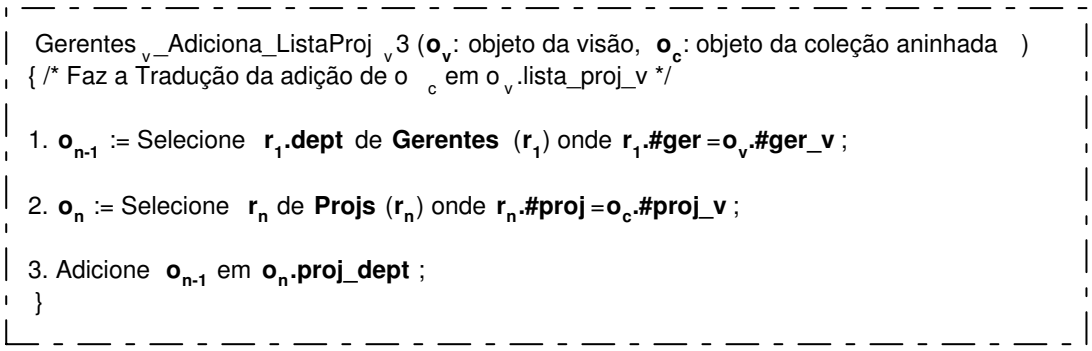

Figura 5.10. *Tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$3'*

multivalorada **proj_dept**, definida por **proj_dept**: **Tproj**→**Tdept**. Desta forma aplica-se o Caso 1.2 do algoritmo C1.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C1.

Linha 1: De ψ_{10} temos que o caminho de derivação de **lista_proj_v** (**dept•proj_dept**$^{-1}$) é composto de mais de uma ligação. Assim $\mathbf{o}_{n-1}$ recebe a referência do objeto $\mathbf{r}_1$•**dept** de **Depts**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$.

Linha 2: $\mathbf{o}_n$ recebe o objeto $\mathbf{r}_n$ de **Projs**, tal que $\mathbf{r}_n$ é SE ao objeto $\mathbf{o}_c$. Note que, de acordo com a AC de objeto ψ_3, ($\mathbf{r}_n \equiv \mathbf{o}_c$) sss ($\mathbf{r}_n$•♯**proj**= $\mathbf{o}_c$•♯**proj_v**).

Linha 3: É gerada a atualização requerida; a qual consiste em adicionar o objeto $\mathbf{o}_{n-1}$ (instância de **Depts** selecionado na linha 1) na coleção **proj_dept** do objeto $\mathbf{o}_n$ (instância de **Projs** selecionado na linha 2).

```
CREATE OR REPLACE TRIGGER Adiciona_Projeto3_Oracle
   INSTEAD OF INSERT ON NESTED TABLE lista_proj_v OF
Gerentes_V
 BEGIN
 Insert Into
  TABLE(Select p.proj_dept From PROJs p Where
          p.#proj= :new.#proj_v)
 Select g.dept From Gerentes g Where g.#ger= :Parent.#ger_v;
END;
```

Figura 5.11. *Tradutor 'Adiciona_Projeto3_Oracle'*

Na Figura 5.11, mostramos a titulo de ilustração o trigger 'Adiciona_Projeto3_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$3' gerado para o banco de dados Oracle 8i.

5.1.1.3. Caso 1.3 - a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação monovalorada.

Neste caso, o pedido de atualização, *"Adicione* $\boldsymbol{o_c}$ *em* $\boldsymbol{o_v}\bullet\boldsymbol{lista\_T_c}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o_n}$ *em* $\boldsymbol{R_n}$*"*, onde: $\mathbf{o}_n$ é uma instância do tipo $\mathbf{T}_{R_n}$, tal que $\mathbf{o}_n$ é SE a $\mathbf{o}_c$. Neste caso $\mathbf{o}_n$ ainda não é uma instância de $\mathbf{R}_n$. A Figura 5.12 mostra o trecho do Algoritmo C1 que trata o Caso 1.3. A seguir descrevemos cada linha do algoritmo.

Caso 1.3: a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa de uma ligação monovalorada ℓ: $T_{R_n} \rightarrow T_{R_{n-1}}$.

```
 /* Cria o objeto o_n */
1.  Construtor_o_n :  = Cria_Construtor_Objeto_Coleção(T_Rn , T_c);
    /*Cria um construtor de objeto do tipo T_Rn */
2.  τ := τ ∪ { ≺ o_n :  = Construtor_o_n; ≻ }

 /* Seleciona o objeto o_n-1 */
3.  Se n=2 /* o caminho φ é composto de apenas uma ligação */
4.      τ := τ ∪ {≺ o_n-1 := (Selecione r_1 de R_1 onde r_1≡o_v); ≻}
5.  senão /* n>2 */
6.      τ:=τ∪ {≺ o_n-1 := (Selecione r_1•ℓ_1•...•ℓ_n-2 de R_1 onde r_1≡o_v);≻}
        /* o_n-1 é uma instância de R_n-1, tal que o_n-1 está relacionado
        com r_1 através do caminho ℓ_1•ℓ_2•...  •ℓ_n-2 */

 /* Atribui o valor de o_n•ℓ */
7.  Se ℓ é uma ligação de referência
8.      τ := τ ∪ {≺ o_n•ℓ := (Selecione Ref(o_n-1); ≻ }
9.  Senão Se ℓ é uma chave estrangeira, definida por
10.  ℓ = R_n[a_1, ..., a_k] ⊂ R_n-1[d_1, ..., d_k]
11.      Para j de 1 até k Faça
12.        τ := τ ∪ {≺ o_n•a_j := o_n-1•d_j; ≻ }

 /* Gera a atualização */
13.  τ := τ ∪ { ≺ Adicione o_n em R_n; ≻ }
```

Figura 5.12. *Caso 1.3 do Algoritmo C1*

Linhas 1-2: Cria o objeto $\mathbf{o}_n$.

$\mathbf{o}_n$ recebe o objeto criado pelo construtor de objetos **Construtor_o**$_n$, o qual cria um objeto do tipo $\mathbf{T}_{R_n}$ SE ao objeto o_c (adicionado em **lista_T**$_c$). O construtor de

objetos **Construtor_o**$_n$ do tipo $\mathbf{T}_{R_n}$ foi gerado pelo procedimento Cria_Construtor _Objeto_Coleção ($\mathbf{T}_{R_n}$, $\mathbf{T}_c$).

Linhas 3-6: Seleciona o objeto $\mathbf{o}_{n-1}$.

Linhas 3-4: Se n=2 (o caminho φ é composto de apenas uma ligação) então $\mathbf{o}_{n-1}$ recebe o objeto r_1 de R_1 tal que r_1 é SE ao objeto o_v, "pai"do objeto o_c (adicionado em **lista_T**$_c$).

Linhas 5-6: Se n>2 (o caminho φ é composto de mais de uma ligação), então $\mathbf{o}_{n-1}$ recebe o objeto r_{n-1} de R_{n-1} tal que r_{n-1} está relacionado com o objeto r_1 de R_1, onde r_1 é SE a o_v, através do caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$. De acordo com a condição 3.2 do Caso 1, o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ é monovalorado, bem como sua inversa. Assim, temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{n-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{n-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{n-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto possível definir o tradutor em tempo de projeto.

Linhas 7-12: Atribui o valor de $\mathbf{o}_n \bullet \ell$.

Linhas 7-8: Se ℓ é uma ligação de referência então o atributo ℓ do objeto o_n recebe a referência do ojeto o_{n-1};

Linhas 9-12: Se ℓ é uma chave estrangeira definida por $\ell = \mathbf{R}_n[\mathbf{a}_1, \ldots, \mathbf{a}_k] \subset \mathbf{R}_{n-1}[\mathbf{d}_1, \ldots, \mathbf{d}_k]$, então é atribuído para os atributos $\mathbf{a}_1$,..., $\mathbf{a}_k$ do objeto $\mathbf{o}_n$ os valores correspondentes dos atributos $\mathbf{d}_1$,..., $\mathbf{d}_k$ do objeto $\mathbf{o}_{n-1}$.

Linha 13: É requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_n$ (instância de $\mathbf{T}_{R_n}$) na tabela $\mathbf{R}_n$.

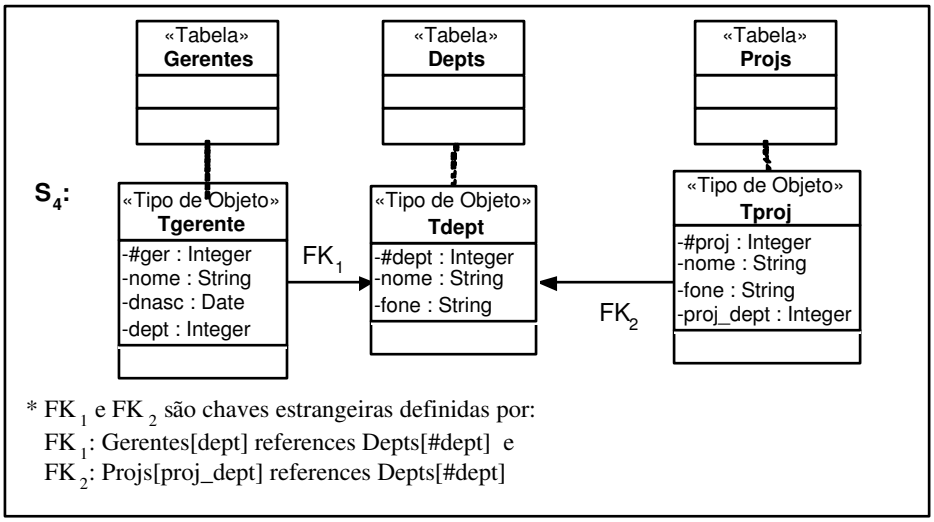

Figura 5.13. *Esquema do banco de dados S_4*

Exemplo 5.3

Considere $\mathbf{S}_4$, o esquema do banco de dados apresentado na Figura 5.13, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v**, apresentado na Figura 5.4. Neste exemplo as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_4$ são semelhantes as especificadas para o Caso 1.1, com exceção da AC do atributo multivalorado **lista_proj_v** dada a seguir:

- Assertivas de Correspondência de Caminhos **lista_proj_v**
  ψ_{11}: **Tgerente_v•lista_proj_v** $\equiv$ **Tgerente•**$\mathrm{FK}_1 \bullet \mathrm{FK}_2{}^{-1}$

A Figura 5.14 apresenta o tradutor '$\mathrm{Gerentes}_v$_Adiciona_$\mathrm{ListaProj}_v4$', gerado pelo algoritmo C1, o qual é usado para traduzir a adição de um objeto na coleção aninhada **lista_proj_v** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualizações na tabela base.

```
Gerentes v_Adiciona_ListaProj v4 (ov: objeto da visão , oc: objeto da coleção aninhada)
{ /* Faz a Tradução da adição de o c em o v.lista_proj_v */

1. on := Tproj (#proj :oc.#proj_v , nome : oc.nome_v, fone : o c.fone_v );
   /* Construtor de objeto de Tproj criado por Cria_Construtor_Objeto_Coleção(Tproj, Tproj_v ) */

2. on-1 = (Selecione r1.FK1 de Gerentes (r1) onde r1.#ger=ov.#ger_v );

3. on.proj_dept = on-1.#dept;

4. Adicione on em Projs ;
 }
```

Figura 5.14. *Tradutor '$\mathrm{Gerentes}_v$_Adiciona_$\mathrm{ListaProj}_v4$'*

De acordo com o algoritmo C1, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{n-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.13, a ligação multivalorada **lista_proj** é virtual, obtida da inversa da ligação monovalorada FK_2, definida por FK_2: **Tproj**→**Tdept**. Desta forma aplica-se o Caso 1.3 do algoritmo C1.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C1.

Linha 1: o_n recebe o construtor de objetos ***Tproj(♯proj:*** o_c**•*proj_v, nome:*** o_c**•*nome_v, fone:*** o_c**•*fone_v)***, o qual cria um objeto do tipo **Tproj** SE ao objeto o_c (adicionado em **lista_proj_v**). Este construtor de objetos foi gerado pelo procedimento Cria_Construtor _Objeto_Coleção (**Tproj**, **Tproj_v**) baseado nas ACs ψ_7, ψ_8 e ψ_9 que especificam formalmente o relacionamento entre **Tproj** e **Tproj_v**.

Linha 2: De ψ_{11} temos que o caminho de derivação de **lista_proj_v** (FK_1•$\text{FK}_2{}^{-1}$) é composto de mais de uma ligação. Assim $\mathbf{o}_{n-1}$ recebe o objeto $\mathbf{r}_1$•FK_1 de **Depts**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r}_1 \equiv \mathbf{o}_v$) sss ($\mathbf{r}_1$•**♯ger**= $\mathbf{o}_v$•**♯ger_v**).

Linha 3: Como FK_1 é uma chave estrangeira , então o atributo **proj_dept** do objeto o_n (instância de **Tproj**) recebe o valor $\mathbf{o}_{n-1}$•**♯dept**.

Linha 4: É gerada a atualização requerida; a qual consiste em adicionar o objeto $\mathbf{o}_n$ na tabela **Projs**.

```
CREATE OR REPLACE TRIGGER Adiciona_Projeto4_Oracle
   INSTEAD OF INSERT ON NESTED TABLE lista_proj_v OF Gerentes_V
 BEGIN
  Insert Into PROJs
    (#proj, nome, fone, proj_dept)
  values ( :new.#proj_v, :new.nome_v, :new.fone_v,
           (Select     g.dept From Gerentes g
            where     g.#ger = :parent.    #ger_v ) );
END;
```

Figura 5.15. *Tradutor 'Adiciona_Projeto4_Oracle'*

Na Figura 5.15, mostramos a título de ilustração o trigger 'Adiciona_Projeto4_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Adiciona_ListaProj$_v$4' gerado para o banco de dados Oracle 8i.

5.1.2. Caso 2: No caminho de derivação de Lista_$\mathbf{T}_c$ a ligação multivalorada não é a última

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ (vide Figura 5.1) com as tabelas bases é de um dos tipos: $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
onde $\mathbf{a}_1$, $\mathbf{a}_2$, ... , $\mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1$, $\mathbf{b}_2$, ... , $\mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência de $\mathbf{lista\_T}_c$ é dada por:
$\mathbf{T}_v \bullet \mathbf{lista\_T}_c \equiv \mathbf{T}_{R_1} \bullet \varphi$, onde:

   3.1 $\varphi = \ell_1 \bullet ... \bullet \ell_{j-1} \bullet ... \bullet \ell_{n-1}$ é um caminho de $\mathbf{T}_{R_1}$ e ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_{R_i} \rightarrow \mathbf{T}_{R_{i+1}}$, para 1≤i≤n-1, como apresentado na Figura 5.16;

   3.2 As ligações ℓ_1, ..., ℓ_{j-2} e ℓ_j, ..., ℓ_{n-1} e suas inversas são monovaloradas, exceto a inversa da ligação ℓ_j que pode ser monovalorada ou multivalorada. Desta forma, temos que cada objeto de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ e vice-versa. Temos ainda que cada objeto de $\mathbf{R}_j$ está relacionado com um único objeto de $\mathbf{R}_n$ através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$. A ligação ℓ_{j-1} é multivalorada virtual, obtida da inversa da ligação monovalorada ℓ: $\mathbf{T}_{R_j} \rightarrow \mathbf{T}_{R_{j-1}}$, onde 1 < j≤n-1. Esta condição define que a ligação multivalorada não é a última ligação do caminho φ;

   3.3 $\mathbf{T}_{R_j}$ é um tipo que representa um relaciomento N:M ou 1:N entre os tipos $\mathbf{T}_{R_{j-1}}$ e $\mathbf{T}_{R_{j+1}}$ [43]. Assim sendo $\mathbf{T}_{R_j}$ possui apenas duas ligações $\ell_j : \mathbf{T}_{R_j} \rightarrow \mathbf{T}_{R_{j+1}}$ e $\ell : \mathbf{T}_{R_j} \rightarrow \mathbf{T}_{R_{j-1}}$. Todos os outros atributos de $\mathbf{T}_{R_j}$ admitem valor nulo.

Neste caso, o pedido de atualização, *"Adicione $\boldsymbol{o}_c$ em $\boldsymbol{o}_v \bullet$ **lista_** $\boldsymbol{T}_c$"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione $\boldsymbol{o}_j$ em $\boldsymbol{R}_j$"*, onde $\mathbf{o}_j$ é uma instância do tipo $\mathbf{T}_{R_j}$. Neste caso $\mathbf{o}_j$ ainda não é uma instância de $\mathbf{R}_j$, mas deve existir um objeto $\mathbf{o}_n$ em R_n tal que $\mathbf{o}_n$ é SE ao objeto $\mathbf{o}_c$. Os valores dos atributos do objeto $\mathbf{o}_j$ são definidos da seguinte forma:

- $\ell = \mathbf{o}_{j-1}$, onde $\mathbf{o}_{j-1}$ é o objeto de $\mathbf{R}_{j-1}$ que está relacionado com o objeto $\mathbf{o}_1$ de $\mathbf{R}_1$ através do caminho $\ell_1 \bullet ... \bullet \ell_{j-2}$;

- $\ell_j = \mathbf{o}_{j+1}$, onde $\mathbf{o}_{j+1}$ é o objeto de $\mathbf{R}_{j+1}$ que está relacionado com o objeto $\mathbf{o}_n$ de $\mathbf{R}_n$ através do caminho $\ell_{j+1} \bullet ... \bullet \ell_{n-1}$;

- os demais atributos de $\mathbf{o}_j$ recebem valores nulos.

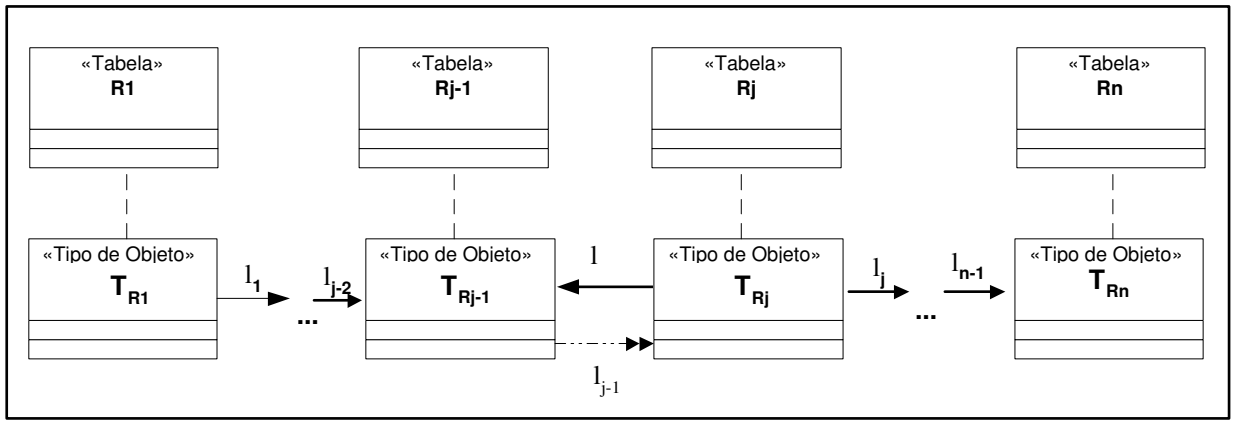

Figura 5.16. *Caminho* $\varphi = \ell_1 \bullet ... \bullet \ell_{j-2} \bullet \ell_{j-1} \bullet ... \bullet \ell_{n-1}$ *do tipo base* $\boldsymbol{T}_{R_1}$

A Figura 5.17 mostra o trecho do Algoritmo C1 que trata o Caso 2. A seguir descrevemos cada linha do algoritmo.

Caso 2:

```
 /* Seleciona o objeto o_{j-1} */
1.   Se n=3 /* o caminho φ é composto de apenas duas ligações */
2.     τ := τ ∪ { ≺ o_{j-1} := (Selecione r_1 de R_1 onde r_1≡o_v) ≻ };
3.   senão /* n>3 */
4.     τ := τ ∪ {≺ o_{j-1}:=(Selecione r_1•ℓ_1ℓ_2•...•ℓ_{j-2} de R_1 onde r_1≡o_v);≻}
       /* o_{j-1} é uma instância de R_{j-1}, tal que o_{j-1} está relacionado com
       r_1 através do caminho ℓ_1•ℓ_2•...  •ℓ_{j-2} */

 /* Seleciona o objeto o_{j+1} */
5.   Se j=(n-1) /* ℓ_j é a última ligação do caminho φ */
6.     τ := τ ∪ {≺ o_{j+1} := (Selecione r_n de R_n onde r_n≡o_c); ≻}
7.   senão /* j<(n-1) */
8.     τ :=τ ∪ {≺ o_{j+1}:= (Selecione r_{j+1} de R_{j+1} onde
9.                          r_{j+1}•ℓ_{j+1}•...•ℓ_{n-1} ≡ o_c); ≻}
       /* o_{j+1} é uma instância de R_{j+1}, tal que o_{j+1} está relacionado com
       r_n de R_n, onde r_n≡o_c, através do caminho ℓ_{j+1}•ℓ_{j+2}•...  •ℓ_{n-1} */

 /* Cria o objeto o_j */
10.  Se ℓ e ℓ_j são atributos de referência então
11.    τ := τ ∪ {≺ o_j := T_j(ℓ:  Ref(o_{j-1}), ℓ_j:  Ref(o_{j+1})); ≻}
12.  Senão /* ℓ e ℓ_j são chaves estrangeiras definidas por
             ℓ:  R_j [c_1,..., c_k]⊂ R_{j-1} [d_1,..., d_k] e
             ℓ_j:  R_j [c_{k+1},..., c_m]⊂ R_{j+1} [e_{k+1},..., e_m]*/
13.    Para i de 1 até k faça:
14.          τ := τ ∪ {≺ v_i = o_{j-1}•d_i; ≻}
15.    Para i de k+1 até m faça:
16.          τ := τ ∪ {≺ v_i = o_{j+1}•e_i; ≻}
17.    τ := τ ∪ {≺ o_j := T_j(c_1:v_1, ..., c_m:v_m); ≻}

 /* Gera a atualização */
18.  τ := τ ∪ { ≺ Adicione o_j em R_j; ≻ }
```

Figura 5.17. *Caso 2 do Algoritmo C1*

Linhas 1-4: Seleciona o objeto $\mathbf{o}_{j-1}$.

Linhas 1-2: Se n=3 (o caminho φ é composto de apenas duas ligações), então $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$, "pai"do objeto $\mathbf{o}_c$ (adicionado em $\mathbf{lista\_T}_c$).

Linhas 3-4: Se n>3 (o caminho φ é composto de mais de duas ligações), então $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_{j-1}$ de $\mathbf{R}_{j-1}$ tal que $\mathbf{r}_{j-1}$ está relacionado com o objeto $\mathbf{r}_1$ de $\mathbf{R}_1$, onde $\mathbf{r}_1$ é SE a $\mathbf{o}_v$, através do caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{j-2}$. De acordo com a condição 3.2 do Caso 2, o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{j-2}$ é monovalorado, assim temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet \ldots \bullet \ell_{j-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{j-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto possível definir o tradutor em tempo de projeto.

Linhas 5-9: Seleciona o objeto $\mathbf{o}_{j+1}$.

Linhas 5-6: Se j=(n-1) (ℓ_j é a última ligação do caminho φ), então $\mathbf{o}_{j+1}$ recebe o objeto $\mathbf{r}_n$ de $\mathbf{R}_n$ tal que $\mathbf{r}_n$ é SE ao objeto $\mathbf{o}_c$, adicionado na coleção $\mathbf{lista\_T}_c$ do objeto o_v da visão $\mathbf{V}$.

Linhas 7-9: Se j<(n-1) (ℓ_j não é a última ligação do caminho φ), então $\mathbf{o}_{j+1}$ recebe o objeto $\mathbf{r}_{j+1}$ de $\mathbf{R}_{j+1}$ tal que $\mathbf{r}_{j+1}$ está relacionado com o objeto $\mathbf{r}_n$ de $\mathbf{R}_n$, onde $\mathbf{r}_n$ é SE a $\mathbf{o}_c$, através do caminho $\ell_{j+1} \bullet \ell_{j+2} \bullet \ldots \bullet \ell_{n-1}$. De acordo com a condição 3.2 do Caso 2, o caminho $\ell_{j+1} \bullet \ell_{j+2} \bullet \ldots \bullet \ell_{n-1}$ é monovalorado, assim temos que um objeto $\mathbf{r}_{j+1}$ de $\mathbf{R}_{j+1}$ está relacionado com um único objeto de $\mathbf{R}_n$ e vice-versa. Caso o caminho $\ell_{j+1} \bullet \ell_{j+2} \bullet \ldots \bullet \ell_{n-1}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{j+1}$ associado a $\mathbf{r}_n$, causando assim ambigüidade a nível de dados e não sendo portanto possível definir o tradutor em tempo de projeto.

Linhas 10-17: Cria o objeto $\mathbf{o}_j$.

Linhas 10-11: Se ℓ e ℓ_j são atributos de referência, então é criado o objeto $\mathbf{o}_j$, cujos valores dos atributos ℓ e ℓ_j são definidos da seguinte forma: ℓ recebe a referência de $\mathbf{o}_{j-1}$ (selecionado nas linhas 1-4) e ℓ_j recebe a referência de $\mathbf{o}_{j+1}$ (selecionado nas linhas 5-9).

Linhas 12-17: Se ℓ e ℓ_j são chaves estrangeiras onde ℓ: $\mathbf{R}_j$ [$\mathbf{c}_1$,..., $\mathbf{c}_k$]$\subset$ $\mathbf{R}_{j-1}$ [$\mathbf{d}_1$,..., $\mathbf{d}_k$] e ℓ_j: $\mathbf{R}_j$ [$\mathbf{c}_{k+1}$,..., $\mathbf{c}_m$]$\subset$ $\mathbf{R}_{j+1}$ [$\mathbf{e}_{k+1}$,..., $\mathbf{e}_m$] , então é criado o objeto $\mathbf{o}_j$, cujos atributos $\mathbf{c}_1$,..., $\mathbf{c}_k$ recebem os valores correspondentes dos atributos $\mathbf{d}_1$,..., $\mathbf{d}_k$ do objeto $\mathbf{o}_{j-1}$ e $\mathbf{c}_{k+1}$,..., $\mathbf{c}_m$ recebem os valores correspondentes dos atributos $\mathbf{e}_{k+1}$,..., $\mathbf{e}_m$ do objeto $\mathbf{o}_{j+1}$.

Linha 18: É requisitada a atualização na tabela base, que consiste em adicionar $\mathbf{o}_j$ (instância de $\mathbf{T}_{R_j}$ criada nas linha 10-17) na tabela $\mathbf{R}_j$.

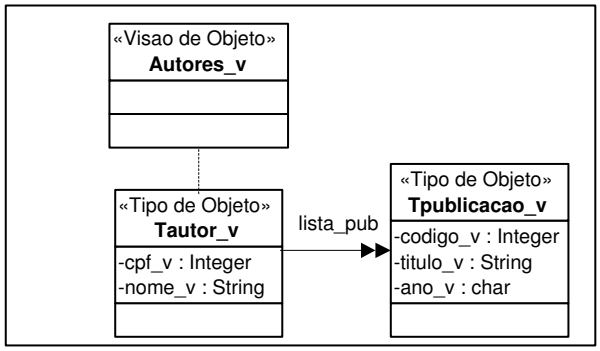

Figura 5.18. *Esquema da visão de objetos* ***AutoresV***

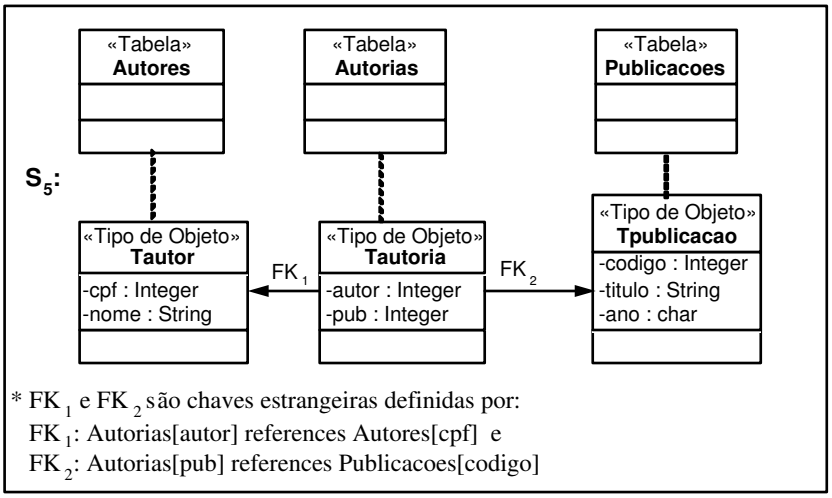

Figura 5.19. *Esquema do banco de dados* S_5

Exemplo 5.4

Suponha o esquema da visão **Autores_v** apresentado na Figura 5.18, e $\mathbf{S}_5$, o esquema do banco de dados apresentado na Figura 5.19. Considere as ACs da visão **Autores_v** com o esquema $\mathbf{S}_5$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Autores_v**≡**Autores**
- Assertivas de Correspondência de Objetos
  ψ_2: [**Tautor_v**, {**cpf_v**}] ≡ [**Tautor**, {**cpf**}]
  ψ_3: [**Tpublicacao_v**, {**codigo_v**}] ≡ [**Tpublicacao**, {**codigo**}]
- Assertivas de Correspondência de Caminhos entre os tipos **Tautor_v** e **Tautor**
  ψ_4: **Tautor_v•lista_pub** ≡ **Tautor**•$\mathrm{FK_1}^{-1}$•$\mathrm{FK_2}$
  ψ_5: **Tautor_v•cpf_v** ≡ **Tautor•cpf**
  ψ_6: **Tautor_v•nome_v** ≡ **Tautor•nome**

- Assertivas de Correspondência de Caminhos entre os tipos **Tpublicacao_v** e **Tpublicacao**
  ψ_7: **Tpublicacao_v•codigo_v ≡ Tpublicacao•codigo**
  ψ_8: **Tpublicacao_v•titulo_v ≡ Tpublicacao•titulo**
  ψ_9: **Tpublicacao_v•ano_v ≡ Tpublicacao•ano**

A Figura 5.20 apresenta o tradutor 'Autores$_v$_Adiciona_Pub$_v$', gerado pelo algoritmo C1, o qual é usado para traduzir a adição de um objeto na coleção aninhada **lista_pub** do objeto $\mathbf{o}_v$ da visão **Autores_v** em atualizações na tabela base.

```
Autores_v_Adiciona_Pub_v (o_v : objeto da visão,  o_c : objeto da coleção aninhada )
{ /* Faz a Tradução da adição de o_c em o_v.lista_pub */

1. o_j-1 := Selecione r_1 de Autores (r_1) onde r_1.cpf = o_v.cpf_v ;

2. o_j+1 := Selecione r_n de Publicacoes (r_n) onde r_n.codigo=o_c.codigo_v ;

3. v_1 := o_j-1.cpf;

4. v_2 := o_j+1.codigo ;

5. o_j := Tautoria ( autor : v_1 , pub : v_2);
   /* Cria um objeto de Tautoria */

6. Adicione o_j em Autorias ;
 }
```

Figura 5.20. *Tradutor 'Autores$_v$_Adiciona_Pub$_v$'*

De acordo com o algoritmo C1, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{j-1} que no nosso exemplo corresponde a ligação **lista_pub**. Como mostrado na Figura 5.19, a ligação multivalorada **lista_pub** é virtual, obtida da inversa da ligação monovalorada FK_1, definida por FK_1: **Tautoria→Tautor**. Desta forma aplica-se o Caso 2 do algoritmo C1.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C1.

Linha 1: De ψ_4 temos que o caminho de derivação de **lista_pub** ($\text{FK}_1{}^{-1}$•FK_2) é composto de apenas duas ligações. Assim $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$ de **Autores**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r}_1$≡$\mathbf{o}_v$) sss ($\mathbf{r}_1$•**cpf**= $\mathbf{o}_v$•**cpf_v**).

Linha 2: De ψ_4 temos que a ligação FK_2 é a última ligação do caminho de derivação de **lista_pub** ($\text{FK}_1{}^{-1}$•FK_2). Assim $\mathbf{o}_{j+1}$ recebe o objeto $\mathbf{r}_n$ de **Publicacoes**, tal que $\mathbf{r}_n$ é SE ao objeto $\mathbf{o}_c$. Note que, de acordo com a AC de objeto ψ_3, ($\mathbf{r}_n$≡$\mathbf{o}_c$) sss ($\mathbf{r}_n$•**codigo**= $\mathbf{o}_c$•**codigo_v**).

Linha 3-5: Como FK_1 e FK_2 são chaves estrangeiras, então é criado o objeto $\mathbf{o}_j$ instância de **Tautoria**, onde o atributo **autor** recebe o valor do atributo **cpf** do objeto $\mathbf{o}_{j-1}$, e o atributo **pub** recebe o valor do atributo **codigo** do objeto $\mathbf{o}_{j+1}$.

Linha 6: É gerada a atualização requerida; a qual consiste em adicionar o objeto $\mathbf{o}_j$ (instância de **Tautoria** criado nas linha 3-5) na tabela **Autorias**.

```
CREATE OR REPLACE TRIGGER Adiciona_Publicacao_Oracle
   INSTEAD OF INSERT ON NESTED TABLE lista_pub OF Autores_V
BEGIN
 Insert Into Autorias
 (autor, pub)
 values ( (Select a  .cpf From Autores a   where  a.cpf =   :parent. cpf_v ),
          (Select p.codigo From Publicacoes p where  p.codigo =  :new.codigo_v ) );
END;
```

Figura 5.21. *Tradutor 'Adiciona_Publicacao_Oracle'*

Na Figura 5.21, mostramos a título de ilustração o trigger 'Adiciona_Publicacao_Oracle' que corresponde ao tradutor 'Autores$_v$_Adiciona_Pub$_v$' gerado para o banco de dados Oracle 8i e supondo que o modelo do banco seja relacional.

5.2. Definindo Tradutores para Operações de Remoção de Coleções Aninhadas

Nesta seção considere a visão **V** cujos objetos são do tipo $\mathbf{T}_v$ e $\mathbf{lista\_T}_c$ um atributo multivalorado (coleção aninhada) de $\mathbf{T}_v$, cujos objetos são do tipo $\mathbf{T}_c$, como mostrado na Figura 5.22. Suponha $\mathbf{o}_v$ um objeto da visão **V**. Considere um pedido de remoção do objeto $\mathbf{o}_c$ do tipo $\mathbf{T}_c$ da coleção $\mathbf{lista\_T}_c$ do objeto $\mathbf{o}_v$. Esse pedido de atualização no nosso formalismo é definido por: "Remova $\mathbf{o}_c$ de $\mathbf{o}_v$.$\mathbf{lista\_T}_c$"

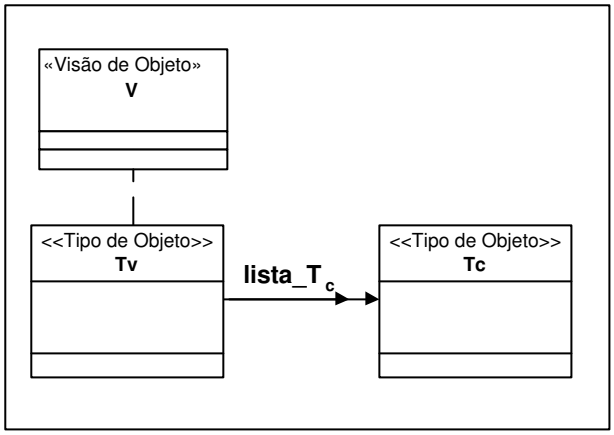

Figura 5.22. *Esquema da Visão* ***V***

O algoritmo C2, apresentado no Apêndice A, gera tradutores para remoção da coleção aninhada $\mathbf{lista\_T}_c$ da visão **V**. Este algoritmo gera tradutores de atualização para a situação descrita a seguir. Mostraremos que nesta situação não existe ambigüidade a nível de dados e a tradução pode ser gerada em tempo de projeto.

As ACs de **V** com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de **V** é de um dos tipos:
   $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1$, $\mathbf{a}_2$, ... , $\mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1$, $\mathbf{b}_2$, ... , $\mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência de $\mathbf{lista\_T}_c$ é dada por:
   $\mathbf{T}_v \bullet \mathbf{lista\_T}_c \equiv \mathbf{T}_{R_1} \bullet \varphi$, onde:

   3.1. $\varphi = \ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{j-1} \bullet$ ... $\bullet \ell_{n-1}$ é um caminho de $\mathbf{T}_{R_1}$ e ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_{R_i} \rightarrow \mathbf{T}_{R_{i+1}}$, para 1≤i≤n-1, como apresentado na Figura 5.23;

   3.2. ℓ_{j-1} é a ligação multivalorada, tal que:

   - Se ℓ_{j-1} não é a última ligação do caminho φ (j<n), então as ligações ℓ_1, ..., ℓ_{j-2} e ℓ_j, ..., ℓ_{n-1} e suas inversas são monovaloradas, exceto a inversa da ligação ℓ_j que pode ser monovalorada ou multivalorada. Desta forma, temos que cada objeto de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{j-2}$ e vice-versa. Temos ainda que cada objeto de $\mathbf{R}_j$ está relacionado com um único objeto de $\mathbf{R}_n$ através do caminho $\ell_j \bullet \ell_{j+1} \bullet$ ... $\bullet \ell_{n-1}$;
   - Se ℓ_{n-1} é a última ligação do caminho φ (j=n), então as ligações ℓ_1, ..., ℓ_{n-2} e suas inversas são ligações monovaloradas, ou seja, cada objeto de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{n-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{n-2}$ e vice-versa.

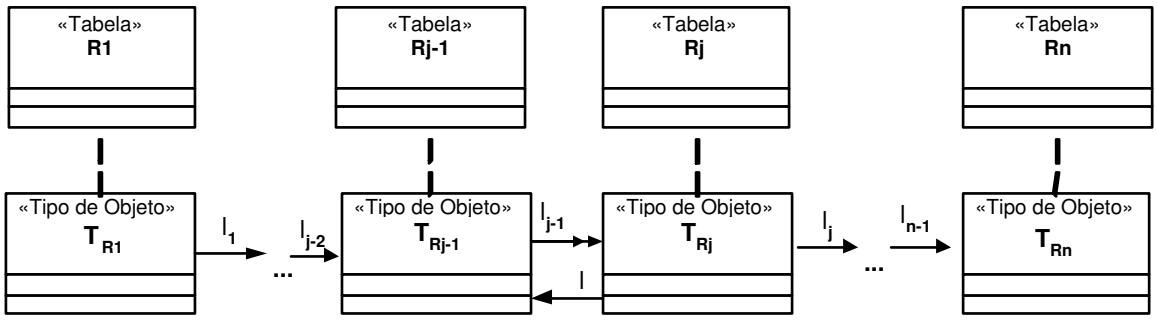

Figura 5.23. *Caminho* $\varphi = \ell_1 \bullet ... \bullet \ell_{j-1} \bullet ... \bullet \ell_{n-1}$ *do tipo base* $\boldsymbol{T}_{R_1}$

De acordo com o Algoritmo C2, existem três casos que precisam ser tratados separadamente:

- O **Caso 1** trata a situação em que a ligação multivalorada ℓ_{j-1} é direta; [2]
- O **Caso 2** trata a situação em que a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação multivalorada;
- O **Caso 3** trata a situação em que a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação monovalorada.

É importante lembrar que os casos 1 e 2 só se aplicam para banco de dados objeto-relacionais uma vez que não existe atributo multivalorado em banco de dados relacionais. A seguir discutimos cada caso do algoritmo.

[2]Como visto no Capítulo 2, ℓ_{j-1} é uma ligação direta se: (i) ℓ_{j-1} é uma ligação de atributo de valor; ou (ii) ℓ_{j-1} é uma ligação de atributo de referência; ou (iii) ℓ_{j-1} é uma ligação de chave estrangeira.

5.2.1. Caso 1 - a ligação multivalorada ℓ_{j-1} é direta

Neste caso, o pedido de atualização, *"Remova* $\boldsymbol{o_c}$ *de* $\boldsymbol{o_v}\bullet\boldsymbol{lista\_T_c}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* $\boldsymbol{o_j}$ *de* $\boldsymbol{o_{j-1}}\bullet\ell_{j-1}$*"*, onde:

- $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, que está relacionada com a instância $\mathbf{r}_1$ de $\mathbf{R}_1$ ($\mathbf{r}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1\bullet\ell_2\bullet$ ... $\bullet\ell_{j-2}$ ($\mathbf{r}_1\bullet\ell_1\bullet...\bullet\ell_{j-2} = o_{j-1}$).
- $\mathbf{o}_j$ é uma instância da coleção ℓ_{j-1} do objeto $\mathbf{o}_{j-1}$ que está relacionada com a instância $\mathbf{r}_n$ de $\mathbf{R}_n$ ($\mathbf{r}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j\bullet\ell_{j+1}\bullet...\bullet\ell_{n-1}$ ($o_j\bullet\ell_j\bullet\ell_{j+1}\bullet...\bullet\ell_{n-1} = \mathbf{r}_n$);

A Figura 5.24 mostra o trecho do Algoritmo C2 que trata o Caso 1. A seguir descrevemos cada linha do algoritmo.

```
-----------------------------------------------------------------------
/* Seleciona o objeto o_{j-1} */
1. Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */
2.    τ := τ ∪ {≺ o_{j-1} := (Selecione r_1 de R_1 onde r_1≡o_v); ≻}
3. senão /* j>2 */
4.    τ := τ ∪ {≺ o_{j-1} := (Selecione r_1•ℓ_1• ℓ_2•...•ℓ_{j-2} de R_1
5.                            onde r_1≡o_v); ≻}
     /* o_{j-1} é uma instância de R_{j-1}, tal que o_{j-1} está relacionado com a
     instância r_1 de R_1 (r_1≡o_v) através do caminho ℓ_1•ℓ_2•...  •ℓ_{j-2} */

/* Gera a atualização */
6. Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */
7.    τ := τ ∪ {≺ Remova o_j de o_{j-1}•ℓ_{j-1} onde o_j≡o_c; ≻}
8. senão /* j<n */
9.    τ:= τ ∪ {≺ Remova o_j de o_{j-1}•ℓ_{j-1} onde o_j•ℓ_j•...•ℓ_{n-1}≡o_c;≻}
     /* o_j é uma instância de o_{j-1}•ℓ_{j-1}, tal que o_j está relacionado com
     o objeto r_n de R_n (r_n ≡ o_c) através do caminho ℓ_j•ℓ_{j+1}•...  •ℓ_{n-1} */
-----------------------------------------------------------------------
```

Figura 5.24. *Caso 1 do Algoritmo C2*

Linhas 1-5: Selecina o objeto $\mathbf{o}_{j-1}$.

Linhas 1-2: Se j=2 (a ligação multivalorada ℓ_{j-1} é a primeira ligação do caminho φ), então $\mathbf{o}_{j-1}$ recebe o objeto r_1 de R_1 tal que r_1 é SE ao objeto o_v da visão $\mathbf{V}$.

Linhas 3-5: Se j>2 (a ligação multivalorada ℓ_{j-1} não é a primeira ligação do caminho φ), então $\mathbf{o}_{j-1}$ recebe o objeto r_{j-1} de R_{j-1} tal que r_{j-1} está relacionado

com o objeto r_1 de R_1, onde r_1 é SE a o_v, através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$. De acordo com a condição 3.2, o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ é monovalorado, assim temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{j-2}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{j-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto viável definir o tradutor em tempo de projeto.

Linhas 6-9: Gera a atualização na tabela base.

Linhas 6-7: Se j=n (ℓ_{j-1} é a última ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_j$ da coleção aninhada ℓ_{j-1} do objeto $\mathbf{o}_{j-1}$ da tabela $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_j$ é SE ao objeto o_c, removido da coleção $\mathbf{lista\_T}_c$ do objeto o_v da visão $\mathbf{V}$.

Linhas 8-9: Se j<n (ℓ_{j-1} não é a última ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_j$ da coleção aninhada ℓ_{j-1} do objeto $\mathbf{o}_{j-1}$ da tabela $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_j$ está relacionado com o objeto $\mathbf{r}_n$ de $\mathbf{R}_n$, onde $\mathbf{r}_n$ é SE a $\mathbf{o}_c$, através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$.

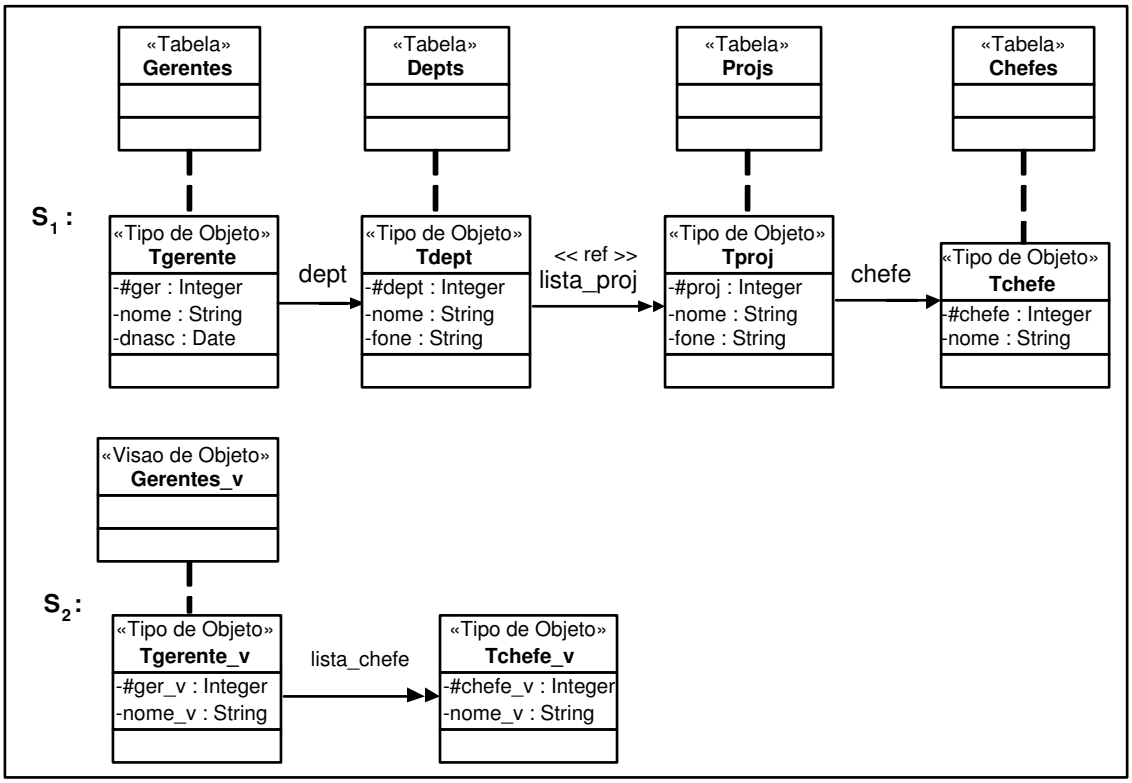

Figura 5.25. *Esquema do banco de dados S_1 e esquema da visão de objetos S_2*

Exemplo 5.5

Sejam $\mathbf{S}_1$, o esquema do banco de dados, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v** apresentados na Figura 5.25. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_1$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v**≡**Gerentes**
- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
  ψ_3: [**Tchefe_v**, {♯**chefe_v**}] ≡ [**Tchefe**, {♯**chefe**}]
- Assertivas de Correspondência de Caminhos entre os tipos **Tgerente_v** e **Tgerente**
  ψ_4: **Tgerente_v•♯ger_v** ≡ **Tgerente•♯ger**
  ψ_5: **Tgerente_v•nome_v** ≡ **Tgerente•nome**
  ψ_6: **Tgerente_v•lista_chefe** ≡ **Tgerente•dept•lista_proj•chefe**
- Assertivas de Correspondência de Caminhos entre os tipos **Tchefe_v** e **Tchefe**
  ψ_7: **Tchefe_v•♯chefe_v** ≡ **Tchefe•♯chefe**
  ψ_8: **Tchefe_v•nome_v** ≡ **Tchefe•nome**

A Figura 5.26 apresenta o tradutor 'Gerentes$_v$_Remove_Chefe1', gerado pelo algoritmo C2, o qual é usado para traduzir a remoção de um objeto da coleção aninhada **lista_chefe** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualização na tabela base.

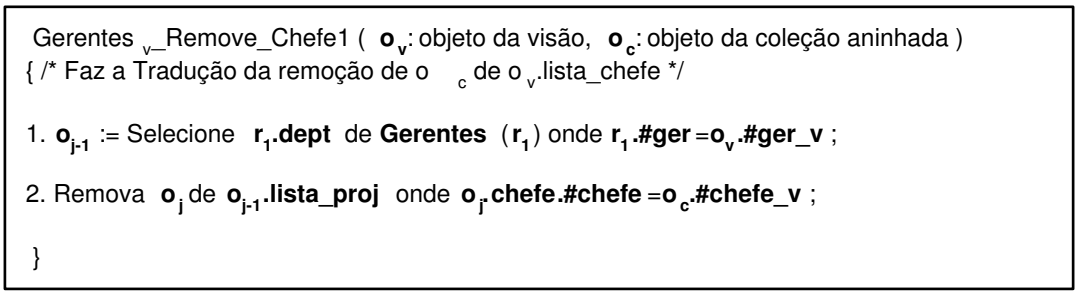

Figura 5.26. *Tradutor 'Gerentes$_v$_Remove_Chefe1'*

De acordo com o algoritmo C2, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{j-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.25, a ligação multivalorada **lista_proj** é um atributo de **Tdept**, logo é uma ligação direta. Desta forma aplica-se o Caso 1 do algoritmo C2.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C2.

Linha 1: De ψ_6 temos que a ligação **lista_proj** não é a primeira ligação do caminho de derivação de **lista_chefe** (**dept•lista_proj•chefe**). Assim $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$**•dept**

de **Depts**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r}_1 \equiv \mathbf{o}_v$) sss ($\mathbf{r}_1 \bullet \sharp\mathbf{ger} = \mathbf{o}_v \bullet \sharp\mathbf{ger\_v}$).

Linha 2: De ψ_6 temos que a ligação **lista_proj** não é a última ligação do caminho de derivação de **lista_chefe** (**dept•lista_proj•chefe**). Assim é gerada a atualização requerida; a qual consiste em remover o objeto $\mathbf{o}_j$ (instância da coleção **lista_proj** do objeto $\mathbf{o}_{j-1}$ selecionado na linha 1), tal que $\mathbf{o}_j$ está relacionado com objeto $\mathbf{r}_n$ de **Chefes**, onde $\mathbf{r}_n$ é SE ao objeto $\mathbf{o}_c$, através do caminho **chefe**. Note que, de acordo com a AC de objeto ψ_3, ($\mathbf{r}_n \equiv \mathbf{o}_c$) sss ($\mathbf{r}_n \bullet \sharp\mathbf{chefe} = \mathbf{o}_v \bullet \sharp\mathbf{chefe\_v}$).

```
CREATE OR REPLACE TRIGGER Remove_Chefe1_Oracle
   INSTEAD OF DELETE ON NESTED TABLE lista_chefe OF Gerentes_v
 BEGIN
      Delete From   TABLE (Select g.dept.lista _proj From Gerentes g
                                Where g.#ger     = :PARENT. #ger_ v) p
      Where p.chefe.#chefe= :old.    #chefe_v;".
END;
```

Figura 5.27. *Tradutor 'Remove_Chefe1_Oracle'*

Na Figura 5.27, mostramos a título de ilustração o trigger 'Remove_Chefe1_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Remove_Chefe1' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma remoção na coleção **lista_chefe** do objeto $\mathbf{o}_v$, este trigger é disparado. O valor *:old* refere-se ao objeto deletado, no caso $\mathbf{o}_c$. O valor *:parent* refere-se ao objeto "pai"do objeto deletado, no caso $\mathbf{o}_v$.

5.2.2. Caso 2 - a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação multivalorada.

Neste caso, o pedido de atualização, *"Remova* $\boldsymbol{o_c}$ *de* $\boldsymbol{o_v}\bullet\boldsymbol{lista\_T_c}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* $\boldsymbol{o_{j-1}}$ *de* $\boldsymbol{o_j}\bullet\ell$*"*, onde:

- $\mathbf{o}_j$ é uma instância de R_j que está relacionada com a instância $\mathbf{r}_n$ de $\mathbf{R}_n$ ($\mathbf{r}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j\bullet\ell_{j+1}\bullet...\bullet\ell_{n-1}$ ($o_j\bullet\ell_j\bullet...\bullet\ell_{n-1}= r_n$).
- $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{o}_j\bullet\ell$ que está relacionada com a instância $\mathbf{r}_1$ de $\mathbf{R}_1$ ($\mathbf{r}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1\bullet\ell_2\bullet$ ... $\bullet\ell_{j-2}$ ($\mathbf{r}_1\bullet\ell_1\bullet...\bullet\ell_{j-2}= o_{j-1}$);

A Figura 5.28 mostra o trecho do Algoritmo C2 que trata o Caso 2. A seguir descrevemos cada linha do algoritmo.

```
/* Seleciona o objeto o_j */
1. Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */
2.    τ := τ ∪ {≺ o_j := (Selecione r_n de R_n onde r_n≡o_c); ≻}
3. senão /* j<n */
4.    τ := τ ∪ {≺ o_j :=(Selecione r_j de R_j onde r_j•ℓ_j•...•ℓ_{n-1}≡o_c);≻}
     /* o_j é uma instância de R_j, tal que o_j está relacionado com a
     instância r_n de R_n (r_n ≡ o_c) através do caminho ℓ_j•ℓ_{j+1}•...  •ℓ_{n-1} */

/* Gera a atualização */
5. Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */
6.    τ := τ ∪ {≺ Remova o_{j-1} de o_j•ℓ onde o_{j-1}≡o_v; ≻}
7. senão /* j>2 */
8.    τ := τ ∪ {≺ o_1 := (Selecione r_1 de R_1 onde r_1≡o_v);≻}
9.    τ := τ ∪ {≺ Remova o_{j-1} de o_j•ℓ onde o_{j-1}≡ o_1•ℓ_1•...•ℓ_{j-2};≻}
     /* o_{j-1} é uma instância de o_j•ℓ, tal que o_{j-1} está relacionado com a
     instância o_1 de R_1 (o_1 ≡ o_v) através do caminho ℓ_1•ℓ_2•...  •ℓ_{j-2} */
```

Figura 5.28. *Caso 2 do Algoritmo C2*

Linhas 1-4: Seleciona o objeto $\mathbf{o}_j$.

Linhas 1-2: Se j=n (ℓ_{j-1} é a última ligação do caminho φ), então $\mathbf{o}_j$ recebe o objeto r_n de R_n tal que r_n é SE ao objeto o_c, removido da coleção $\mathbf{lista\_T}_c$ do objeto o_v da visão $\mathbf{V}$.

Linhas 3-4: Se j<n (ℓ_{j-1} não é a última ligação do caminho φ), então $\mathbf{o}_j$ recebe o objeto r_j de R_j tal que r_j está relacionado com o objeto r_n de R_n, onde r_n é SE a o_c, através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$. De acordo com a condição 3.2, o caminho $\ell_j \bullet \ell_{j+1} \bullet \mathbf{...} \bullet \ell_{n-1}$ é monovalorado, assim temos que um objeto $\mathbf{r}_j$ de $\mathbf{R}_j$ está relacionado com um único objeto de $\mathbf{R}_n$ através do caminho $\ell_j \bullet \ell_{j+1} \bullet$ ... $\bullet \ell_{n-1}$.

Linhas 5-9: Gera a atualização na tabela base.

Linhas 5-6: Se j=2 (a ligação multivalorada ℓ_{j-1} é a primeira ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_{j-1}$ da coleção aninhada ℓ do objeto $\mathbf{o}_j$ da tabela $\mathbf{R}_j$, tal que $\mathbf{o}_{j-1}$ é SE ao objeto $\mathbf{o}_v$ da visão $\mathbf{V}$.

Linhas 7-9: Se j>2 (a ligação multivalorada ℓ_{j-1} não é a primeira ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_{j-1}$ da coleção aninhada ℓ do objeto $\mathbf{o}_j$ da tabela $\mathbf{R}_j$, tal que $\mathbf{o}_{j-1}$ está relacionado com o objeto $\mathbf{o}_1$ de R_1, onde $\mathbf{o}_1$ é SE a $\mathbf{o}_v$, através do caminho $\ell_1 \bullet \ell_2 \bullet \mathbf{...} \bullet \ell_{j-2}$. De acordo com a condição 3.2, o caminho $\ell_1 \bullet \ell_2 \bullet \mathbf{...} \bullet \ell_{j-2}$ é monovalorado, assim temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{j-2}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet \mathbf{...} \bullet \ell_{j-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{j-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto viável definir o tradutor em tempo de projeto.

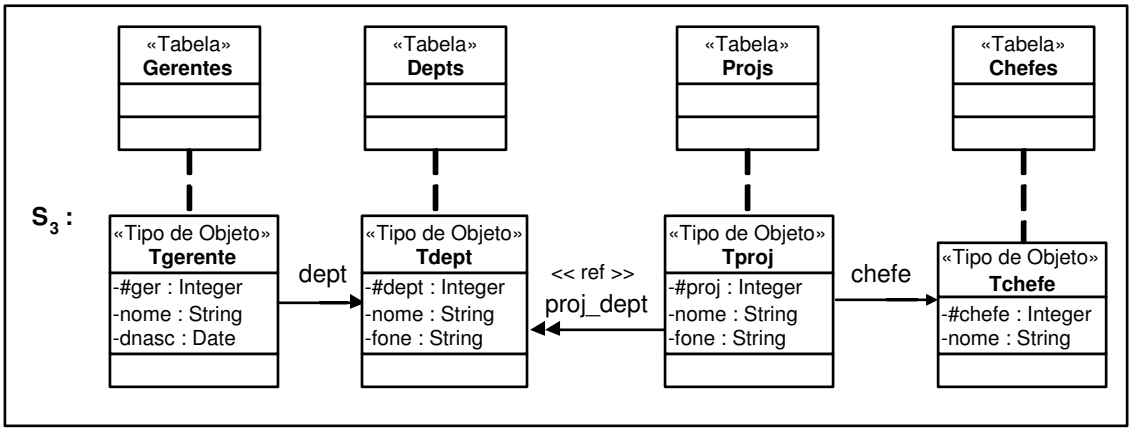

Figura 5.29. *Esquema do banco de dados S_3*

Exemplo 5.6

Considere $\mathbf{S}_3$, o esquema do banco de dados apresentado na Figura 5.29, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v** apresentado na Figura 5.25. Neste exemplo, as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_3$ são semelhantes as especificadas para o Caso 1, com exceção da AC do atributo multivalorado **lista_chefe** dada a seguir:

- Assertiva de Correspondência do Atributo **lista_chefe**
  ψ_9: **Tgerente_v•lista_chefe** ≡ **Tgerente•dept•proj_dept**$^{-1}$**•chefe**

A Figura 5.30 apresenta o tradutor 'Gerentes$_v$_Remove_Chefe2', gerado pelo algoritmo C2, o qual é usado para traduzir a remoção de um objeto da coleção aninhada **lista_chefe** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualização na tabela base.

```
Gerentes v_Remove_Chefe2 (o v: objeto da visão , o c: objeto da coleção aninhada )
{ /* Faz a Tradução da remoção de o c de o v.lista_chefe */

1. o j := Selecione r j de Projs (r j) onde r j.chefe.#chefe=o c.#chefe_v;

2. o 1 := Selecione r 1 de Gerentes (r 1) onde r 1.#ger=o v.#ger_v;

3. Remova o j-1 de o j.proj_dept onde o j-1 = o 1.dept;
 }
```

Figura 5.30. *Tradutor 'Gerentes$_v$_Remove_Chefe2'*

De acordo com o algoritmo C2, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{j-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.29, a ligação multivalorada **lista_proj** é virtual, obtida da inversa da ligação multivalorada **proj_dept**, definida por **proj_dept**: **Tproj**→**Tdept**. Desta forma aplica-se o Caso 2 do algoritmo C2.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C2.

Linha 1: De ψ_9 temos que a ligação **lista_proj** não é a última ligação do caminho de derivação de **lista_chefe** (**dept•proj_dept**$^{-1}$**•chefe**). Assim $\mathbf{o}_j$ recebe o objeto $\mathbf{r}_j$ de **Projs**, tal que $\mathbf{r}_j$ está relacionado com o objeto $\mathbf{r}_n$ de **Chefes**, onde $\mathbf{r}_n$ é SE ao objeto $\mathbf{o}_c$, através do caminho **chefe**. Note que, de acordo com a AC de objeto ψ_3, ($\mathbf{r}_n$≡$\mathbf{o}_c$) sss ($\mathbf{r}_n$•♯**chefe**= $\mathbf{o}_v$•♯**chefe_v**).

Linhas 2-3: De ψ_9 temos que a ligação **lista_proj** não é a primeira ligação do caminho de derivação de **lista_chefe** (**dept•proj_dept**$^{-1}$**•chefe**). Assim é gerada a atualização requerida; a qual consiste em remover o objeto $\mathbf{o}_{j-1}$ (instância da coleção **proj_dept** do objeto $\mathbf{o}_j$ selecionado na linha 1), tal que $\mathbf{o}_{j-1}$ está relacionado com objeto

$\mathbf{o}_1$ de **Gerentes**, onde $\mathbf{o}_1$ é SE ao objeto $\mathbf{o}_v$, através do caminho **dept**. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{o}_1 \equiv \mathbf{o}_v$) sss ($\mathbf{o}_1 \bullet \sharp \mathbf{ger} = \mathbf{o}_v \bullet \sharp \mathbf{ger\_v}$).

```
CREATE OR REPLACE TRIGGER Remove_Chefe2_Oracle
   INSTEAD OF DELETE ON NESTED TABLE lista_chefe OF Gerentes_v
 BEGIN
     Delete From TABLE(Select p.proj_dept From PROJs p
                              Where  p.chefe.#chefe= :old.#chefe_v) p2
     Where p2 = (Select g.dept From Gerentes g
                   Where g.#ger= :Parent.#ger_v)
END;
```

Figura 5.31. *Tradutor 'Remove_Chefe2_Oracle'*

Na Figura 5.31, mostramos a título de ilustração o trigger 'Remove_Chefe2_Oracle' que corresponde ao tradutor 'Gerentes_v_Remove_Chefe2' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma remoção na coleção **lista_chefe** do objeto $\mathbf{o}_v$, este trigger é disparado.

5.2.3. Caso 3 - a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa da ligação ℓ, onde ℓ é uma ligação monovalorada.

Neste caso, o pedido de atualização, *"Remova* $\boldsymbol{o_c}$ *de* $\boldsymbol{o_v}\bullet\boldsymbol{lista\_T_c}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* $\boldsymbol{o_j}$ *de* $\boldsymbol{R_j}$ *onde* $\boldsymbol{o_j}\bullet\ell{=}\boldsymbol{o_{j-1}}$*"*, onde:

- $\mathbf{o}_{j-1}$ é uma instância de R_{j-1} que está relacionada com a instância $\mathbf{r}_1$ de $\mathbf{R}_1$ ($\mathbf{r}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1\bullet\ell_2\bullet...\bullet\ell_{j-2}$ ($\mathbf{r}_1\bullet\ell_1\bullet...\bullet\ell_{j-2}= o_{j-1}$).
- $\mathbf{o}_j$ é uma instância de R_j que está relacionada com a instância $\mathbf{r}_n$ de $\mathbf{R}_n$ ($\mathbf{r}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j\bullet\ell_{j+1}\bullet...\bullet\ell_{n-1}$ ($o_j\bullet\ell_j\bullet...\bullet\ell_{n-1}= \mathbf{r}_n$) e $\mathbf{o}_j\bullet\ell{=}\mathbf{o}_{j-1}$;

A Figura 5.32 mostra o trecho do Algoritmo C2 que trata o Caso 3. A seguir descrevemos cada linha do algoritmo.

```
/* Seleciona o objeto o_{j-1} */
1. Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */
2.    τ := τ ∪ {≺ o_{j-1} := (Selecione r_1 de R_1 onde r_1≡o_v); ≻}
3. senão /* j>2 */
4.    τ := τ ∪ {≺ o_{j-1} := (Selecione r_1•ℓ_1• ℓ_2•...•ℓ_{j-2} de R_1
5.                            onde r_1≡o_v); ≻}
     /* o_{j-1} é uma instância de R_{j-1}, tal que o_{j-1} está relacionado com r_1
     através do caminho ℓ_1•ℓ_2•...  •ℓ_{j-2} */

/* Gera a atualização */
6. Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */
7.    τ := τ ∪ {≺ Remova o_j de R_j onde o_j•ℓ=o_{j-1} e o_j≡o_c; ≻}
8. senão /* j<n */
9.    τ := τ ∪ {≺ Remova o_j de R_j onde o_j•ℓ=o_{j-1} e o_j•ℓ_j•...•ℓ_{n-1}≡o_c;≻}
     /* o_j é uma instância de R_j, tal que o_j está relacionado com o objeto
     o_n de R_n (o_n ≡ o_c) através do caminho ℓ_j•ℓ_{j+1}•...  •ℓ_{n-1} e o_j•ℓ=o_{j-1}*/
```

Figura 5.32. *Caso 3 do Algoritmo C2*

Linhas 1-5: Selecina o objeto $\mathbf{o}_{j-1}$.

Linhas 1-2: Se j=2 (a ligação multivalorada ℓ_{j-1} é a primeira ligação do caminho φ), então $\mathbf{o}_{j-1}$ recebe o objeto r_1 de R_1 tal que r_1 é SE ao objeto o_v da visão **V**.

Linhas 3-5: Se j>2 (a ligação multivalorada ℓ_{j-1} não é a primeira ligação do caminho φ), então $\mathbf{o}_{j-1}$ recebe o objeto r_{j-1} de R_{j-1} tal que r_{j-1} está relacionado com o objeto r_1 de R_1, onde r_1 é SE a o_v, através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$. De acordo com a condição 3.2, o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ é monovalorado, assim temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ através do caminho $\ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{j-2}$ e vice-versa. Caso o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ fosse multivalorado, então poderia existir mais de um objeto em $\mathbf{R}_{j-1}$ associado a $\mathbf{r}_1$, causando assim ambigüidade a nível de dados e não sendo portanto viável definir o tradutor em tempo de projeto.

Linhas 6-9: Gera a atualização na tabela base.

Linhas 6-7: Se j=n (ℓ_{j-1} é a última ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_j$ de $\mathbf{R}_j$ onde $\mathbf{o}_j \bullet \ell$ é igual ao objeto $\mathbf{o}_{j-1}$ da tabela $\mathbf{R}_{j-1}$ e $\mathbf{o}_j$ é SE ao objeto o_c, removido da coleção $\mathbf{lista\_T}_c$ do objeto o_v da visão $\mathbf{V}$.

Linhas 8-9: Se j<n (ℓ_{j-1} não é a última ligação do caminho φ), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{o}_j$ de $\mathbf{R}_j$ onde $\mathbf{o}_j \bullet \ell$ é igual ao objeto $\mathbf{o}_{j-1}$ da tabela $\mathbf{R}_{j-1}$ e $\mathbf{o}_j$ está relacionado com o objeto r_n de R_n, onde r_n é SE a o_c, através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$. De acordo com a condição 3.2, o caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$ é monovalorado, assim temos que um objeto $\mathbf{o}_j$ de $\mathbf{R}_j$ está relacionado com um único objeto de $\mathbf{R}_n$ através do caminho $\ell_j \bullet \ell_{j+1} \bullet$ ... $\bullet \ell_{n-1}$.

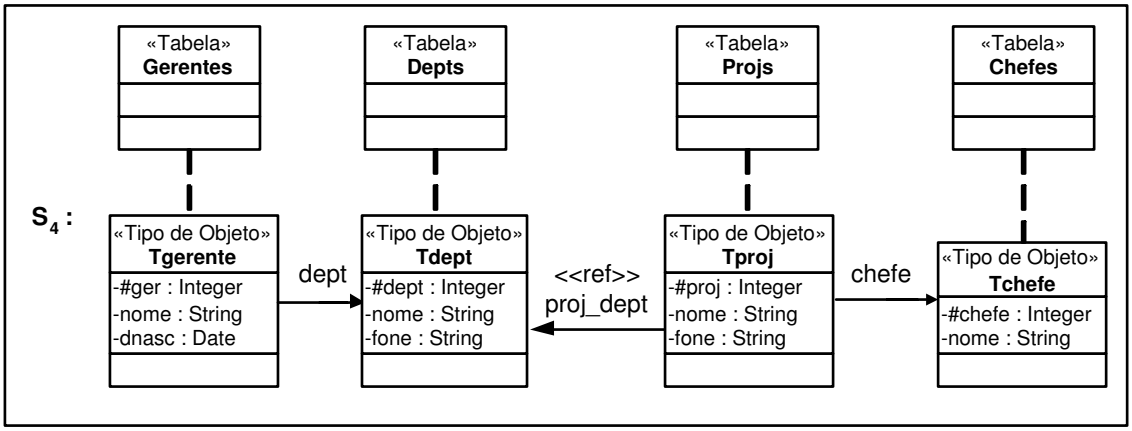

Figura 5.33. *Esquema do banco de dados S_4*

Exemplo 5.7

Considere $\mathbf{S}_4$, o esquema do banco de dados apresentado na Figura 5.33, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v** apresentado na Figura 5.25. Neste exemplo as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_4$ são semelhantes as especificadas para o Caso 1, com exceção da AC do atributo multivalorado **lista_chefe** dada a seguir:

- Assertiva de Correspondência do Atributo **lista_chefe**
  ψ_{10}: **Tgerente_v•lista_chefe** ≡ **Tgerente•dept•proj_dept**$^{-1}$**•chefe**

A Figura 5.34 apresenta o tradutor 'Gerentes$_v$_Remove_Chefe3', gerado pelo algoritmo C2, o qual é usado para traduzir a remoção de um objeto da coleção aninhada **lista_chefe** do objeto $\mathbf{o}_v$ da visão **Gerentes_v** em atualização na tabela base.

```
Gerentes v_Remove_Chefe3 ( o v: objeto da visão,  o c: objeto da coleção aninhada )
{ /* Faz a Tradução da remoção de o c de o v.lista_chefe */

1. o j-1 := Selecione  r 1.dept  de Gerentes  (r 1) onde r 1.#ger =o v.#ger_v ;

2. Remova  o j de Projs  onde o j.proj_dept =o j-1 e o j.chefe.#chefe =o c.#chefe_v ;
}
```

Figura 5.34. *Tradutor 'Gerentes$_v$_Remove_Chefe3'*

De acordo com o algoritmo C2, inicialmente analisamos as caracteristicas da ligação multivalorada ℓ_{j-1} que no nosso exemplo corresponde a ligação **lista_proj**. Como mostrado na Figura 5.33, a ligação multivalorada **lista_proj** é virtual, obtida da inversa da ligação monovalorada **proj_dept**, definida por **proj_dept**: **Tproj**→**Tdept**. Desta forma aplica-se o Caso 3 do algoritmo C2.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo C2.

Linha 1: De ψ_{10} temos que a ligação **lista_proj** não é a primeira ligação do caminho de derivação de **lista_chefe** (**dept•proj_dept**$^{-1}$**•chefe**). Assim $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$**•dept** de **Depts**, tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r}_1$≡$\mathbf{o}_v$) sss ($\mathbf{r}_1$**•♯ger**= $\mathbf{o}_v$**•♯ger_v**).

Linha 2: De ψ_{10} temos que a ligação **lista_proj** não é a última ligação do caminho de derivação de **lista_chefe** (**dept•proj_dept**$^{-1}$**•chefe**). Assim é gerada a atualização requerida; a qual consiste em remover o objeto $\mathbf{o}_j$ da tabela **Projs** onde $\mathbf{o}_j$**•proj_dept** é igual ao objeto $\mathbf{o}_{j-1}$ (instância de **Depts** selecionada na linha 1) e $\mathbf{o}_j$ está relacionado com o objeto r_n de **Chefes**, onde r_n é SE a o_c, através do caminho **chefe**. Note que, de acordo com a AC de objeto ψ_3, ($\mathbf{r}_n$≡$\mathbf{o}_c$) sss ($\mathbf{r}_n$**•♯chefe**= $\mathbf{o}_v$**•♯chefe_v**).

```
CREATE OR REPLACE TRIGGER Remove_Chefe3_Oracle
   INSTEAD OF DELETE ON NESTED TABLE lista_chefe OF Gerentes_v
 BEGIN
     Delete From PROJs  p
     Where p.proj_dept = (Select g.dept From Gerentes g Where g.#ger = :parent.#ger_v )
              and
              p.chefe.#chefe = :old.#chefe_v;
END;
```

Figura 5.35. *Tradutor 'Remove_Chefe3_Oracle'*

Na Figura 5.35, mostramos a título de ilustração o trigger 'Remove_Chefe3_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Remove_Chefe3' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma remoção na coleção **lista_chefe** do objeto $\mathbf{o}_v$, este trigger é disparado.

5.3. Definindo Tradutores para Operações de Modificação de Atributos Monovalorados de Visões

Nesta seção, considere a visão **V** cujos objetos são do tipo $\mathbf{T}_v$ como mostrado na Figura 5.36. Suponha $\mathbf{o}_v$ um objeto da visão **V** que foi selecionado pela cláusula "where"de um comando de "update"do SQL. Note que um comando de atualização em SQL pode alterar várias tuplas. Desta forma o tradutor para modificação de atributos monovalorados de visões é chamado para cada tupla afetada, de forma similar aos "Triggers For Each Row"do Oracle [24].

Considere um pedido de modificação do atributo monovalorado de valor $\mathbf{a}_v$ do objeto $\mathbf{o}_v$. Esse pedido de atualização no nosso formalismo é definido por:
"Atribua $\mathbf{o}_v \bullet \mathbf{a}_v$='v'",
onde lê-se: atribua o valor 'v' para o atributo $\mathbf{a}_v$ do objeto $\mathbf{o}_v$. O comando Atribua corresponde a cláusula "SET"do comando Update do SQL.

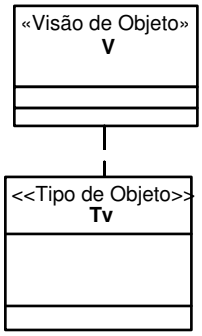

Figura 5.36. *Visão* $\boldsymbol{V}$ *do tipo* $\boldsymbol{T}_v$

O algoritmo V3, apresentado no Apêndice A, gera tradutores para modificação do atributo monovalorado $\mathbf{a}_v$ de $\mathbf{T}_v$. Este algoritmo gera tradutores de atualização para os dois casos apresentados a seguir:

- O **Caso 1** trata a modificação de atributos monovalorados de valor, cuja ACC é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \mathbf{a}$, onde $\mathbf{a}$ é um atributo de $\mathbf{T}_{R_1}$, o qual é o tipo da tabela base.
- O **Caso 2** trata a modificação de atributos monovalorados de valor, cuja ACC é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$, onde φ é um caminho de $\mathbf{T}_{R_1}$, o qual é o tipo da tabela base.

Mostraremos que nestes casos não existe ambigüidade a nível de dados e a tradução pode ser gerada em tempo de projeto.

5.3.1. Caso 1 - A AC de caminho é do tipo $\mathbf{T}_v\bullet\mathbf{a}_v \equiv \mathbf{T}_{R_1}\bullet\mathbf{a}$

As ACs de **V** com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de **V** com as tabelas bases é de um dos tipos:
   $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência do atributo monovalorado $\mathbf{a}_v$ á dada por:
   $\mathbf{T}_v\bullet\mathbf{a}_v \equiv \mathbf{T}_{R_1}\bullet\mathbf{a}$, onde **a** é um atributo de $\mathbf{T}_{R_1}$. Na nossa abordagem, não permitiremos a modificação de atributos que sejam identificadores, portanto **a** não é um identificador de $\mathbf{R}_1$ [43].

Neste caso, o pedido de atualização, *"Atribua* $\boldsymbol{o}_v\bullet\boldsymbol{a}_v$*= 'v'"*, deve ser traduzido na seguinte atualização no banco de dados: *"Atribua* $\boldsymbol{o}_1\bullet\boldsymbol{a}=\boldsymbol{o}_v\bullet\boldsymbol{a}_v$*"*, onde $\mathbf{o}_1$ é uma instância de $\mathbf{R}_1$ semanticamente equivalente (SE) a $\mathbf{o}_v$. A Figura 5.37 mostra o trecho do Algoritmo V3 que trata o Caso 1. A seguir descrevemos cada linha do algoritmo.

```
-----------------------------------------------------------------------

Caso 1: A ACC é do tipo T_v•a_v ≡ T_R1•a

1.        τ := τ ∪ { ≺ o_1 := (Selecione r_1 de R_1 onde r_1≡o_v);
2.                   Atribua o_1•a= o_v•a_v; ≻ }

-----------------------------------------------------------------------
```

Figura 5.37. *Caso 1 do Algoritmo V3*

Linha 1: $\mathbf{o}_1$ recebe o objeto $\mathbf{r}_1$ de $\mathbf{R}_1$, tal que $\mathbf{r}_1$ é semanticamente equivalente ao objeto $\mathbf{o}_v$ de **V**. Note que, de acordo com a AC dos objetos de $\mathbf{R}_1$ com os objetos de**V**, $\mathbf{r}_1\equiv\mathbf{o}_v$ sss $\mathbf{r}_1\bullet\mathbf{a}_i= \mathbf{o}_v\bullet\mathbf{b}_i$, para $1\leq i\leq m$ (vide item 2 acima).

Linha 2: É requisitada a atualização na tabela base, que consiste em atribuir o valor $\mathbf{o}_v\bullet\mathbf{a}_v$ para $\mathbf{o}_1\bullet\mathbf{a}$.

5.3.2. Caso 2 - A AC de caminho é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$

As ACs de **V** com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de **V** é de um dos tipos:
   $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v, \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1$, $\mathbf{a}_2$, ... , $\mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1$, $\mathbf{b}_2$, ... , $\mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência do atributo monovalorado $\mathbf{a}_v$ é dada por:
   $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$, onde φ é um caminho de $\mathbf{T}_{R_1}$ definido por $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-1} \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a}$ (vide Figura 5.38), tal que:

   3.1. ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_{R_i} \rightarrow \mathbf{T}_{R_{i+1}}$, para 1≤i≤n-1. Considere ℓ_{j-1}, 2≤j≤n, a ligação monovalorada direta[3] onde será realizada a atualização, a qual é definida em tempo de projeto. Só poderá existir no máximo uma ligação ℓ_i no caminho φ cuja inversa é multivalorada. Caso contrário existe ambigüidade a nível de dados e o tradutor não poderá ser definido em tempo de projeto. No caso em que φ possui uma ligação cuja inversa é multivalorada, então esta ligação deve ser escolhida, uma vez que se uma das outras ligações de φ fosse escolhida existiria ambigüidade, como mostramos no Exemplo 5.9. No caso de todas as inversas das ligações serem monovaloradas, então qualquer uma das ligações do caminho pode ser escolhida. Portanto o projetista deve ser consultado para determinar a ligação em tempo de projeto. Assim, temos que as ligações ℓ_1, ..., ℓ_{j-2} e ℓ_j, ..., ℓ_{n-1} e suas inversas são monovaloradas e a inversa de ℓ_{j-1} pode ser multivalorada ou monovalorada. Na nossa abordagem, não permitiremos a modificação de atributos que sejam identificadores, portanto ℓ_{j-1} não é um identificador de $\mathbf{R}_{j-1}$.

   3.2. **a** é um atributo de $\mathbf{T}_{R_n}$, tal que **a** não é um identificador de $\mathbf{R}_n$. Esta condição garante que com o valor de **a** iremos recuperar apenas uma instância de $\mathbf{R}_n$;

[3]Como visto no Capítulo 2, ℓ_{j-1} é uma ligação direta se: (i) ℓ_{j-1} é uma ligação de atributo de valor; ou (ii) ℓ_{j-1} é uma ligação de atributo de referência; ou (iii) ℓ_{j-1} é uma ligação de chave estrangeira.

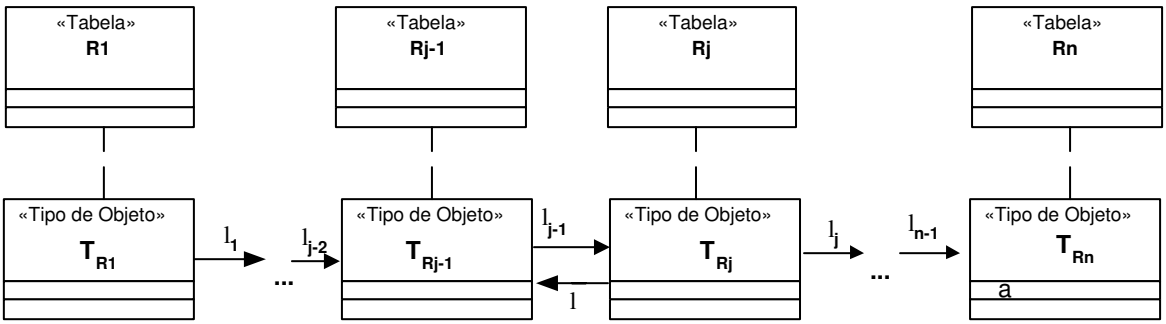

Figura 5.38. *Caminho* $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-1} \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a}$ *do tipo base* $\boldsymbol{T}_{R_1}$

O pedido de atualização, *"Atribua* $\boldsymbol{o}_v \bullet \boldsymbol{a}_v$*='v'"*, deve ser traduzido na seguinte atualização no banco de dados: *"Atribua* $\boldsymbol{o}_{j-1} \bullet \ell_{j-1} = \boldsymbol{o}_j$*"*, onde:

- $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, tal que $o_{j-1} = o_1 \bullet \ell_1 \bullet ... \bullet \ell_{j-2}$, onde $\mathbf{o}_1$ é uma instância de $\mathbf{R}_1$, $\mathbf{o}_1 \equiv \mathbf{o}_v$. Assim temos que $\mathbf{o}_{j-1}$ está relacionado com $\mathbf{o}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$. De acordo com a condição 3.1, temos que cada objeto de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ e vice-versa.

- $\mathbf{o}_j$ é uma instância de R_j, tal que $o_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a} = o_v \bullet a_v$. Assim temos que dado $\mathbf{o}_n$ de $\mathbf{R}_n$ tal que $\mathbf{o}_n \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$ ($\mathbf{a}$ é chave primária de $\mathbf{R}_n$), então $\mathbf{o}_j$ está relacionada com a instância $\mathbf{o}_n$ através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$. De acordo com a condição 3.1, temos que cada objeto de $\mathbf{R}_j$ está relacionado com um único objeto de $\mathbf{R}_n$ e vice-versa.

A Figura 5.39 mostra o trecho do Algoritmo V3 que trata o Caso 2. A seguir descrevemos cada linha do algoritmo.

Linhas 1-5: Seleciona o objeto $\mathbf{o}_{j-1}$.

Linhas 1-2: Se j=2 (a ligação monovalorada ℓ_{j-1} é a primeira ligação do caminho φ) , então $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$ da visão $\mathbf{V}$.

Linhas 3-5: Se 2<j≤n (a ligação monovalorada ℓ_{j-1} não é a primeira ligação do caminho φ), então $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$ tal que $\mathbf{o}_{j-1}$ está relacionado com o objeto $\mathbf{r}_1$ de $\mathbf{R}_1$, onde $\mathbf{r}_1$ é SE a $\mathbf{o}_v$, através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$. Note que estamos tratando de atualização de atributos monovalorados, portanto de acordo com a condição 3.1 do Algoritmo V3, o caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ é monovalorado, assim temos que um objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ está relacionado com um único objeto de $\mathbf{R}_{j-1}$ e vice-versa.

Linhas 6-10: Seleciona o objeto $\mathbf{o}_j$.

Caso 2: A ACC é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$

```
 /* Seleciona o objeto o_{j-1}*/
1.   Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */
2.        τ :  = τ ∪ { ≺ o_{j-1} := (Selecione r_1 de R_1 onde r_1≡o_v); ≻ }
3.   senão /* 2<j≤n*/
4.        τ := τ ∪ {≺ o_{j-1} := (Selecione r_1•ℓ_1• ℓ_2•...• ℓ_{j-2} de R_1 (r_1)
5.        onde r_1≡o_v);≻}
         /* o_{j-1} é uma instância de R_{j-1}, tal que o_{j-1} está relacionado
         com r_1 através do caminho ℓ_1•ℓ_2•...  •ℓ_{j-2} */

 /* Seleciona o objeto o_j*/
6.   Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */
7.        τ := τ ∪ { ≺ o_j := (Selecione r_n de R_n onde r_n•a= o_v•a_v); ≻ }
8.   senão /* 2≤j<n */
9.        τ := τ ∪ {≺ o_j := (Selecione r_j de R_j onde r_j•ℓ_j•...•ℓ_{n-1}•a =
10.                          o_v•a_v);≻ }
         /* o_j é uma instância de R_j, tal que o_j•ℓ_j•...•ℓ_{n-1}•a = o_v•a_v */

 /* Gera a atualização */
11.  Se ℓ_{j-1} é um atributo de valor
12.       τ := τ ∪ {≺ Atribua o_{j-1}•(ℓ_{j-1}) = o_j;≻}
13.  Senão se ℓ_{j-1} é um atributo de referência
14.       τ := τ ∪ {≺ Atribua o_{j-1}•(ℓ_{j-1}) = Ref(o_j);≻}
15.  Senão /* ℓ_{j-1} é uma chave estrangeira definida por
           ℓ_{j-1}:  R_{j-1} [c_1,..., c_k] ⊂ R_j [d_1,..., d_k]*/
16.       Para i de 1 até k faça:
17.          τ := τ ∪ {≺ Atribua o_{j-1}•c_i = o_j•d_i; ≻}
```

Figura 5.39. *Caso 2 do Algoritmo V3*

Linhas 6-7: Se j=n (a ligação monovalorada ℓ_{j-1} é a última ligação do caminho φ), então $\mathbf{o}_j$ recebe o objeto $\mathbf{r}_n$ de $\mathbf{R}_n$ tal que $\mathbf{r}_n \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$. De acordo com a condição 3.2 do Algoritmo V3, **a** é um identificador de $\mathbf{R}_n$, desta forma uma única instância de $\mathbf{R}_n$ será selecionada. Neste caso o objeto $\mathbf{r}_n$ já deve existir, se $\mathbf{r}_n$ não existir em R_n então trataremos esse erro como uma exceção, assim como em outras situações em que o objeto selecionado não existe.

Linhas 8-10: Se 2≤j<n (a ligação monovalorada ℓ_{j-1} não é a última ligação do caminho φ), então $\mathbf{o}_j$ é uma instância de $\mathbf{R}_j$ tal que $\mathbf{o}_j \bullet \ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1} = \mathbf{r}_n$, onde $\mathbf{r}_n$ é um objeto de $\mathbf{R}_n$ e $\mathbf{r}_n \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$. De acordo com a condição 3.2 do Algoritmo V3, **a** é um identificador de $\mathbf{R}_n$, desta forma uma única instância de $\mathbf{R}_n$ será selecionada. De acordo com a condição 3.1 do Algoritmo C2 o caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$ é monovalorado, assim temos que um objeto $\mathbf{r}_n$ de $\mathbf{R}_n$ está relacionado com um único objeto de $\mathbf{R}_j$ e vice-versa.

Linhas 11-17: Gera a atualização na tabela base.

Linhas 11-12: Se ℓ_{j-1} é um atributo de valor, então a atualização na tabela base consiste em atribuir como valor do atributo ℓ_{j-1} do objeto $\mathbf{o}_{j-1}$ o objeto $\mathbf{o}_j$.

Linha 13-14: Se ℓ_{j-1} é um atributo de referência, então a atualização na tabela base consiste em atribuir como valor do atributo ℓ_{j-1} do objeto $\mathbf{o}_{j-1}$ a referência do objeto $\mathbf{o}_j$.

Linha 15-17: Se ℓ_{j-1} é uma chave estrangeira definida por ℓ_{j-1}: $\mathbf{R}_{j-1}$ [$\mathbf{c}_1$,..., $\mathbf{c}_k$]⊂ $\mathbf{R}_j$ [$\mathbf{d}_1$,..., $\mathbf{d}_k$], então a atualização na tabela base consiste em atribuir para os atributos $\mathbf{c}_1$,..., $\mathbf{c}_k$ do objeto $\mathbf{o}_{j-1}$ os valores correspondentes dos atributos $\mathbf{d}_1$,..., $\mathbf{d}_k$ do objeto $\mathbf{o}_j$.

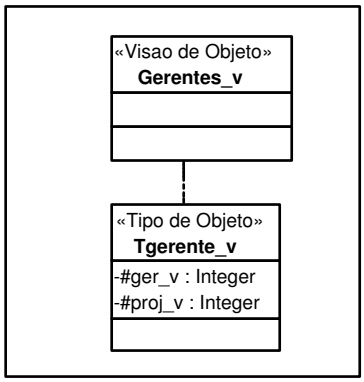

Figura 5.40. *Esquema da visão de objetos* ***Gerentes_v***

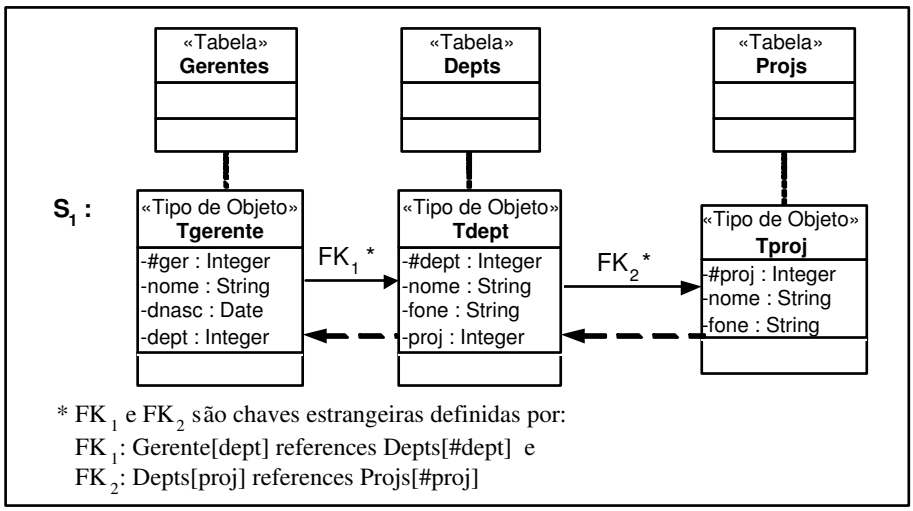

Figura 5.41. *Esquema do banco de dados* $\boldsymbol{S}_1$

Exemplo 5.8

Suponha o esquema da visão **Gerentes_v** apresentado na Figura 5.40, e $\mathbf{S}_1$, o esquema do banco de dados apresentado na Figura 5.41. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_1$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v**≡**Gerentes**
- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
- Assertivas de Correspondência do Atributo ♯**proj_v** de **Tgerente_v**
  ψ_3: **Tgerente_v**•♯**proj_v** ≡ **Tgerente**•FK$_1$•FK$_2$

Neste exemplo, como todas as inversas das ligações do caminho FK$_1$•FK$_2$ são monovaloradas então qualquer uma das ligações pode ser escolhida para ser realizada a atualização. Suponha que a ligação FK$_1$ do caminho FK$_1$•FK$_2$ foi escolhida. A Figura 5.42 apresenta o tradutor *'Gerentes$_v$_Modifica_Proj1'*, gerado pelo algoritmo V3, o qual é usado para traduzir modificações do atributo monovalorado ♯**proj_v** da visão **Gerentes_v**. Note que o tradutor recebe como entrada o objeto $\mathbf{o}_v$ da visão que no Oracle corresponde ao objeto *:new*.

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo V3.

Linha 1: De ψ_3 temos que a ligação FK$_1$ é a primeira ligação do caminho de derivação de ♯**proj_v** (FK$_1$•FK$_2$). Assim $\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$ de **Gerentes**, tal que $\mathbf{r}_1$≡$\mathbf{o}_v$. Note que, de acordo com a ACO ψ_2, $\mathbf{r}_1$≡$\mathbf{o}_v$ sss $\mathbf{r}_1$•♯**ger**= $\mathbf{o}_v$•♯**ger_v**.

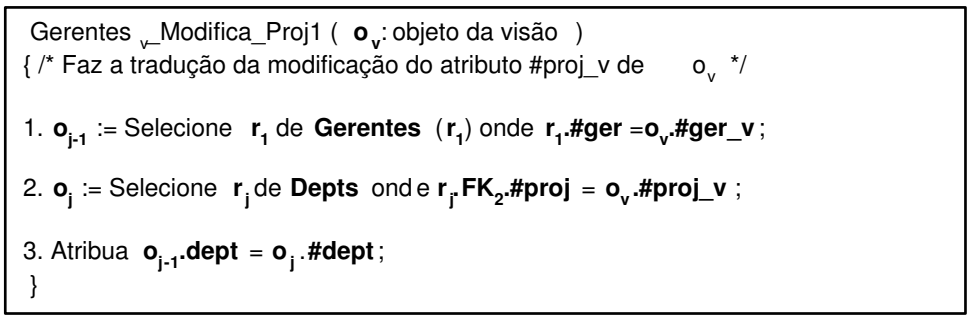

Figura 5.42. *Tradutor* 'Gerentes$_v$_Modifica_Proj1'

Linha 2: De ψ_3 temos que a ligação FK_1 não é a última ligação do caminho de derivação de $\sharp\mathbf{proj\_v}$ ($\text{FK}_1 \bullet \text{FK}_2$). Assim, $\mathbf{o}_j$ recebe o objeto $\mathbf{r}_j$ de **Depts**, onde $\mathbf{r}_j \bullet \text{FK}_2 \bullet \sharp\mathbf{proj} = \mathbf{o}_v \bullet \sharp\mathbf{proj\_v}$.

Linha 3: Como FK_1 é uma chave estrangeira, então a atualização requerida consiste em atribuir para o atributo **dept** de $\mathbf{o}_{j-1}$ o valor do atributo $\sharp$**dept** de $\mathbf{o}_j$.

```
CREATE OR REPLACE TRIGGER Modifica_Projeto1_Oracle INSTEAD OF UPDATE On
Gerentes_v
FOR EACH ROW
 BEGIN
    Update Gerentes g
    SET g.dept = (Select d.#dept From Depts d, Projs p
                    Where p.#proj = :new.proj_v and d.proj =p.#proj)
     Where  g.#ger=:new.#ger_v ;
END;
```

Figura 5.43. *Tradutor 'Modifica_Projeto1_Oracle'*

Na Figura 5.43, mostramos a título de ilustração o trigger 'Modifica_Projeto1_Oracle' que corresponde ao tradutor 'Gerentes$_v$_Modifica_Proj1' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma modificação no atributo monovalorado $\sharp\mathbf{proj\_v}$ da visão **Gerentes_v**, este trigger é disparado.

Exemplo 5.9

Suponha o esquema da visão **Gerentes_v** apresentado na Figura 5.40, e $\mathbf{S}_2$, o esquema do banco de dados apresentado na Figura 5.44. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_2$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v**$\equiv$**Gerentes**

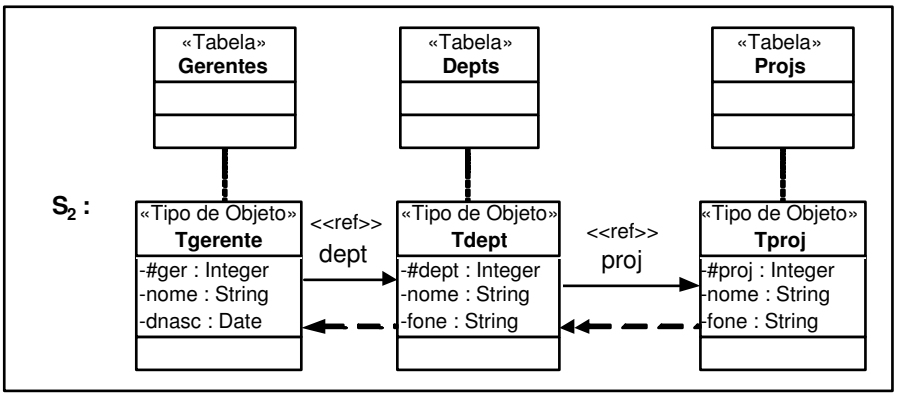

Figura 5.44. *Esquema do banco de dados* $\boldsymbol{S}_2$

- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
- Assertivas de Correspondência do Atributo ♯**proj_v** de **Tgerente_v**
  ψ_3: **Tgerente_v**•♯**proj_v** ≡ **Tgerente**•**dept**•**proj**

Neste exemplo, como a inversa da ligação **proj** do caminho **dept**•**proj** é multivalorada, então **proj** é a ligação onde deve ser realizada a atualização (Condição 3.1 do Caso 2). Note que a atualização não poderia ser realizada na ligação **dept**, pois como a inversa da ligação **proj** é multivalorada, dado um projeto **p** poderia existir mais de um departamento associado a **p**, causando assim ambigüidade a nível de dados. A Figura 5.45 apresenta o tradutor *'Gerentes$_v$_Modifica_Proj2'*, gerado pelo algoritmo V3, o qual é usado para traduzir modificações do atributo monovalorado ♯**proj_v** da visão **Gerentes_v**. Note que o tradutor recebe como entrada o objeto $\mathbf{o}_v$ da visão que no Oracle corresponde ao objeto *:new*.

```
Gerentes_v_Modifica_Proj2 ( o_v : objeto da visão )
{ /* Faz a tradução da modificação do atributo #proj_v de  o_v */

1. o_{j-1} := Selecione r_1.dept de Gerentes (r_1) onde r_1.#ger = o_v.#ger_v ;

2. o_j := Selecione r_n de Projs (r_n) onde r_n.#proj = o_v.#proj_v ;

3. Atribua o_{j-1}.proj = Ref( o_j );
 }
```

Figura 5.45. *Tradutor 'Gerentes$_v$_Modifica_Proj2'*

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo V3.

Linha 1: De ψ_3 temos que a ligação **proj** não é a primeira ligação do caminho de derivação de ♯**proj_v** (**dept**•**proj**). Assim, dado o objeto $\mathbf{r}_1$ de **Gerentes** onde $\mathbf{r}_1 \equiv \mathbf{o}_v$,

$\mathbf{o}_{j-1}$ recebe o objeto $\mathbf{r}_1$•**dept** de **Depts**. Note que, de acordo com a ACO ψ_2, $\mathbf{r}_1 \equiv \mathbf{o}_v$ sss $\mathbf{r}_1$•♯**ger**= $\mathbf{o}_v$•♯**ger_v**.

Linha 2: De ψ_3 temos que a ligação **proj** é a última ligação do caminho de derivação de ♯**proj_v** (**dept**•**proj**). Assim, dado o objeto $\mathbf{r}_n$ de **Projs** onde $\mathbf{r}_n$•♯**proj** = $\mathbf{o}_v$•♯**proj_v**, $\mathbf{o}_j$ recebe o objeto $\mathbf{r}_n$ de **Projs**.

Linha 3: Como **proj** é um atributo de referência, então a atualização requerida consiste em atribuir para o atributo **proj** de $\mathbf{o}_{j-1}$ a referência de $\mathbf{o}_j$.

```
CREATE OR REPLACE TRIGGER Modifica_Projeto2_Oracle INSTEAD OF UPDATE On
Gerentes_v
FOR EACH ROW
 BEGIN
    Update Gerentes g
    SET g.dept.proj = (Select Ref( p) From Projs p Where p.#proj = :new.proj_v)
    Where g.#ger=:new.#ger_v;
END;
```

Figura 5.46. *Tradutor 'Modifica_Projeto2_Oracle'*

Na Figura 5.46, mostramos a título de ilustração o trigger 'Modifica_Projeto2_Oracle' que corresponde ao tradutor *'$Gerentes_v$_Modifica_Proj2'* gerado para o banco de dados Oracle 8i. Sempre que for realizada uma modificação no atributo monovalorado ♯**proj_v** da visão **Gerentes_v**, este trigger é disparado.

5.4. Definindo Tradutores para Operações de Adição em Visões

Nesta seção, considere a visão **V** cujos objetos são do tipo $\mathbf{T}_v$, como mostrado na Figura 5.47. Suponha um pedido de adição do objeto $\mathbf{o}_v$ do tipo $\mathbf{T}_v$ na visão **V**. Esse pedido de atualização no nosso formalismo é definido por:
"Adicione $\mathbf{o}_v$ em **V**"

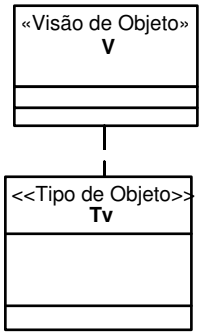

Figura 5.47. *Esquema da Visão* ***V***

O algoritmo V1, apresentado no Apêndice A, gera tradutores para adição na visão **V**. Este algoritmo gera tradutores de atualização para os três casos apresentados a seguir:

- O Caso 1 trata a situação em que a ACE é de eqüivalência ou subconjunto;
- O Caso 2 trata a situação em que a ACE é de diferença;
- O Caso 3 trata a situação em que a ACE é de intersecção.

O caso 1 trata a adição em visão, cujo tipo (da visão) pode ter atributos monovalorados e multivalorados. Os outros casos tratam a adição em visão, cujo tipo (da visão) pode ter apenas atributos monovalorados. Mostraremos que nestes casos não existe ambigüidade a nível de dados e a tradução pode ser gerada em tempo de projeto.

5.4.1. Caso 1 - A ACE é de Eqüivalência ou Subconjunto

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ com as tabelas bases é de um dos tipos:
   $\mathbf{V} \equiv \mathbf{R}_1$ (Equivalência) ou $\mathbf{V} \subset \mathbf{R}_1$ (Subconjunto), onde $\mathbf{R}_1$ é uma relação base do tipo $\mathbf{T}_{R_1}$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_1}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. Para cada assertiva de correspondência de caminho multivalorado entre os tipos $\mathbf{T}_v$ e $\mathbf{T}_{R_1}$ deve ser satisfeita a condição 3 definida na seção 5.1.1.

4. As assertivas de correspondência de caminhos monovalorados entre $\mathbf{T}_v$ e $\mathbf{T}_{R_1}$ podem ser de um dos dois tipos definidos abaixo:

   4.1. $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \mathbf{a}$, onde $\mathbf{a}$ é um atributo de $\mathbf{T}_{R_1}$;

   4.2. $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$, onde φ é um caminho de $\mathbf{T}_{R_1}$ dado por $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a}$, onde ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_{R_i} \rightarrow \mathbf{T}_{R_{i+1}}$, para $1 \leq i \leq n\text{-}1$, tal que:

      4.2.1. As ligações ℓ_2, ..., ℓ_{n-1} e suas inversas são monovaloradas. Desta forma, temos que cada objeto de $\mathbf{R}_2$ está relacionado com um único objeto de $\mathbf{R}_n$, através do caminho $\ell_2 \bullet \ell_3 \bullet$ ... $\bullet \ell_{n-1}$, e vice-versa. ℓ_1 é a ligação monovalorada direta onde será feita a atualização. A inversa desta ligação pode ser multivalorada ou monovalorada;

      4.2.2. $\mathbf{a}$ é um atributo de $\mathbf{T}_{R_n}$, tal que $\mathbf{a}$ é um identificador de $\mathbf{R}_n$. Esta condição garante que com o valor de $\mathbf{a}$ iremos recuperar apenas uma instância de $\mathbf{R}_n$.

Neste caso, o pedido de atualização, *"Adicione* $\boldsymbol{o}_v$ *em* $\boldsymbol{V}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o}$ *em* $\boldsymbol{R}_1$*"*, onde $\mathbf{o}$ é uma instância de $\mathbf{T}_{R_1}$ semanticamente equivalente ao objeto $\mathbf{o}_v$. A Figura 5.48 mostra o trecho do Algoritmo V1 que trata o Caso 1. A seguir descrevemos cada linha do algoritmo V1.

Caso1: A assertiva de correspondência é da forma V ≡ R_1 ou da forma V ⊂ R_1, onde R_1 é uma relação base do banco de dados, cujo tipo é T_{R_1}, faça:

1. Construtor_o := Cria_Construtor_Objeto_Visao(T_{R_1}, T_v);
   /* Retorna um construtor de objeto do tipo T_{R_1}*/
2. τ := { $\prec$ o := Construtor_o;
3. Adicione o em R_1; $\succ$}

4. Para cada atributo multivalorado Lista_T_c de T_v, onde
5. T_v•Lista_T_c ≡ T_{R_1}•φ, faça:
6. τ := τ ∪ {$\prec$ Para cada objeto o_c em o_v•lista_T_c, faça:
7. Adicione o_c em o_v•Lista_T_c; $\succ$ }

Figura 5.48. *Caso 1 do Algoritmo V1*

Linha 1-2: o recebe o objeto criado pelo construtor de objetos **Construtor_o** gerado na linha 1, o qual cria um objeto do tipo $\mathbf{T}_{R_1}$ SE ao objeto $\mathbf{o}_v$ (adicionado na visão **V**). O construtor de objetos **Construtor_o** do tipo $\mathbf{T}_{R_1}$ foi gerado pelo procedimento Cria_Construtor_Objeto_ Visao ($\mathbf{T}_{R_1}$, $\mathbf{T}_v$), o qual é apresentado no Apêndice A. A seguir discutimos brevemente o procedimento Cria_Construtor_Objeto_Visao.

Os construtores de objetos da visão são gerados pelo procedimento Cria_Construtor_ Objeto _Visao. Este procedimento recebe como entrada os tipos $\mathbf{T}_1$ e $\mathbf{T}_v$, onde $\mathbf{T}_1$ é um tipo base e $\mathbf{T}_v$ é um tipo da visão, e gera o construtor de objetos do tipo $\mathbf{T}_1$, que cria um objeto de $\mathbf{T}_1$ semanticamente equivalente a um dado objeto $\mathbf{o}_v$ de $\mathbf{T}_v$. Para gerar um construtor de objetos, que cria um objeto $\mathbf{o}_1$ de $\mathbf{T}_1$ a partir de um objeto $\mathbf{o}_v$ de $\mathbf{T}_v$, deve-se determinar valores para os atributos de $\mathbf{o}_1$ a partir de $\mathbf{o}_v$. Como mostrado no algoritmo Cria_Construtor_Objeto_Visao, os valores dos atributos de $\mathbf{o}_1$ são determinados com base nas assertivas de correspondência entre os tipos $\mathbf{T}_1$ e $\mathbf{T}_v$ ($\mathbf{T}_1$ e $\mathbf{T}_v$ são semanticamente relacionados). Este formalismo permite definir o mapeamento entre os objetos de tipos com estruturas diferentes. Maiores detalhes deste procedimento serão apresentados durante as discussões dos exemplos.

Linha 3: É requisitada a atualização na tabela base, que consiste em adicionar o objeto **o** do tipo $\mathbf{T}_{R_1}$ na tabela $\mathbf{R}_1$ onde **o** é semanticamente equivalente ao objeto $\mathbf{o}_v$, adicionado na visão **V**.

Linha 4-5: Para cada atributo multivalorado **Lista_**$\mathbf{T}_c$ de $\mathbf{T}_v$, onde a assertiva de correspondência é do tipo $\mathbf{T}_v$•**Lista_**$\mathbf{T}_c \equiv \mathbf{T}_{R_1}$•$\varphi$, são executadas as linhas 6-7.

Linhas 6-7: Para cada objeto $\mathbf{o}_c$ do tipo $\mathbf{T}_c$ a ser adicionado em **lista_**$\mathbf{T}_c$, é requisitada a atualização para traduzir a adição do atributo multivalorado, que consiste em adicionar o objeto $\mathbf{o}_c$ na coleção aninhada **Lista_**$\mathbf{T}_c$ do objeto $\mathbf{o}_v$, adicionado na visão

V. Desta forma o tradutor para adição na coleção aninhada $\mathbf{Lista\_T}_c$ será executado.

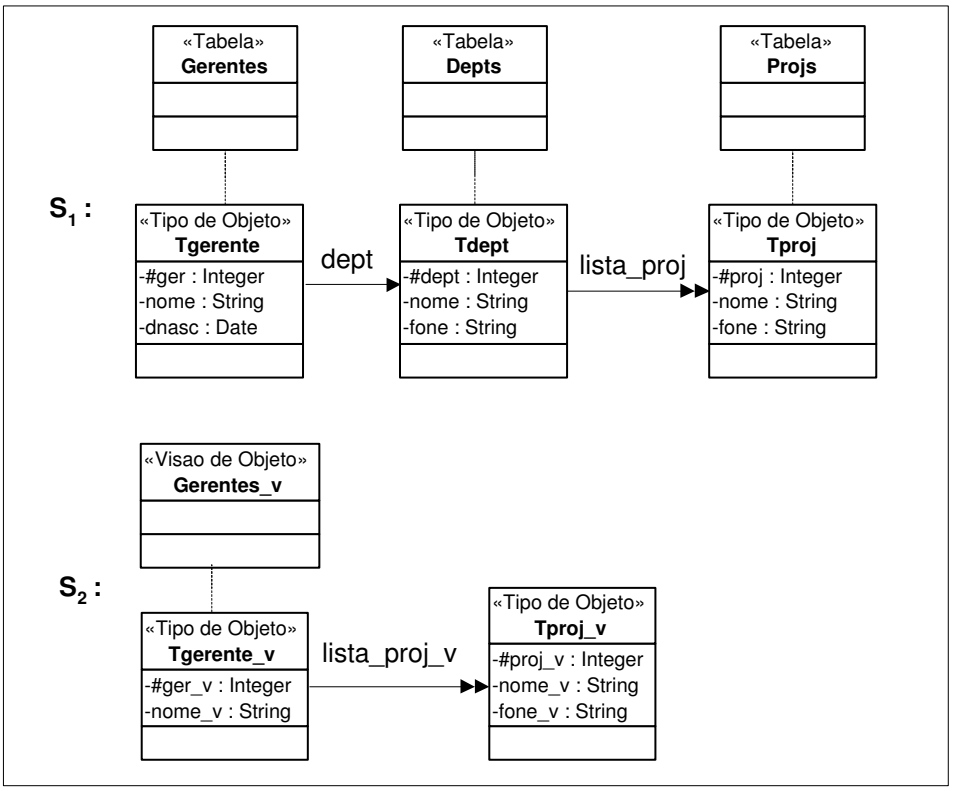

Figura 5.49. *Esquema do banco de dados* $\boldsymbol{S}_1$ *e esquema da visão de objetos* $\boldsymbol{S}_2$

Exemplo 5.10

Sejam $\mathbf{S}_1$, o esquema do banco de dados, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v**, apresentados na Figura 5.49. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_1$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v≡Gerentes**
- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
  ψ_3: [**Tproj_v**, {♯**proj_v**}] ≡ [**Tproj**, {♯**proj**}]
- Assertivas de Correspondência de Caminhos entre os tipos **Tgerente_v** e **Tgerente**
  ψ_4: **Tgerente_v•lista_proj_v ≡ Tgerente•dept•lista_proj**
  ψ_5: **Tgerente_v•♯ger_v ≡ Tgerente•♯ger**
  ψ_6: **Tgerente_v•nome_v ≡ Tgerente•nome**

A Figura 5.50 apresenta o tradutor 'Adiciona_Gerentes$_v$1', gerado pelo algoritmo V1, o qual é usado para traduzir uma adição de um objeto na visão **Gerentes_v** em atualizações nas tabelas base.

```
Adiciona_Gerentes v1 (o_v: objeto da visão )
{ /* Faz a Tradução da adição de  o_v na visão Gerentes_v */

1. o := Tgerente (#ger :o_v.#ger_v, nome : o_v.nome_v );
   /* Construtor de objeto de Tgerente criado por
      Cria_Construtor_Objeto_Visao(Tgerente, Tgerente_v ) */

2. Adicione o em Gerentes ;

3. Para cada objeto o_c em o_v. lista_proj_v do tipo Tproj_v de Tgerente_v , faça:
4.       Adicione o_c em o_v. lista_proj_v ;
 }
```

Figura 5.50. *Tradutor 'Adiciona_Gerentes$_v$1'*

A seguir mostraremos como é gerada cada linha do tradutor usando o Algoritmo V1.

Linha 1: **o** recebe o objeto criado pelo construtor de objetos: ***Tgerente(♯ger:*** o_v***•♯ger_v, nome:*** o_v***•nome_v)***, o qual cria um objeto do tipo **Tgerente** sendo este semanticamente equivalente ao objeto $\mathbf{o}_v$ (adicionado na visão **Gerentes_v**). Este construtor de objetos foi gerado pelo procedimento Cria_Construtor_Objeto_Visao (**Tgerente**, **Tgerente_v**) baseado nas ACs ψ_4, ψ_5 e ψ_6 que especificam formalmente o relacionamento entre **Tgerente** e **Tgerente_v**. Este formalismo permite ao construtor definir automaticamente o mapeamento entre objetos que possuem estruturas diferentes.

Linha 2: É gerada a atualização requerida; a qual consiste em adicionar o objeto **o** (instância de **Tgerente** criada na linha 1) na tabela **Gerentes**.

Linhas 3-4: Para cada objeto $\mathbf{o}_c$ do tipo **Tproj_v** a ser adicionado em **lista_proj_v**, é requisitada a atualização para traduzir a adição do atributo multivalorado, que consiste em adicionar o objeto $\mathbf{o}_c$ na coleção aninhada **lista_proj_v** do objeto $\mathbf{o}_v$, adicionado na visão **Gerentes_v**.

Na Figura 5.51, mostramos a título de ilustração o trigger 'Adiciona_Gerente$_v$Oracle1' que corresponde ao tradutor 'Adiciona_Gerentes$_v$1' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma adição na visão **Gerentes_v**, este trigger é disparado. O valor *:new* refere-se ao objeto inserido, no nosso caso $\mathbf{o}_v$.

```
CREATE OR REPLACE TRIGGER Adiciona_Gerente      v_Oracle1
   INSTEAD OF INSERT ON Gerentes_v
    Lista_nt    ListaProj;
BEGIN
1.   Lista_nt := :new.lista_proj_v;
2.   Insert into   Gerentes   (#ger, nome )
3.   Values (  :new.#ger_v, :new.nome_v    );

4.   FOR i in 1..Lista_nt.count LOOP
5.       Insert into Table( Select v.    lista_proj_v From  Gerentes_v  v
6.                                Where v.    #ger = :new.#ger)
7.       Select Lista_nt(i) from Dual;
8.   End Loop;
END;
```

Figura 5.51. *Tradutor 'Adiciona_Gerente$_v$Oracle1'*

5.4.2. Caso 2 - A ACE é de Diferença

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ com as tabelas bases é dada por: $\mathbf{V} \equiv \mathbf{R}_1$ - $\mathbf{R}_2$ (Diferença), onde $\mathbf{R}_1$ e $\mathbf{R}_2$ são relações bases dos tipos $\mathbf{T}_{R_1}$ e $\mathbf{T}_{R_2}$ respectivamente.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_v$ com os objetos de $\mathbf{T}_{R_1}$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência dos objetos de $\mathbf{T}_v$ com os objetos de $\mathbf{T}_{R_2}$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_2}, \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m\}]$
onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{c}_1, \mathbf{c}_2, ... , \mathbf{c}_m$ são atributos de $\mathbf{T}_{R_2}$.

Neste caso, o pedido de atualização, *"Adicione $\boldsymbol{o}_v$ em $\boldsymbol{V}$"*, deve ser traduzido como especificado abaixo:

- (i) Se existir $\mathbf{r}_1$ em $\mathbf{R}_1$ tal que $\mathbf{r}_1 \equiv \mathbf{o}_v$ e existir $\mathbf{r}_2$ em $\mathbf{R}_2$ tal que $\mathbf{r}_2 \equiv \mathbf{o}_v$ então *"Remova $\boldsymbol{r}_2$ de $\boldsymbol{R}_2$"*, onde $\mathbf{r}_2$ é uma instância de $\mathbf{R}_2$ SE ao objeto $\mathbf{o}_v$;

- (ii) Se não existir $\mathbf{r}_1$ em $\mathbf{R}_1$ tal que $\mathbf{r}_1 \equiv \mathbf{o}_v$ e existir $\mathbf{r}_2$ em $\mathbf{R}_2$ tal que $\mathbf{r}_2 \equiv \mathbf{o}_v$ então *"Adicione $\boldsymbol{o}_1$ em $\boldsymbol{R}_1$"*, onde $\mathbf{o}_1$ é uma instância de $\mathbf{T}_{R_1}$ SE ao objeto $\mathbf{o}_v$; e *"Remova $\boldsymbol{r}_2$ de $\boldsymbol{R}_2$"*, onde $\mathbf{r}_2$ é uma instância de $\mathbf{R}_2$ SE ao objeto $\mathbf{o}_v$;

- (iii) Se não existir $\mathbf{r}_1$ em $\mathbf{R}_1$ tal que $\mathbf{r}_1 \equiv \mathbf{o}_v$ e não existir $\mathbf{r}_2$ em $\mathbf{R}_2$ tal que $\mathbf{r}_2 \equiv \mathbf{o}_v$ então
"Adicione $\boldsymbol{o}_1$ em $\boldsymbol{R}_1$", onde $\mathbf{o}_1$ é uma instância de $\mathbf{T}_{R_1}$ SE ao objeto $\mathbf{o}_v$.

A Figura 5.52 mostra o trecho do Algoritmo V1 que trata o Caso 2. A seguir descrevemos cada linha do algoritmo.

Caso 2: A assertiva de correspondência é da forma V ≡ R_1 - R_2, onde R_1 e R_2 são relações base do banco de dados, cujos tipos são T_{R_1} e T_{R_2}, faça:

```
1.  τ := {≺ Se existir r1 em R1 tal que r1 ≡ ov então
2.              Se existir r2 em R2 tal que r2 ≡ ov então
3.                   Remova r2 de R2 ;
4.           senão ≻}
5.              construtor_o1 := Cria_Construtor_Objeto_Visao(TR1, Tv);
                /* Retorna um construtor de objeto do tipo TR1*/
6.              τ := τ ∪ {≺ o1 := construtor_o1;
7.              Se existir r2 em R2 tal que r2 ≡ ov então
8.                   Adicione o1 em R1;
9.                   Remova r2 de R2 ;
10.              senão
11.                   Adicione o1 em R1; ≻}
```

Figura 5.52. *Caso 2 do Algoritmo V1*

Linha 1: É verificada a existência do objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$, adicionado na visão **V**.

Linhas 2-3: Caso exista o objeto $\mathbf{r}_1$ (verificado na linha 1) e também exista o objeto $\mathbf{r}_2$ de $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE ao objeto $\mathbf{o}_v$, então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{r}_2$ da tabela $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE ao objeto $\mathbf{o}_v$, adicionado na visão **V**.

Linhas 4-6: Caso não exista o objeto $\mathbf{r}_1$ (verificado na linha 1), então $\mathbf{o}_1$ recebe o objeto criado pelo construtor de objetos **Construtor_o**$_1$, o qual cria um objeto do tipo $\mathbf{T}_{R_1}$ SE ao objeto $\mathbf{o}_v$. O construtor de objetos **Construtor_o**$_1$ do tipo $\mathbf{T}_{R_1}$ foi gerado pelo procedimento Cria_Construtor_Objeto_Visao ($\mathbf{T}_{R_1}$, $\mathbf{T}_v$), o qual é apresentado no Apêndice A.

Linhas 7-9: Caso não exista o objeto $\mathbf{r}_1$ (verificado na linha 1), mas exista o objeto $\mathbf{r}_2$ de $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE ao objeto $\mathbf{o}_v$, então é requisitada as atualizações nas tabelas bases, que consistem em adicionar o objeto $\mathbf{o}_1$ do tipo $\mathbf{T}_{R_1}$ na tabela $\mathbf{R}_1$ e remover o objeto $\mathbf{r}_2$ da tabela $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE ao objeto $\mathbf{o}_v$, adicionado na visão **V**.

Linhas 10-11: Caso não exista o objeto $\mathbf{r}_1$ (verificado na linha 1), e também não exista o objeto $\mathbf{r}_2$ de $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE ao objeto $\mathbf{o}_v$, então é requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_1$ do tipo $\mathbf{T}_{R_1}$ na tabela $\mathbf{R}_1$.

5.4.3. Caso 3 - A ACE é de Intersecção

As ACs de **V** com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de **V** com as tabelas bases é dada por: $\mathbf{V} \equiv \cap_{i=1}^{n} \mathbf{R}_i$ (Intersecção), onde $\mathbf{R}_i$ é uma relação base do tipo $\mathbf{T}_{R_i}$.
2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_i}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_i}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_i}$.

Neste caso, o pedido de atualização, *"Adicione* $\boldsymbol{o}_v$ *em* ***V****"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o}_i$ *em* $\boldsymbol{R}_i$*"*, onde $\mathbf{o}_i$ é uma instância de $\mathbf{R}_i$ semanticamente equivalente ao objeto $\mathbf{o}_v$, adicionado na visão **V**. A Figura 5.53 mostra o trecho do Algoritmo V1 que trata o Caso 3. A seguir descrevemos cada linha do algoritmo.

Caso 3: A assertiva de correspondência é da forma $V \equiv \cap_{i=1}^{n} R_i$, onde R_i é uma relação base do banco de dados, cujo tipo é T_{R_i}, faça:

```
1.  τ :  = { ≺ Para cada Ri faça:  ≻}
2.              Construtor_oi :=Cria_Construtor_Objeto_Visao(TRi, Tv);
                /* Retorna um construtor de objetos do tipo TRi */
3.              τ := τ ∪ { ≺ oi :  = Construtor_oi;
4.              Adiciona oi em Ri; ≻}
```

Figura 5.53. *Caso 3 do Algoritmo V1*

Linhas 1-3: Para cada tabela $\mathbf{R}_i$, onde $1 \leq i \leq n$, $\mathbf{o}_i$ recebe o objeto criado pelo construtor de objetos **Construtor_o**$_i$ gerado na linha 2, o qual cria um objeto do tipo $\mathbf{T}_{R_i}$ SE ao objeto $\mathbf{o}_v$ (adicionado na visão **V**). O construtor de objetos **Construtor_o**$_i$ do tipo $\mathbf{T}_{R_i}$ foi gerado pelo procedimento Cria_Construtor_Objeto_Visao ($\mathbf{T}_{R_i}$, $\mathbf{T}_v$), o qual é apresentado no Apêndice A.

Linha 4: É requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_i$ do tipo $\mathbf{T}_{R_i}$ na tabela $\mathbf{R}_i$.

5.5. Definindo Tradutores para Operações de Remoção de Objetos de Visões

Nesta seção considere a visão **V** cujos objetos são do tipo $\mathbf{T}_v$ como mostrato na Figura 5.54. Suponha $\mathbf{o}_v$ um objeto da visão **V**. Considere um pedido de remoção do objeto $\mathbf{o}_v$ da visão **V**. Esse pedido de atualização no nosso formalismo é definido por:
"Remova $\mathbf{o}_v$ de **V**"

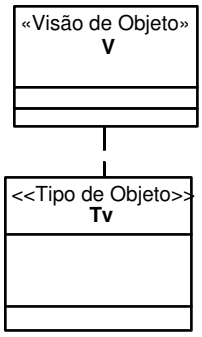

Figura 5.54. *Visão* $\boldsymbol{V}$ *do tipo* $\boldsymbol{T}_v$

O algoritmo V2, apresentado no Apêndice A, gera tradutores para remoção de objetos da visão **V**. Este algoritmo gera tradutores de atualização para os três casos apresentados a seguir:

- O Caso 1 trata a situação onde a ACE é de eqüivalência ou subconjunto;
- O Caso 2 trata a situação onde a ACE é de diferença;
- O Caso 3 trata a situação onde a ACE é de intersecção.

Mostraremos que nestes casos não existe ambigüidade a nível de dados e a tradução pode ser gerada em tempo de projeto. A seguir discutimos cada caso do algoritmo.

5.5.1. Caso 1 - A ACE é de Equivalência ou Subconjunto

As ACs de **V** com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de **V** com as tabelas bases é de um dos tipos:
   V≡**R** (Equivalência) ou **V**⊂ **R** (Subconjunto), onde **R** é uma relação base do tipo $\mathbf{T}_R$.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_R$ com os objetos de $\mathbf{T}_v$ é dada por:
   $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_R, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_R$.

Neste caso, o pedido de atualização, *"Remova* $\boldsymbol{o}_v$ *de* ***V****"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* ***r*** *de* ***R*** *onde* $\boldsymbol{r} \equiv \boldsymbol{o}_v$*"*, onde **r** é uma instância de **R** semanticamente equivalente (SE) ao objeto $\mathbf{o}_v$. A Figura 5.55 mostra o trecho do Algoritmo V2 que trata o Caso 1. A seguir descrevemos a linha do algoritmo.

Caso1: A assertiva de correspondência de extensão é da forma V ≡ R ou
da forma V ⊂ R, onde R é uma relação base do banco de dados faça:

1. τ : = { ≺ Remova r de R onde r ≡ o_v; ≻ }

Figura 5.55. *Caso 1 do Algoritmo V2*

Linha 1: É requisitada a atualização na tabela base, que consiste em remover o objeto **r** da tabela **R** onde **r** é semanticamente equivalente ao objeto $\mathbf{o}_v$, removido da visão **V**.

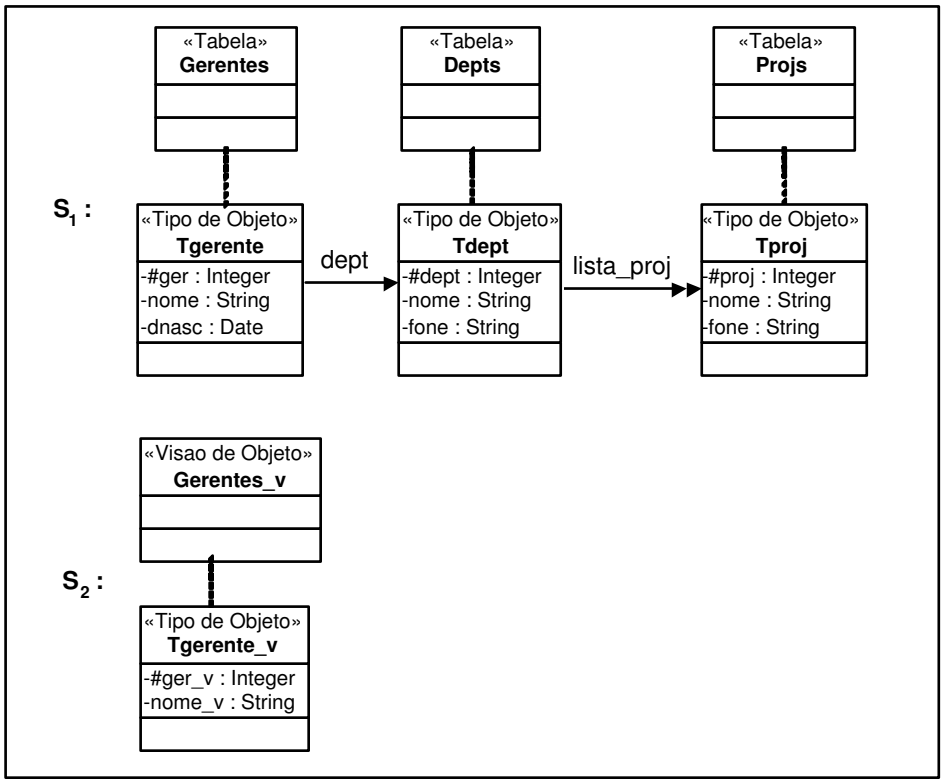

Figura 5.56. *Esquema do banco de dados* $\boldsymbol{S}_1$ *e esquema da visão de objetos* $\boldsymbol{S}_2$

Exemplo 5.11

Sejam $\mathbf{S}_1$, o esquema do banco de dados, e $\mathbf{S}_2$, o esquema da visão **Gerentes_v**, apresentados na Figura 5.56. Considere as ACs da visão **Gerentes_v** com o esquema $\mathbf{S}_1$ dadas a seguir:

- Assertivas de Correspondência de Extensão
  ψ_1: **Gerentes_v**≡**Gerentes**
- Assertivas de Correspondência de Objetos
  ψ_2: [**Tgerente_v**, {♯**ger_v**}] ≡ [**Tgerente**, {♯**ger**}]
- Assertivas de Correspondência de Caminhos entre os tipos **Tgerente_v** e **Tgerente**
  ψ_3: **Tgerente_v**•♯**ger_v** ≡ **Tgerente**•♯**ger**
  ψ_4: **Tgerente_v**•**nome_v** ≡ **Tgerente**•**nome**

A Figura 5.57 apresenta o tradutor 'Remove_Gerentes$_v$1', gerado pelo algoritmo V2, o qual é usado para traduzir uma remoção de um objeto da visão **Gerentes_v** em atualização na tabela base.

A seguir mostraremos como é gerada a linha do tradutor usando o Algoritmo V2.

```
Remove_Gerentes v1 (o_v: objeto da visão )
{ /* Faz a Tradução da remoção de o_v da visão Gerentes_v */

1. Remova r de Gerentes r onde r.#ger = o_v.#ger_v;
}
```

Figura 5.57. *Tradutor 'Remove_Gerentes$_v$1'*

Linha 1: É gerada a atualização requerida; a qual consiste em remover o objeto **r** da tabela **Gerentes** tal que **r** é semanticamente equivalente ao objeto $\mathbf{o}_v$, removido da visão $\mathbf{Gerentes}_v$. Note que, de acordo com a AC de objeto ψ_2, ($\mathbf{r} \equiv \mathbf{o}_v$) sss ($\mathbf{r} \bullet \sharp \mathbf{ger} = \mathbf{o}_v \bullet \sharp \mathbf{ger\_v}$).

Na Figura 5.58, mostramos a título de ilustração o trigger 'Remove_Gerente$_v$Oracle1' que corresponde ao tradutor 'Remove_Gerente$_v$1' gerado para o banco de dados Oracle 8i. Sempre que for realizada uma remoção na visão **Gerentes_v**, este trigger é disparado. O valor *:old* refere-se ao objeto removido, no nosso caso $\mathbf{o}_v$.

```
CREATE OR REPLACE TRIGGER Remove_Gerente v_Oracle1
   INSTEAD OF DELETE ON Gerentes_v
 BEGIN
 1.    Delete From   Gerentes  Where #ger = :old.#ger_v ;
END;
```

Figura 5.58. *Tradutor 'Remove_Gerente$_v$Oracle1'*

5.5.2. Caso 2 - A ACE é de Diferença

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ com as tabelas bases é dada por: $\mathbf{V} \equiv \mathbf{R}_1$ - $\mathbf{R}_2$ (Diferença), onde $\mathbf{R}_1$ e $\mathbf{R}_2$ são relações bases dos tipos $\mathbf{T}_{R_1}$ e $\mathbf{T}_{R_2}$ respectivamente.

2. A assertiva de correspondência dos objetos de $\mathbf{T}_v$ com os objetos de $\mathbf{T}_{R_1}$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_1}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_1}$.

3. A assertiva de correspondência dos objetos de $\mathbf{T}_v$ com os objetos de $\mathbf{T}_{R_2}$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_2}, \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m\}]$
onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{c}_1, \mathbf{c}_2, ... , \mathbf{c}_m$ são atributos de $\mathbf{T}_{R_2}$.

Neste caso o projetista será consultado para escolher a semântica de atualização "Adicione"ou "Remove".

Caso a SA escolhida seja "Adicione", então o pedido de atualização, *"Remova* $\boldsymbol{o}_v$ *de* $\boldsymbol{V}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Adicione* $\boldsymbol{o}_2$ *em* $\boldsymbol{R}_2$*"*, onde $\mathbf{o}_2$ é uma instância de $\mathbf{T}_{R_2}$ SE ao objeto $\mathbf{o}_v$, removido da visão $\mathbf{V}$.

Caso a SA escolhida seja "Remove", então o pedido de atualização, *"Remova* $\boldsymbol{o}_v$ *de* $\boldsymbol{V}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* $\boldsymbol{r}_1$ *de* $\boldsymbol{R}_1$*"*, onde $\mathbf{r}_1$ é uma instância de $\mathbf{R}_1$ SE ao objeto $\mathbf{o}_v$, removido da visão $\mathbf{V}$.

A Figura 5.59 mostra o trecho do Algoritmo V2 que trata o Caso 2. A seguir descrevemos cada linha do algoritmo.

Linhas 1-3: Se a semântica de atualização escolhida pelo projetista foi "Adicione", então é verificada a existência do objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$, e em seguida é verificada a existência do objeto $\mathbf{r}_2$ de $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE a $\mathbf{o}_v$, removido da visão $\mathbf{V}$.

Linhas 4-5: Caso exista o objeto $\mathbf{r}_1$ (verificado na linha 2) e não exista o objeto $\mathbf{r}_2$ (verificado na linha 3), então $\mathbf{o}_2$ recebe o objeto criado pelo construtor de objetos **Construtor_o**$_2$, o qual cria um objeto do tipo $\mathbf{T}_{R_2}$ SE ao objeto $\mathbf{o}_v$. O construtor de objetos **Construtor_o**$_2$ do tipo $\mathbf{T}_{R_2}$ foi gerado pelo procedimento Cria_Construtor_Objeto_Visao ($\mathbf{T}_{R_2}$, $\mathbf{T}_v$), o qual é apresentado no Apêndice A.

Linha 6: É requisitada a atualização na tabela base, que consiste em adicionar o objeto $\mathbf{o}_2$ do tipo $\mathbf{T}_{R_2}$ na tabela $\mathbf{R}_2$.

Linhas 7-9: Se a semântica de atualização escolhida pelo projetista foi "Remove", então é verificada a existência do objeto $\mathbf{r}_1$ de $\mathbf{R}_1$ tal que $\mathbf{r}_1$ é SE ao objeto $\mathbf{o}_v$, e em

---

Caso 2: A assertiva de correspondência de extensão é da forma: $V \equiv R_1 - R_2$, onde R_1 e R_2 são relações base do banco de dados, cujos tipos são T_{R_1} e T_{R_2}. A semântica de atualização (SA) já foi escolhida pelo projetista, faça:

1. Se SA = "Adicione"então
2. τ : = { $\prec$ Se existir r_1 em R_1 onde $r_1 \equiv o_v$ então
3. Se não existir r_2 em R_2 onde $r_2 \equiv o_v$ então
4. Construtor_o_2 : = Cria_Construtor_Objeto_Visao(T_{R_2},T_v);
5. o_2 : = Construtor_o_2;
6. Adicione o_2 em R_2; $\succ$ }

7. Se SA = "Remove"então
8. τ : = { $\prec$ Se existir r_1 em R_1 onde $r_1 \equiv o_v$ então
9. Se não existir r_2 em R_2 onde $r_2 \equiv o_v$ então
10. Remova r_1 de R_1; $\succ$ }

Figura 5.59. *Caso 2 do Algoritmo V2*

seguida é verificada a existência do objeto $\mathbf{r}_2$ de $\mathbf{R}_2$ tal que $\mathbf{r}_2$ é SE a $\mathbf{o}_v$, removido da visão $\mathbf{V}$.

Linha 10: Caso exista o objeto $\mathbf{r}_1$ (verificado na linha 8) e não exista o objeto $\mathbf{r}_2$ (verificado na linha 9), então é requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{r}_1$ da tabela R_1.

5.5.3. Caso 3 - A ACE é de Intersecção

As ACs de $\mathbf{V}$ com as tabelas bases devem satisfazer as seguintes restrições:

1. A assertiva de correspondência de extensão de $\mathbf{V}$ com as tabelas bases é dada por: $\mathbf{V} \equiv \cap_{i=1}^{n} \mathbf{R}_i$ (Intersecção), onde $\mathbf{R}_i$ é uma relação base do tipo $\mathbf{T}_{R_i}$.
2. A assertiva de correspondência dos objetos de $\mathbf{T}_{R_i}$ com os objetos de $\mathbf{T}_v$ é dada por: $[\mathbf{T}_v,\{\mathbf{a}_1,\mathbf{a}_2, ...,\mathbf{a}_m\}] \equiv [\mathbf{T}_{R_i}, \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m\}]$
   onde $\mathbf{a}_1, \mathbf{a}_2, ... , \mathbf{a}_m$ são atributos de $\mathbf{T}_v$ e $\mathbf{b}_1, \mathbf{b}_2, ... , \mathbf{b}_m$ são atributos de $\mathbf{T}_{R_i}$.

Neste caso, o pedido de atualização, *"Remova* $\boldsymbol{o}_v$ *de* $\boldsymbol{V}$*"*, deve ser traduzido na seguinte atualização no banco de dados: *"Remova* $\boldsymbol{r}_k$ *de* $\boldsymbol{R}_k$ *onde* $\boldsymbol{r}_k \equiv \boldsymbol{o}_v$*"*, onde $\mathbf{r}_k$ é uma instância de $\mathbf{R}_k$ semanticamente equivalente (SE) ao objeto $\mathbf{o}_v$, removido na visão $\mathbf{V}$, 1≤k≤n. Neste caso o projetista será consultado para escolher a relação base $\mathbf{R}_k$ onde será feita a remoção. A Figura 5.60 mostra o trecho do Algoritmo V2 que trata o Caso 3. A seguir descrevemos cada linha do algoritmo.

Caso 3: A assertiva de correspondência de extensão é da forma: V ≡ $\cap_{i=1}^{n}$ R_i, onde R_i é uma relação base do banco de dados. R_k é a relação escolhida pelo projetista para 1≤k≤n, onde será realizada a remoção, faça:

1. Se SA = R_k então
2. τ : = { ≺ Remova r_k de R_k onde $r_k \equiv o_v$; ≻}

Figura 5.60. *Caso 3 do Algoritmo V2*

Linha 1-2: É requisitada a atualização na tabela base, que consiste em remover o objeto $\mathbf{r}_k$ da tabela $\mathbf{R}_k$ onde $\mathbf{r}_k$ é semanticamente equivalente a $\mathbf{o}_v$ e $\mathbf{R}_k$ foi a tabela base escolhida pelo projetista para ser efetuada a atualização.

6

Conclusões

Neste trabalho abordamos o problema de atualização de banco de dados objeto-relacional através de visões de objetos. No nosso enfoque, são gerados tradutores para cada uma das operações de atualização de visões de objetos. Os tradutores são definidos em tempo de projeto e são armazenados juntamente com a definição da visão. Assim, uma vez definido o tradutor, o usuário especifica atualizações através da visão e o tradutor as traduz em atualizações no banco de dados, sem a necessidade de qualquer diálogo adicional.

A principal contribuição deste trabalho são os algoritmos propostos para geração de tradutores para as operações de atualização de visões de objetos em bancos de dados objeto-relacionais. Os algoritmos recebem como entrada o esquema da visão de objetos, o esquema do banco de dados e as assertivas de correspondências (ACs) do esquema da visão com o esquema do banco de dados. No enfoque proposto, só tratamos de visões perservadoras de objetos.

Neste trabalho apresentamos a definição formal dos vários tipos de assertivas de correspondências que são utilizadas para especificar formalmente as correspondências entre o esquema da visão de objetos e o esquema do banco de dados, o qual pode ser relacional ou objeto-relacional. O formalismo proposto permite especificar várias formas de correspondências, inclusive casos onde os esquemas têm estruturas diferentes (heterogeneidade estrutural). As vantagens do uso das ACs é que permitem especificar as condições em que é possível definir um tradutor em tempo de projeto, isto é, não existe ambigüidade a nível de dados, e permitem determinar a seqüência de atualizações necessárias no banco de dados para realizar uma atualização solicitada na visão de objetos. No nosso enfoque o problema de definição do tradutor para uma dada operação de atualização da visão de objetos, consiste em determinar a seqüência de atualizações no banco de dados que são requeridas para a manutenção das assertivas das correspondências "relevantes"para esta operação

Os algoritmos propostos geram tradutores para os seguintes tipos básicos de operações de atualização de visões de objetos em bancos de dados objeto-relacionais: inserção e

remoção de objetos, modificação dos valores de atributos monovalorados e inserção e remoção de objetos em coleções aninhadas.

Os algoritmos C1, C2 geram tradutores para as operações de adição e remoção em uma coleção aninhada da visão. Estes algoritmos tratam visões cuja ACE é de equivalência ou subconjunto. Nestes algoritmos definimos as condições necessárias para garantir que não existe ambiguidade a nivel da dados. É importante ressaltar que atributos multivalorados (coleções aninhadas) podem ser representados de várias formas em um banco de dados objeto-relacional. Os algoritmos C1 e C2 propostos tratam todas estas possibilidades de modelagem. O algoritmo C1 utiliza o procedimento Cria_Construtor_Objeto_Coleção para gerar um construtor de objetos o qual cria uma instância de um tipo base semanticamente equivalente a uma determinada instância de uma coleção aninhada de visão.

Os algoritmos V1, V2 geram tradutores para as operações de adição e remoção em uma visão e o algoritmo V3 gera tradutores para as operações de atualização de atributos monovalorados de visões. Estes algoritmos tratam visões cuja ACE é de equivalência, subconjunto, diferença ou intersecção. Os algoritmos V1 e V2 utilizam o procedimento Cria_Construtor_Objeto_ Visão para gerar um construtor de objetos o qual cria uma instância de um tipo base semanticamente equivalente a uma determinada instância de uma visão.

Os procedimentos Cria_Construtor_Objeto_Coleção e Cria_Construtor_Objeto_Visão recebem como entrada um tipo do esquema da visão $\mathbf{T}_v$ e um tipo base $\mathbf{T}_b$ e usa as ACs de $\mathbf{T}_v$ com $\mathbf{T}_b$ para gerar uma instância de $\mathbf{T}_b$ semanticamente equivalente a uma instância de $\mathbf{T}_v$.

Como trabalhos futuros pretendemos:

- Desenvolver uma ferramenta que implementa os algoritmos propostos para o Banco de Dados Objeto-Relacional Oracle. Pretendemos utilizar a ferramenta para gerar tradutores de atualização de visão para banco de dados com grandes volumes de dados, para verificar o desempenho do uso de visões de objetos;

- Estender este trabalho para tratar o problema de atualização de banco de dados objeto relacional através de visões XML (eXtended Markup Language). Uma atualização da visão XML é simplesmente uma atualização que é especificada em uma visão XML, mas que deve ser traduzida em atualizações a serem executadas na base de dados. Pretendemos utilizar as visões de objetos como interface entre a visão XML e as tabelas relacionais ou objeto relacionais do Banco de Dados [62]. Em [62], o processo de tradução é realizado em dois passos: (i) as atualizações XML submetidas a uma visão XML são traduzidas em atualizações SQL na Visão de Objetos Default (V.O.D), que é uma visão de objeto que possui esquema compatível com o esquema da visão XML, isto é, possui os mesmos tipos e mesma estrutura do esquema da visão XML. Assim, a tradução de atualizações na visão XML em atualizações na visão de objetos default torna-se trivial; (ii) as atualizações na visão de

objeto default são por sua vez traduzidas, pelos tradutores (INSTEAD of triggers) de atualização da visão descritos no nosso enfoque, em atualizações no banco de dados.

- Estender os algoritmos propostos para tratar outros tipos de visões (definidas com novas formas de ACs), assim como tratar os atributos estruturados de tipos da visão que são semanticamente equivalentes a atributos estruturados de tipos base, onde a estrutura do atributo do tipo base é diferente da estrutura do atributo do tipo da visão.

7

Apêndice A

Algoritmo C1: Gera o Tradutor para a Adição de um objeto em uma coleção aninhada.

Esse algoritmo gera o tradutor τ para a operação de adição do objeto $\mathbf{o}_c$ em $\mathbf{o}_v$•**lista_**$\mathbf{T}_c$, onde $\mathbf{o}_c$ é um objeto do tipo $\mathbf{T}_c$ e **lista_**$\mathbf{T}_c$ é uma coleção aninhada do objeto $\mathbf{o}_v$ da visão **V**.
{

Caso 1: No caminho de derivação de **Lista_**$\mathbf{T}_c$ a ligação multivalorada é a última (vide Seção 5.1.1).

Caso 1.1: a ligação multivalorada ℓ_{n-1} é direta.

/* Seleciona ou cria o objeto $\mathbf{o}_n$ */

Se ℓ_{n-1} é uma coleção aninhada de referência

τ : = $\tau \cup \{ \prec \mathbf{o}_n :=$ (Selecione Referência(r_n) de R_n como r_n onde $r_n \equiv o_c$); $\succ \}$

/* $\mathbf{o}_n$ é uma referência a uma instância de $\mathbf{R}_n$ tal que $\mathbf{r}_n \equiv \mathbf{o}_c$ */

senão Se ℓ_{n-1} é uma coleção aninhada de valor estruturado

Construtor_o_n : = Cria_Construtor_Objeto_Coleção($\mathbf{T}_{R_n}$, T_c);

/* Retorna um construtor de objeto do tipo $\mathbf{T}_{R_n}$*/

$\tau := \tau \cup \{ \prec \mathbf{o}_n$: = Construtor_o_n; $\succ \}$

senão /* ℓ_{n-1} é uma coleção aninhada de valor atômico */

$\tau := \tau \cup \{ \prec \mathbf{o}_n$: = $\mathbf{o}_c$; $\succ \}$

/* Seleciona o objeto $\mathbf{o}_{n-1}$ */

Se n=2 /* o caminho φ é composto de apenas uma ligação */

τ : = $\tau \cup \{ \prec \mathbf{o}_{n-1}$:= (Selecione r_1 de R_1 onde $r_1 \equiv o_v$ [1]); $\succ \}$

senão /* n> 2 */

τ : = $\tau \cup \{ \prec \mathbf{o}_{n-1}$:= (Selecione $r_1 \bullet \ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{n-2}$ de R_1 onde $r_1 \equiv o_v$); $\succ \}$

/* $\mathbf{o}_{n-1}$ é uma instância de $\mathbf{R}_{n-1}$, tal que $\mathbf{o}_{n-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet$... $\bullet \ell_{n-2}$ */

/* Gera a atualização */

$\tau := \tau \cup \{ \prec$ Adicione $\mathbf{o}_n$ em $\mathbf{o}_{n-1} \bullet \ell_{n-1}$; $\succ \}$

Caso 1.2: a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa de uma ligação multivalorada ℓ: $\mathbf{T}_{R_n} \rightarrow \mathbf{T}_{R_{n-1}}$.

/* Seleciona o objeto $\mathbf{o}_{n-1}$ */

Se n=2 /* o caminho φ é composto de apenas uma ligação */

τ : = $\tau \cup \{ \prec \mathbf{o}_{n-1}$:= (Selecione Ref(r_1) de R_1 onde $r_1 \equiv o_v$) $\succ \}$;

senão /* n>2 */

τ : = $\tau \cup \{ \prec \mathbf{o}_{n-1}$:= (Selecione $r_1 \bullet \ell_1 \bullet \ell_2 \bullet$ ... $\bullet \ell_{n-2}$ de R_1 onde $r_1 \equiv o_v$); $\succ \}$

/* $\mathbf{o}_{n-1}$ é uma referência a uma instância de $\mathbf{R}_{n-1}$ tal que $\mathbf{o}_{n-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet$... $\bullet \ell_{n-2}$ */

/* Seleciona o objeto $\mathbf{o}_n$ */

τ : = $\tau \cup \{ \prec \mathbf{o}_n$:= (Selecione r_n de R_n as r_n onde $r_n \equiv o_c$); $\succ \}$

/* $\mathbf{o}_n$ é uma instância de $\mathbf{R}_n$ tal que $\mathbf{o}_n \equiv \mathbf{o}_c$ */

/* Gera a atualização */

$\tau := \tau \cup \{ \prec$ Adicione $\mathbf{o}_{n-1}$ em $\mathbf{o}_n \bullet \ell$; $\succ \}$

Caso 1.3: a ligação multivalorada ℓ_{n-1} é virtual, obtida da inversa de uma ligação monovalorada ℓ: $\mathbf{T}_{R_n} \rightarrow \mathbf{T}_{R_{n-1}}$.

/* Cria o objeto $\mathbf{o}_n$ */

Construtor_o$_n$: = Cria_Construtor_Objeto_Coleção($\mathbf{T}_{R_n}$, $\mathbf{T}_c$);

/*Cria um construtor de objeto do tipo $\mathbf{T}_{R_n}$ */

[1] Como definido na seção 3, ($r_1 \equiv o_v$) sss ($o_v \bullet a_1 = r_1 \bullet b_1$ e $o_v \bullet a_2 = r_1 \bullet b_2$ e ... $o_v \bullet a_m = r_1 \bullet b_m$) , onde [$\mathbf{V}$, {$\mathbf{a}_1$, $\mathbf{a}_2$, ..., $\mathbf{a}_m$}] $\equiv$ [$\mathbf{R}_1$, {$\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_m$}] é a assertiva de correspondência dos objetos de $\mathbf{V}$ com os objetos de $\mathbf{R}_1$

$\tau := \tau \cup \{ \prec \mathbf{o}_n : = \text{Construtor\_o}_n; \succ \}$

/* Seleciona o objeto $\mathbf{o}_{n-1}$ */

Se n=2 /* o caminho φ é composto de apenas uma ligação */

$\tau := \tau \cup \{\prec \mathbf{o}_{n-1} := (\text{Selecione } r_1 \text{ de } R_1 \text{ onde } r_1 \equiv o_v); \succ\}$

senão /* n>2 */

$\tau := \tau \cup \{\prec \mathbf{o}_{n-1} := (\text{Selecione } r_1 \bullet \ell_1 \bullet ... \bullet \ell_{n-2} \text{ de } R_1 \text{ onde } r_1 \equiv o_v); \succ\}$

/* $\mathbf{o}_{n-1}$ é uma instância de $\mathbf{R}_{n-1}$, tal que $\mathbf{o}_{n-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-2}$ */

/* Atribui o valor de $\mathbf{o}_n \bullet \ell$ */

Se ℓ é uma ligação de referência

$\tau := \tau \cup \{\prec o_n \bullet \ell := (\text{Selecione Ref}(o_{n-1}); \succ \}$

Senão Se ℓ é uma chave estrangeira, definida por

$\ell = \mathbf{R}_n[\mathbf{a}_1, ..., \mathbf{a}_k] \subset \mathbf{R}_{n-1}[\mathbf{d}_1, ..., \mathbf{d}_k]$

Para j de 1 até k Faça

$\tau := \tau \cup \{\prec o_n \bullet \mathbf{a}_j := \mathbf{o}_{n-1} \bullet \mathbf{d}_j; \succ \}$

/* Gera a atualização */

$\tau := \tau \cup \{ \prec \text{Adicione } \mathbf{o}_n \text{ em } \mathbf{R}_n; \succ \}$

Caso 2: No caminho de derivação de $\mathbf{Lista\_T}_c$ a ligação multivalorada não é a última (vide Seção 5.1.2).

/* Seleciona o objeto $\mathbf{o}_{j-1}$ */

Se n=3 /* o caminho φ é composto de apenas duas ligações */

$\tau := \tau \cup \{ \prec \mathbf{o}_{j-1} := (\text{Selecione } r_1 \text{ de } R_1 \text{ onde } r_1 \equiv o_v) \succ \};$

senão /* n>3 */

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} := (\text{Selecione } r_1 \bullet \ell_1 \ell_2 \bullet ... \bullet \ell_{j-2} \text{ de } R_1 \text{ onde } r_1 \equiv o_v); \succ\}$

/* $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_{j-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ */

Se ℓ é um atributo de referência

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} := (\text{Selecione Ref}(\mathbf{o}_{j-1})); \succ \}$

/* $\mathbf{o}_{j-1}$ é uma referência a uma instância de $\mathbf{R}_{j-1}$ */

/* Seleciona o objeto $\mathbf{o}_{j+1}$ */

Se j=(n-1) /* ℓ_j é a última ligação do caminho φ */

$\tau := \tau \cup \{\prec \mathbf{o}_{j+1} :=$ (Selecione r_n de R_n onde $r_n \equiv o_c$); $\succ\}$

senão /* j<(n-1) */

$\tau := \tau \cup \{\prec \mathbf{o}_{j+1} :=$ (Selecione $\mathbf{r}_{j+1}$ de $\mathbf{R}_{j+1}$ onde $\mathbf{r}_{j+1} \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1} \equiv o_c$); $\succ\}$

/* $\mathbf{o}_{j+1}$ é uma instância de $\mathbf{R}_{j+1}$, tal que $\mathbf{o}_{j+1}$ está relacionado com $\mathbf{r}_n$ de $\mathbf{R}_n$, onde $\mathbf{r}_n \equiv \mathbf{o}_c$, através do caminho $\ell_{j+1} \bullet \ell_{j+2} \bullet ... \bullet \ell_{n-1}$ */

/* Cria o objeto $\mathbf{o}_j$ */

Se ℓ e ℓ_j são atributos de referência então

$\tau := \tau \cup \{\prec \mathbf{o}_j := \mathbf{T}_j(\ell$: Ref($\mathbf{o}_{j-1}$), ℓ_j: Ref($\mathbf{o}_{j+1}$)); $\succ\}$

Senão /* ℓ e ℓ_j são chaves estrangeiras definidas por

ℓ: $\mathbf{R}_j$ [$\mathbf{c}_1$,..., $\mathbf{c}_k$]$\subset$ $\mathbf{R}_{j-1}$ [$\mathbf{d}_1$,..., $\mathbf{d}_k$] e

ℓ_j: $\mathbf{R}_j$ [$\mathbf{c}_{k+1}$,..., $\mathbf{c}_m$]$\subset$ $\mathbf{R}_{j+1}$ [$\mathbf{e}_{k+1}$,..., $\mathbf{e}_m$]*/

Para i de 1 até k faça:

$\tau := \tau \cup \{\prec \mathbf{v}_i = \mathbf{o}_{j-1} \bullet \mathbf{d}_i; \succ\}$

Para i de k+1 até m faça:

$\tau := \tau \cup \{\prec \mathbf{v}_i = \mathbf{o}_{j+1} \bullet \mathbf{e}_i; \succ\}$

$\tau := \tau \cup \{\prec \mathbf{o}_j := \mathbf{T}_j(\mathbf{c}_1{:}\mathbf{v}_1, ..., \mathbf{c}_m{:}\mathbf{v}_m); \succ\}$

/* Gera a atualização */

$\tau := \tau \cup \{ \prec$ Adicione $\mathbf{o}_j$ em $\mathbf{R}_j; \succ \}$

Retorne (τ);

}/* fim do algoritmo C1 */

Algoritmo C2: Gera o Tradutor para a Remoção de um objeto de uma coleção aninhada.

Esse algoritmo gera o tradutor τ para a operação de remoção do objeto $\mathbf{o}_c$ de $\mathbf{o}_v \bullet \mathbf{lista\_T}_c$, onde $\mathbf{o}_c$ é um objeto do tipo $\mathbf{T}_c$ e $\mathbf{lista\_T}_c$ é uma coleção aninhada do objeto $\mathbf{o}_v$ da visão $\mathbf{V}$.
{
A visão $\mathbf{V}$ satisfaz as condições definidas na Seção 5.2

<u>**Caso 1**</u>: a ligação multivalorada ℓ_{j-1} é direta.

/* Seleciona o objeto $\mathbf{o}_{j-1}$ */

Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} :=$ (Selecione r_1 de R_1 onde $r_1 \equiv o_v$); $\succ\}$

senão /* j>2 */

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} :=$ (Selecione $r_1 \bullet \ell_1 \bullet\ \ell_2 \bullet ... \bullet \ell_{j-2}$ de R_1 onde $r_1 \equiv o_v$); $\succ\}$

/* $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_{j-1}$ está relacionado com a instância $\mathbf{r}_1$ de $\mathbf{R}_1$ ($\mathbf{r}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1 \bullet \ell_2 \bullet ...\ \bullet \ell_{j-2}$ */

/* Gera a atualização */

Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */

$\tau := \tau \cup \{\prec$ Remova o_j de $o_{j-1} \bullet \ell_{j-1}$ onde $o_j \equiv o_c$; $\succ\}$

senão /* j<n */

$\tau := \tau \cup \{\prec$ Remova o_j de $o_{j-1} \bullet \ell_{j-1}$ onde $o_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \equiv o_c$; $\succ\}$

/* $\mathbf{o}_j$ é uma instância de $o_{j-1} \bullet \ell_{j-1}$, tal que $\mathbf{o}_j$ está relacionado com o objeto $\mathbf{r}_n$ de $\mathbf{R}_n$ ($\mathbf{r}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j \bullet \ell_{j+1} \bullet ...\ \bullet \ell_{n-1}$ */

<u>**Caso 2**</u>: a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa de uma ligação multivalorada ℓ: $\mathbf{T}_{R_j} \rightarrow \mathbf{T}_{R_{j-1}}$.

/* Seleciona o objeto $\mathbf{o}_j$ */

Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */

$\tau := \tau \cup \{\prec \mathbf{o}_j :=$ (Selecione r_n de R_n onde $r_n \equiv o_c$); $\succ\}$

senão /* j<n */

$\tau := \tau \cup \{\prec \mathbf{o}_j :=$(Selecione $\mathbf{r}_j$ de $\mathbf{R}_j$ onde $\mathbf{r}_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \equiv o_c$);$\succ\}$

/* $\mathbf{o}_j$ é uma instância de $\mathbf{R}_j$, tal que $\mathbf{o}_j$ está relacionado com a instância $\mathbf{r}_n$ de $\mathbf{R}_n$ ($\mathbf{r}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$ */

/* Gera a atualização */

Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */

$\tau := \tau \cup \{\prec$ Remova $\mathbf{o}_{j-1}$ de $\mathbf{o}_j \bullet \ell$ onde $\mathbf{o}_{j-1} \equiv \mathbf{o}_v$; $\succ\}$

senão /* j>2 */

$\tau := \tau \cup \{\prec \mathbf{o}_1 :=$ (Selecione $\mathbf{r}_1$ de $\mathbf{R}_1$ onde $\mathbf{r}_1 \equiv \mathbf{o}_v$);$\succ\}$

$\tau := \tau \cup \{\prec$ Remova $\mathbf{o}_{j-1}$ de $\mathbf{o}_j \bullet \ell$ onde $\mathbf{o}_{j-1} \equiv \mathbf{o}_1 \bullet \ell_1 \bullet ... \bullet \ell_{j-2}$;$\succ\}$

/* $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{o}_j \bullet \ell$, tal que $\mathbf{o}_{j-1}$ está relacionado com a instância $\mathbf{o}_1$ de $\mathbf{R}_1$ ($\mathbf{o}_1 \equiv \mathbf{o}_v$) através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ */

<u>Caso 3</u>: a ligação multivalorada ℓ_{j-1} é virtual, obtida da inversa de uma ligação monovalorada ℓ: $\mathbf{T}_{R_j} \rightarrow \mathbf{T}_{R_{j-1}}$.

/* Seleciona o objeto $\mathbf{o}_{j-1}$ */

Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} :=$ (Selecione r_1 de R_1 onde $r_1 \equiv o_v$); $\succ\}$

senão /* j>2 */

$\tau := \tau \cup \{\prec \mathbf{o}_{j-1} :=$ (Selecione $r_1 \bullet \ell_1 \bullet \ \ell_2 \bullet ... \bullet \ell_{j-2}$ de R_1 onde $r_1 \equiv o_v$); $\succ\}$

/* $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_{j-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ */

/* Gera a atualização */

Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */

$\tau := \tau \cup \{\prec$ Remova $\mathbf{o}_j$ de $\mathbf{R}_j$ onde $\mathbf{o}_j \bullet \ell = \mathbf{o}_{j-1}$ e $\mathbf{o}_j \equiv \mathbf{o}_c$; $\succ\}$

senão /* j<n */

$\tau := \tau \cup \{\prec$ Remova $\mathbf{o}_j$ de $\mathbf{R}_j$ onde $\mathbf{o}_j \bullet \ell = \mathbf{o}_{j-1}$ e $\mathbf{o}_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \equiv \mathbf{o}_c$;$\succ\}$

/* $\mathbf{o}_j$ é uma instância de $\mathbf{R}_j$, tal que $\mathbf{o}_j$ está relacionado com o objeto $\mathbf{o}_n$ de $\mathbf{R}_n$ ($\mathbf{o}_n \equiv \mathbf{o}_c$) através do caminho $\ell_j \bullet \ell_{j+1} \bullet ... \bullet \ell_{n-1}$ e $\mathbf{o}_j \bullet \ell = \mathbf{o}_{j-1}$*/

Retorne (τ);
}/* fim do algoritmo C2 */

Algoritmo V3: Gera o tradutor para a modificação de atributo monovalorado de um objeto de visão.

Esse algoritmo gera o tradutor τ para a operação de atualização do atributo monovalorado $\mathbf{a}_v$ de $\mathbf{o}_v$, onde $\mathbf{o}_v$ é um objeto da visão $\mathbf{V}$.
{
Caso 1: A ACC é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \mathbf{a}$

$\tau := \tau \cup \{ \prec \mathbf{o}_1 :=$ (Selecione $\mathbf{r}_1$ de $\mathbf{R}_1$ onde $\mathbf{r}_1 \equiv \mathbf{o}_v$);

Atribua $\mathbf{o}_1 \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$; $\succ$ }

Caso 2: A ACC é do tipo $\mathbf{T}_v \bullet \mathbf{a}_v \equiv \mathbf{T}_{R_1} \bullet \varphi$

/* Seleciona o objeto $\mathbf{o}_{j-1}$*/

Se j=2 /* ℓ_{j-1} é a primeira ligação do caminho φ */

$\tau := \tau \cup \{ \prec \mathbf{o}_{j-1} :=$ (Selecione $\mathbf{r}_1$ de $\mathbf{R}_1$ onde $\mathbf{r}_1 \equiv \mathbf{o}_v$); $\succ$ }

senão /* 2<j≤n*/

$\tau := \tau \cup \{ \prec \mathbf{o}_{j-1} :=$ (Selecione $\mathbf{r}_1 \bullet \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ de $\mathbf{R}_1$ ($\mathbf{r}_1$)

onde $\mathbf{r}_1 \equiv \mathbf{o}_v$);$\succ$}

/* $\mathbf{o}_{j-1}$ é uma instância de $\mathbf{R}_{j-1}$, tal que $\mathbf{o}_{j-1}$ está relacionado com $\mathbf{r}_1$ através do caminho $\ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{j-2}$ */

/* Seleciona o objeto $\mathbf{o}_j$*/

Se j=n /* ℓ_{j-1} é a última ligação do caminho φ */

$\tau := \tau \cup \{ \prec \mathbf{o}_j :=$ (Selecione $\mathbf{r}_n$ de R_n onde $r_n \bullet \mathbf{a} = o_v \bullet \mathbf{a}_v$); $\succ$ }

senão /* 2≤j<n */

$\tau := \tau \cup \{ \prec \mathbf{o}_j :=$ (Selecione $\mathbf{r}_j$ de $\mathbf{R}_j$ onde $\mathbf{r}_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$);$\succ$}

/* $\mathbf{o}_j$ é uma instância de $\mathbf{R}_j$, tal que $\mathbf{o}_j \bullet \ell_j \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{a}_v$ */

/* Gera a atualização */

Se ℓ_{j-1} é um atributo de valor

$\tau := \tau \cup \{ \prec$ Atribua $\mathbf{o}_{j-1} \bullet (\ell_{j-1}) = \mathbf{o}_j$;$\succ$}

Senão se ℓ_{j-1} é um atributo de referência

$\tau := \tau \cup \{ \prec$ Atribua $\mathbf{o}_{j-1} \bullet (\ell_{j-1}) =$ Ref($\mathbf{o}_j$);$\succ$}

Senão /* ℓ_{j-1} é uma chave estrangeira definida por

ℓ_{j-1}: $\mathbf{R}_{j-1}$ [$\mathbf{c}_1$,..., $\mathbf{c}_k$]$\subset$ $\mathbf{R}_j$ [$\mathbf{d}_1$,..., $\mathbf{d}_k$]*/

Para i de 1 até k faça:

$\tau := \tau \cup \{\prec$ Atribua $\mathbf{o}_{j-1}\bullet\mathbf{c}_i = \mathbf{o}_j\bullet\mathbf{d}_i; \succ \}$

Retorne (τ);
}/* fim do algoritmo V3 */

Algoritmo V1: Gera o Tradutor para a Adição de Objetos em uma Visão.

Esse algoritmo gera o tradutor τ para a operação de adição do objeto $\mathbf{o}_v$ do tipo $\mathbf{T}_v$ na visão $\mathbf{V}$.
{
$\tau := \emptyset$;

<u>**Caso1**</u>: A assertiva de correspondência é da forma $\mathbf{V} \equiv \mathbf{R}_1$ ou da forma $\mathbf{V} \subset \mathbf{R}_1$, onde $\mathbf{R}_1$ é uma relação base do banco de dados, cujo tipo é $\mathbf{T}_{R_1}$, faça:

Construtor_o := Cria_Construtor_Objeto_Visao($\mathbf{T}_{R_1}$, $\mathbf{T}_v$);
/* Retorna um construtor de objeto do tipo $\mathbf{T}_{R_1}$*/
τ := { $\prec$ **o** := **Construtor_o**;
Adicione **o** em $\mathbf{R}_1$; $\succ$}
Para cada atributo multivalorado **Lista_**$\mathbf{T}_c$ de $\mathbf{T}_v$, onde $\mathbf{T}_v\bullet$**Lista_**$\mathbf{T}_c \equiv \mathbf{T}_{R_1}\bullet\varphi$, faça:

$\tau := \tau \cup \{\prec$ Para cada objeto $\mathbf{o}_c$ em $\mathbf{o}_v\bullet$**lista_**$\mathbf{T}_c$, faça:

Adicione $\mathbf{o}_c$ em $\mathbf{o}_v\bullet$**Lista_**$\mathbf{T}_c$; $\succ$ }

<u>**Caso2**</u>: A assertiva de correspondência é da forma $\mathbf{V} \equiv \mathbf{R}_1$ - $\mathbf{R}_2$, onde $\mathbf{R}_1$ e $\mathbf{R}_2$ são relações base do banco de dados, cujos tipos são $\mathbf{T}_{R_1}$ e $\mathbf{T}_{R_2}$, faça:

τ : = { $\prec$ Se existir $\mathbf{r}_1$ em $\mathbf{R}_1$ tal que $\mathbf{r}_1 \equiv \mathbf{o}_v$ então

Se existir $\mathbf{r}_2$ em $\mathbf{R}_2$ tal que $\mathbf{r}_2 \equiv \mathbf{o}_v$ então

Remova $\mathbf{r}_2$ de $\mathbf{R}_2$;

senão $\succ$}

construtor_o$_1$:= Cria_Construtor_Objeto_Visao($\mathbf{T}_{R_1}$, $\mathbf{T}_v$);

/* Retorna um construtor de objeto do tipo $\mathbf{T}_{R_1}$*/

$\tau := \tau \cup \{\prec$ $\mathbf{o}_1$:= **construtor_o**$_1$;

Se existir $\mathbf{r}_2$ em $\mathbf{R}_2$ tal que $\mathbf{r}_2 \equiv \mathbf{o}_v$ então

Adicione $\mathbf{o}_1$ em $\mathbf{R}_1$;

Remova $\mathbf{r}_2$ de $\mathbf{R}_2$;

senão

Adicione $\mathbf{o}_1$ em $\mathbf{R}_1$; $\succ$}

<u>**Caso 3**</u>: A assertiva de correspondência é da forma $\mathbf{V} \equiv \cap_{i=1}^{n} \mathbf{R}_i$, onde $\mathbf{R}_i$ é uma relação base do banco de dados, cujo tipo é $\mathbf{T}_{R_i}$, faça:

τ : = { $\prec$ Para cada $\mathbf{R}_i$ faça: $\succ$}

Construtor_o$_i$:=Cria_Construtor_Objeto_Visao($\mathbf{T}_{R_i}$, $\mathbf{T}_v$);

/* Retorna um construtor de objetos do tipo $\mathbf{T}_{R_i}$ */

$\tau := \tau \cup$ { $\prec$ $\mathbf{o}_i$: = **Construtor_o**$_i$;

Adiciona $\mathbf{o}_i$ em $\mathbf{R}_i$; $\succ$}

Retorne (τ);
}/* fim do algoritmo V1 */

Algoritmo V2: Gera o Tradutor para a Remoção de Objetos de uma Visão.

Esse algoritmo gera o tradutor τ para a operação de remoção do objeto $\mathbf{o}_v$ (do tipo $\mathbf{T}_v$) da visão $\mathbf{V}$.

{
$\tau := \emptyset$;

Caso 1: A assertiva de correspondência de extensão é da forma $\mathbf{V} \equiv \mathbf{R}$ ou da forma $\mathbf{V} \subset \mathbf{R}$, onde $\mathbf{R}$ é uma relação base do banco de dados faça:

$\tau := \{ \prec$ Remova $\mathbf{r}$ de $\mathbf{R}$ onde $\mathbf{r} \equiv \mathbf{o}_v; \succ \}$

Caso 2: A assertiva de correspondência de extensão é da forma $\mathbf{V} \equiv \mathbf{R}_1$ - $\mathbf{R}_2$, onde $\mathbf{R}_1$ e $\mathbf{R}_2$ são relações base do banco de dados, cujos tipos são $\mathbf{T}_{R_1}$ e $\mathbf{T}_{R_2}$. A semântica de atualização (SA) já foi escolhida pelo projetista, faça:

Se SA = "Adicione"então
$\tau := \{ \prec$ Se existir $\mathbf{r}_1$ em $\mathbf{R}_1$ onde $\mathbf{r}_1 \equiv \mathbf{o}_v$ então
Se não existir $\mathbf{r}_2$ em $\mathbf{R}_2$ onde $\mathbf{r}_2 \equiv \mathbf{o}_v$ então
Construtor_o$_2$:= Cria_Construtor_Objeto_Visao($\mathbf{T}_{R_2}$,$\mathbf{T}_v$);
$\mathbf{o}_2$:= **Construtor_o**$_2$;
Adicione $\mathbf{o}_2$ em $\mathbf{R}_2$; $\succ \}$

Se SA = "Remove"então
$\tau := \{ \prec$ Se existir $\mathbf{r}_1$ em $\mathbf{R}_1$ onde $\mathbf{r}_1 \equiv \mathbf{o}_v$ então
Se não existir $\mathbf{r}_2$ em $\mathbf{R}_2$ onde $\mathbf{r}_2 \equiv \mathbf{o}_v$ então
Remova $\mathbf{r}_1$ de $\mathbf{R}_1$; $\succ \}$

Caso 3: A assertiva de correspondência de extensão é da forma $\mathbf{V} \equiv \cap_{i=1}^{n} \mathbf{R}_i$, onde $\mathbf{R}_i$ é uma relação base do banco de dados. $\mathbf{R}_k$ é a relação escolhida pelo projetista para 1≤k≤n, onde será realizada a remoção, faça:

Se SA = $\mathbf{R}_k$ então

$\tau := \{ \prec$ Remova $\mathbf{r}_k$ de $\mathbf{R}_k$ onde $\mathbf{r}_k \equiv \mathbf{o}_v; \succ\}$

Retorne (τ);

```
}/* fim do algoritmo V2 */
```

Cria_Construtor_Objeto_Coleção ($\mathbf{T}_1$: tipo base, $\mathbf{T}_c$: tipo da coleção aninhada)

/* Este procedimento recebe como entrada os tipos $\mathbf{T}_1$ e $\mathbf{T}_c$, onde $\mathbf{T}_1$ é um tipo base e $\mathbf{T}_c$ é o tipo de uma coleção aninhada, e gera o construtor de objetos do tipo $\mathbf{T}_1$. O construtor gerado cria um objeto de $\mathbf{T}_1$ semanticamente equivalente a um dado objeto $\mathbf{o}_c$ de $\mathbf{T}_c$.

Para gerar um construtor de objetos, que cria um objeto $\mathbf{o}_1$ de $\mathbf{T}_1$ a partir de um objeto $\mathbf{o}_c$ de $\mathbf{T}_c$, deve-se determinar valores para os atributos de $\mathbf{o}_1$ a partir de $\mathbf{o}_c$. Os valores dos atributos de $\mathbf{o}_1$ são determinados a partir das assertivas de correspondência entre os tipos $\mathbf{T}_1$ e $\mathbf{T}_c$, os quais são semanticamente relacionados. Neste algoritmo considere que o "pai"do objeto $\mathbf{o}_c$ é o objeto $\mathbf{o}_v$ do tipo $\mathbf{T}_v$. */

Declare

i : integer;

Reg1 : Registro

atributo : varchar2(30),

valor : varchar2(2000) ;

end;

v [2] : Array[1..n] of Reg1;

Begin

i:=1;

Para cada atributo **b** de $\mathbf{T}_c$, tal que existe uma ACC ψ relacionando **b** com um atributo ou caminho de $\mathbf{T}_1$ faça:

Caso1: **b** é monovalorado

Caso1.1: **b** é de valor

Caso1.1.1: **b** é atômico

Caso1.1.1.1: ψ é da forma: $\mathbf{T}_c\bullet\mathbf{b} \equiv \mathbf{T}_1\bullet\mathbf{a}$, onde **a** é um atributo de $\mathbf{T}_1$.

v(i).atributo := **a**;

v(i).valor := $\mathbf{o}_c\bullet\mathbf{b}$;

Caso1.1.1.2: ψ é da forma: $\mathbf{T}_c\bullet\mathbf{b} \equiv \mathbf{T}_1\bullet\varphi$, onde $\varphi = \ell_1\bullet\ell_2\bullet...\bullet\ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e

[2] A variável **v** é um array(n) do tipo **Reg1**, cuja estrutura é composta dos atributos **atributo** e **valor**. **v(i)**.atributo irá receber os nomes dos atributos de $\mathbf{T}_1$ e **v(i)**.valor irá armazenar os valores dos atributos para criar uma instância de $\mathbf{T}_1$

$\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para 1≤i≤n.

v(i).atributo := ℓ_1;

v(i).valor := (Selecione $\mathbf{r}_2$ de $\mathbf{R}_2$ onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a} = \mathbf{o}_c \bullet \mathbf{b}$);

Caso1.1.2: **b** é estruturado

Caso1.1.2.1: ψ é da forma: $\mathbf{T}_c \bullet \mathbf{b} \equiv \mathbf{T}_1 \bullet \varphi$, onde $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e $\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para 1≤i≤n.

Se ℓ_1 é de referência então

v(i).atributo := ℓ_1;

v(i).valor :=(Selecione Referência($\mathbf{r}_2$) de $\mathbf{R}_2$ ($\mathbf{r}_2$) onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \equiv \mathbf{o}_c \bullet \mathbf{b}$ [3]);

senão

Se ℓ_1 é uma chave estrangeira definida por $\ell_1 = \mathbf{R}_1[\mathbf{a}_1,...,\mathbf{a}_k] \subset \mathbf{R}_2[\mathbf{d}_1,...,\mathbf{d}_k]$, onde $\mathbf{T}_1$, $\mathbf{T}_2$ são os tipos dos objetos das relações $\mathbf{R}_1$, $\mathbf{R}_2$.

Para j de 1 até k Faça

v(i).atributo := $\mathbf{a}_j$;

v(i).valor := (Selecione $r_2 \bullet \mathbf{d}_j$ de R_2 (r_2) onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \equiv \mathbf{o}_c \bullet \mathbf{b}$);

i:= i+1;

j:= j+1;

Fim Para;

senão /* ℓ_1 é um atributo de valor */

v(i).atributo := ℓ_1;

v(i).valor := Cria_Construtor_Objeto_Coleção($\mathbf{T}_{\ell_1}$, $\mathbf{T}_b$);

/* $\mathbf{T}_{\ell_1}$ é o tipo de ℓ_1 e $\mathbf{T}_b$ é o tipo de **b** */

Caso1.1.2.2: ψ é da forma: $[\mathbf{T}_c \bullet \mathbf{b}, \{ \mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m \}] \equiv [\mathbf{T}_1, \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}]$, onde $\{ \mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m\}$ são atributos de $\mathbf{T}_b$ e $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$ são atributos de $\mathbf{T}_1$.

Para j de 1 até m Faça

v(i).atributo := $\mathbf{a}_i$;

v(i).valor := $\mathbf{o}_c.\mathbf{b}.\mathbf{c}_j$;

j=j+1;

i:=i+1;

[3]Utilizamos a ACO para fazer o matching entre os objetos dos tipos $\mathbf{T}_n$ e $\mathbf{T}_b$, onde $\mathbf{T}_b$ é o tipo de **b**

Caso1.2: **b** é de referência, onde ψ é da forma: $\mathbf{T}_c\bullet\mathbf{b} \equiv \mathbf{T}_1\bullet\varphi$, onde $\varphi = \ell_1\bullet\ell_2\bullet...\bullet\ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e $\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para $1 \leq i \leq n$.

Se ℓ_1 é de referência então

v(i).atributo := ℓ_1;

v(i).valor := (Selecione Referência($\mathbf{r}_2$) de $\mathbf{R}_2$ ($\mathbf{r}_2$) onde $\mathbf{r}_2\bullet\ell_2\bullet...\bullet\ell_{n-1} = \mathbf{o}_c\bullet\mathbf{b}$)

senão

ℓ_1 é uma chave estrangeira definida por $\ell_1 = \mathbf{R}_1[\mathbf{a}_1,...,\mathbf{a}_k] \subset \mathbf{R}_2[\mathbf{d}_1,...,\mathbf{d}_k]$,

onde $\mathbf{T}_1$, $\mathbf{T}_2$ são os tipos dos objetos das relações $\mathbf{R}_1$, $\mathbf{R}_2$.

Para j de 1 até k Faça

v(i).atributo := $\mathbf{a}_j$;

v(i).valor := (Selecione $r_2\bullet\mathbf{d}_j$ de R_2 (r_2) onde $\mathbf{r}_2\bullet\ell_2\bullet...\bullet\ell_{n-1} \equiv \mathbf{o}_c\bullet\mathbf{b}$);

i:= i+1;

j:= j+1;

Fim Para;

Caso2: **b** é multivalorado

Caso2.1: **b** é de valor

Caso2.1.1: **b** é atômico

Caso2.1.1.1: ψ é da forma: $\mathbf{T}_c\bullet\mathbf{b} \equiv \mathbf{T}_1\bullet\mathbf{a}$, onde **a** é um atributo multivalorado de $\mathbf{T}_1$, cujo de valor atômico .

v(i).atributo := **a**;

v(i).valor := $\mathbf{o}_c$.**b**;

Caso2.1.1.2: ψ é da forma: $\mathbf{T}_c\bullet\mathbf{b} \equiv [\mathbf{T}_1,\{a1, a2, ..., am\}]$, onde $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$ são atributos de $\mathbf{T}_1$.

Para j de 1 até m Faça

v(i).atributo := $\mathbf{a}_i$;

v(i).valor := $\mathbf{o}_c$.**b**[j];

j=j+1;

i:=i+1;

Caso2.1.2: **b** é estruturado[4]

Null;

Caso2.2: **b** é de referência

Null;
i:=i+1;

Retorne ($\mathbf{T}_1$(v(1).atributo:v(1).valor, v(2).atributo:v(2).valor, ..., v(n).atributo: v(n).valor))[5];
End;
}/* Fim do algoritmo Cria_Construtor_Objeto_Coleção */

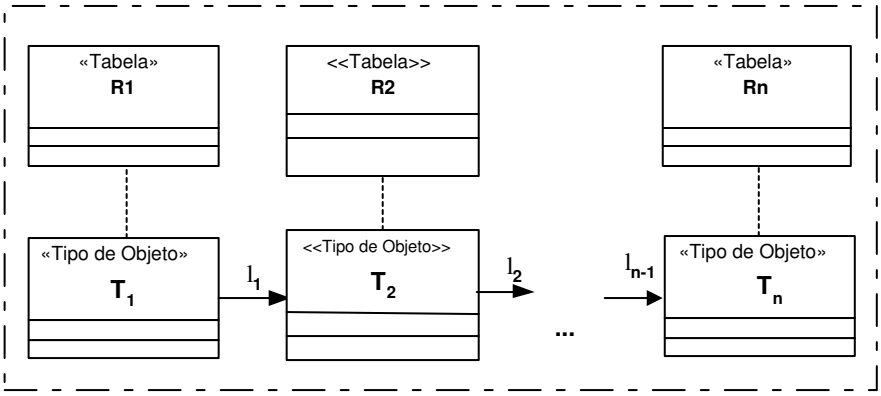

Figura 7.1. *Caminho* $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ *do tipo base* $\boldsymbol{T}_1$

[4]Tratamos a adição em atributos multivalorados (do tipo estruturado ou referência) com dois ou mais níveis de aninhamento através de uma nova chamada de adição na coleção desejada.

[5]Este algoritmo retorna um construtor de objetos do tipo $\mathbf{T}_1$, cujo atributos (v(1).atributo,..., v(n).atributo) são formados pelos respectivos valores (v(1).valor,..., v(n).valor).

Cria_Construtor_Objeto_Visao ($\mathbf{T}_1$: tipo base, $\mathbf{T}_v$: tipo da visão)

/* Este procedimento recebe como entrada os tipos $\mathbf{T}_1$ e $\mathbf{T}_v$, onde $\mathbf{T}_1$ é um tipo base e $\mathbf{T}_v$ é um tipo da visão, e gera o construtor de objetos do tipo $\mathbf{T}_1$. O construtor gerado cria um objeto de $\mathbf{T}_1$ semanticamente equivalente a um dado objeto $\mathbf{o}_v$ de $\mathbf{T}_v$.

Para gerar um construtor de objetos, que cria um objeto $\mathbf{o}_1$ de $\mathbf{T}_1$ a partir de um objeto $\mathbf{o}_v$ de $\mathbf{T}_v$, deve-se determinar valores para os atributos de $\mathbf{o}_1$ a partir de $\mathbf{o}_v$. Os valores dos atributos de $\mathbf{o}_1$ são determinados a partir das assertivas de correspondência entre os tipos $\mathbf{T}_1$ e $\mathbf{T}_v$, os quais são semanticamente relacionados. Este formalismo permite definir o mapeamento entre os objetos de tipos com estruturas diferentes. */

Declare

i : integer;

Reg1 : Registro

atributo : varchar2(30),

valor : varchar2(2000) ;

end;

v [6] : Array[1..n] of Reg1;

Begin

i:=1;

Para cada atributo **b** de $\mathbf{T}_v$, tal que existe uma ACC ψ relacionando **b** com um atributo ou caminho de $\mathbf{T}_1$ faça:

Caso1: **b** é monovalorado

Caso1.1: **b** é de valor

Caso1.1.1: **b** é atômico

Caso1.1.1.1: ψ é da forma: $\mathbf{T}_v \bullet \mathbf{b} \equiv \mathbf{T}_1 \bullet \mathbf{a}$, onde **a** é um atributo de $\mathbf{T}_1$.

v(i).atributo := **a**;

v(i).valor := $\mathbf{o}_v \bullet \mathbf{b}$;

Caso1.1.1.2: ψ é da forma: $\mathbf{T}_v \bullet \mathbf{b} \equiv \mathbf{T}_1 \bullet \varphi$, onde $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e

[6] A variável **v** é um array(n) do tipo **Reg1**, cuja estrutura é composta dos atributos **atributo** e **valor**. **v(i)**.atributo irá receber os nomes dos atributos de $\mathbf{T}_1$ e **v(i)**.valor irá armazenar os valores dos atributos para criar uma instância de $\mathbf{T}_1$

$\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para 1≤i≤n.

v(i).atributo := ℓ_1;

v(i).valor := (Selecione $\mathbf{r}_2$ de $\mathbf{R}_2$ onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \bullet \mathbf{a} = \mathbf{o}_v \bullet \mathbf{b}$);

Caso1.1.2: **b** é estruturado

Caso1.1.2.1: ψ é da forma: $\mathbf{T}_v \bullet \mathbf{b} \equiv \mathbf{T}_1 \bullet \varphi$, onde $\varphi = \ell_1 \bullet \ell_2 \bullet ... \bullet \ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e $\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para 1≤i≤n.

Se ℓ_1 é de referência então

v(i).atributo := ℓ_1;

v(i).valor :=(Selecione Referência($\mathbf{r}_2$) de $\mathbf{R}_2$ ($\mathbf{r}_2$) onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \equiv \mathbf{o}_v \bullet \mathbf{b}$ [7]);

senão

Se ℓ_1 é uma chave estrangeira definida por $\ell_1 = \mathbf{R}_1[\mathbf{a}_1,...,\mathbf{a}_k] \subset \mathbf{R}_2[\mathbf{d}_1,...,\mathbf{d}_k]$, onde $\mathbf{T}_1$, $\mathbf{T}_2$ são os tipos dos objetos das relações $\mathbf{R}_1$, $\mathbf{R}_2$.

Para j de 1 até k Faça

v(i).atributo := $\mathbf{a}_j$;

v(i).valor := (Selecione $r_2 \bullet \mathbf{d}_j$ de R_2 (r_2) onde $\mathbf{r}_2 \bullet \ell_2 \bullet ... \bullet \ell_{n-1} \equiv \mathbf{o}_v \bullet \mathbf{b}$);

i:= i+1;

j:= j+1;

Fim Para;

senão /* ℓ_1 é um atributo de valor */

v(i).atributo := ℓ_1;

v(i).valor := Cria_Construtor_Objeto_Visao($\mathbf{T}_{\ell_1}$, $\mathbf{T}_b$);

/* $\mathbf{T}_{\ell_1}$ é o tipo de ℓ_1 e $\mathbf{T}_b$ é o tipo de **b** */

Caso1.1.2.2: ψ é da forma: $[\mathbf{T}_v \bullet \mathbf{b}, \{ \mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m \}] \equiv [\mathbf{T}_1, \{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}]$, onde $\{ \mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m\}$ são atributos de $\mathbf{T}_b$ e $\{\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_m\}$ são atributos de $\mathbf{T}_1$.

Para j de 1 até m Faça

v(i).atributo := $\mathbf{a}_i$;

v(i).valor := $\mathbf{o}_v.\mathbf{b}.\mathbf{c}_j$;

j=j+1;

i:=i+1;

[7] Utilizamos a ACO para fazer o matching entre os objetos dos tipos $\mathbf{T}_n$ e $\mathbf{T}_b$, onde $\mathbf{T}_b$ é o tipo de **b**

Caso1.2: **b** é de referência, ψ é da forma: $\mathbf{T}_v\bullet\mathbf{b} \equiv \mathbf{T}_1\bullet\varphi$, onde $\varphi = \ell_1\bullet\ell_2\bullet...\bullet\ell_{n-1}$ é um caminho de $\mathbf{T}_1$ (vide Figura 7.1), tal que ℓ_i é uma ligação definida por ℓ_i: $\mathbf{T}_i \rightarrow \mathbf{T}_{i+1}$ e $\mathbf{T}_i$ é o tipo dos objetos de $\mathbf{R}_i$, para $1 \leq i \leq n$.

Se ℓ_1 é de referência então

v(i).atributo := ℓ_1;

v(i).valor := (Selecione Referência($\mathbf{r}_2$) de $\mathbf{R}_2$ ($\mathbf{r}_2$) onde $\mathbf{r}_2\bullet\ell_2\bullet...\bullet\ell_{n-1} = \mathbf{o}_v\bullet\mathbf{b}$);

senão

ℓ_1 é uma chave estrangeira definida por $\ell_1 = \mathbf{R}_1[\mathbf{a}_1,...,\mathbf{a}_k] \subset \mathbf{R}_2[\mathbf{d}_1,...,\mathbf{d}_k]$,

onde $\mathbf{T}_1$, $\mathbf{T}_2$ são os tipos dos objetos das relações $\mathbf{R}_1$, $\mathbf{R}_2$.

Para j de 1 até k Faça

v(i).atributo := $\mathbf{a}_j$;

v(i).valor := (Selecione $r_2\bullet\mathbf{d}_j$ de R_2 (r_2) onde $\mathbf{r}_2\bullet\ell_2\bullet...\bullet\ell_{n-1} \equiv \mathbf{o}_v\bullet\mathbf{b}$);

i:= i+1;

j:= j+1;

Fim Para;

Caso2: **b** é multivalorado[8]

Null;

i:=i+1;
Fim Para;

Retorne ($\mathbf{T}_1$(v(1).atributo:v(1).valor, v(2).atributo:v(2).valor, ..., v(n).atributo: v(n).valor));
End;
}/* Fim do algoritmo Cria_Construtor_Objeto_Visao */
---

[8]Tratamos a adição em atributos multivalorados através de uma chamada de adição na coleção aninhada, a qual é realizada no tratutor de adição em visões.

Referências Bibliográficas

[1] S. W. Ambler, "The design of a robust persistence layer for relational databases," *Software Development*, Agosto 1998.

[2] S. W. Ambler, "Mapping objects to relational databases," *Software Development*, Outubro 1999.

[3] F. Bancilhon and N. Spyratos, "Updates semantics and relational view," *ACM Transactions on Database Systems*, vol. 6, Dezembro 1993.

[4] T. Barsalou, A. M. Keller, N. Siambela, and G. Wiederhold, "Updating relational databases through object-based views," *SIGMOD Conference*, pp. 248–257, 1991.

[5] K. Brown and B. G. Whitenack, "Crossing chasms - a pattern language for object-rdbms integration," 1995.

[6] U. Dayal and A. Bernstein, "On the correct translation of update operations on relational views," *ACM Transactions on Database Systems*, vol. 7, Setembro 1982.

[7] T. Kwong and R. Tsang, "Dbgen manual v1.0," *http://dbgen.sourceforge.net.*

[8] A. L. Furtado and M. A. Casanova, "Updating relational views. query processing in database system," *New York: Ed. Springer Verlag*, 1985.

[9] M. L. Fussell, "Foundations of object relational mapping," Julho 1997.

[10] C. Hamon and A. M. Keller, "Two-level caching of composite object views of relational databases," *International Conference on Data Engineering*, pp. 428–437, 1995.

[11] A. M. Keller and G. Wiederhold, "Penguim: Objects for programs, relations for persistence," pp. 75–88, Julho 2000.

[12] A. M. Keller and C. Hamon, "A c++ binding for penguin: a system for data sharing among heterogeneous object models," *Foundations of Data Organization and Algorithms*, pp. 215–230, 1993.

[13] A. M. Keller, R. Jensen, and S. Agarwal, "Persistence software: Bridging object-oriented programming and relational databases," *SIGMOD Conference*, pp. 523–528, 1993.

[14] S. Agarwal, C. Keene, and A. M. Keller, "Arquitecting object applications for high performance with relational databases," Agosto 1995.

[15] A. M. Keller, "The role of semantics in translating view updates," *IEEE Computer*, vol. vol.19, pp. 63–73, Janeiro 1986.

[16] R. Lajoie and R. K. Keller, "Design and reuse in object-oriented frameworks: Patterns, contracts, and motifs in concert," *Congress of the Association Canadienne Française pour L'Avancement des Sciences*, vol. 62, Maio 1994.

[17] J. A. Larson and A. P. Sheth, "Updating relational views using knowledge at view definition and view update time," *Information Systems*, vol. 16, no. 2, pp. 145–168, 1991.

[18] B. S. Lee and G. Wiederhold, "Outer joins and filters for instantiating from relational databases through views," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, pp. 108–119, 1994.

[19] A. de Lucena Lermen and C. A. Heuser, "Um framework para mapeamento de aplicações orientadas a objetos a banco de dados relationais/objeto-relacionais," *Semana Acadêmica UFRGS*, 1998.

[20] T. Ling and M. L. Lee, "View update in entity-relationship approach," *Data and Knowledge Engineering*, vol. 19, pp. 135–169, 1996.

[21] S. McClure, "Object database vs. object-relational databases," *IDC Bulletin 14821E*, Agosto 1997.

[22] C. Medeiros and F. W. Tompa, "Understanding the implications of view update policies," *In Proc. of 11th International Conference on Very Large Databases*, 1985.

[23] O. Corporation, "Oracle8i application developer's guide - fundamentals," vol. Release 8.1.5 - A68003-01.

[24] O. Corporation, "Oracle8i application developer's guide - object-relational features," vol. Release 2 (8.1.6) - A76976-01.

[25] O. Corporation, "Pl/sql - user's guide and reference," vol. Release 8.1.5 - A67842-01, Fevereiro 1999.

[26] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman, "Medmaker: A mediator system based on declarative specifications," *International Conference on Data Engineering*, pp. 132–141, Fevereiro 1996.

[27] Poet, "User's guide - poet sql object factory," vol. Poet6.1.

[28] B. Pribyl and S. Feuerstein, "Manking sense of object technology in pl/sql 8," *Oracle Open World*, 1997.

[29] J. Robie and D. Bartels, "A comparison between relational and object oriented databases for object oriented application development,"

[30] W. Technologies, "Relational object framework 1.0 programmer's guide," *Java Beans Edition*.

[31] W. Technologies, "Relational object framework 1.0 tutorial," *Java Beans Edition*, 1998.

[32] S. Spaccapietra and C. Parent, "View integration: A step forward in solving structural conflicts," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, Abril 1994.

[33] Sun, "http://www.sun.com/software/javablend/,"

[34] T. Takahashi and A. M. Keller, "Implementation of object view query on a relational database,"

[35] P. Turner and A. M. Keller, "Reflections on object-relational applications," Setembro 1995.

[36] G. Wiederhold, T. Barsalou, B. S. Lee, N. Siambela, and W. Sujansky, "Use of relational storage and a semantic model to generate objects: The penguin project," *Database'91: Merging Policy, Standards and Technology*, pp. 503–515, 1991.

[37] G. Wiederhold, "Views, objects, and databases," *On Object-Oriented Database Systems*, pp. 29–44, 1991.

[38] G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Computer Magazine*, Março 1992.

[39] P. Brown, *Object-Relational Database Development - A Plumber's Guide*. Informix Press, December 2000.

[40] P. Dorsey and J. R. Hudicka, *Oracle8 Design Using UML Object Modeling*. Oracle Press, November 1998.

[41] C. Larman, *Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design*. 1998.

[42] K. Loney and G. Koch, *Oracle8i: The Complete Reference*. Oracle Press, May 2000.

[43] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*.

[44] M. Kaufman, *The object database standard: ODMG 2.0*. 1997.

[45] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Sistema de Banco de Dados*. Makron Books, terceira edição ed.

[46] M. Stonebraker, *Object-Relational DBMS - The Next Great Wave*. Morgen Kaufmann Publishers, 1996.

[47] M. Stonebraker and P. Brown, *Object-Relational DBMS - Tracking the Next Great Wave*. Morgen Kaufmann Publishers.

[48] G. Guerrini, E. Bertino, B. Catania, and J. Garcia-Molina, "A formal model of views for object-oriented database systems," in *Theory and Practice of Object Systems*, 1997.

[49] D. C. Hay, "Using data model patterns for rapid application development," in *International Oracle User Week*, 1996.

[50] R. Motschnig-Pitrik, "Requirements and comparison of view mechanisms for object-oriented databases," in *Information Systems*, vol. 21, pp. 229–252, 1996.

[51] A. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, andautonomous databases," in *ACM Computing Surveys, v.22, n.3, pp.190*, 1990.

[52] A. Silberschatz, M. Stonebraker, and J. F. Ullman, "Database systems: Achievements and opportunities," in *Communications of the ACM, n.34*, 1991.

[53] S. Spaccapietra, C. Parent, and E. Zimanyi, "Modeling time from a conceptual perspective," in *Proceedings of the International Conference on Information and Knowledge Management*, pp. 432–440, 1998.

[54] I. Tatarinovi, Z. Ives, A. Y. Halevy, and D. Weld, "Updating xml," In Proceedings of SIGMOD 2001, Maio 2001.

[55] V. M. P. Vidal and M. Winslett, "Specifying view update translation in schema integration," in *Proceedings of The Third International Conference on Information and Knowledge Management*, 1994.

[56] V. M. P. Vidal and M. Winslett, "A rigorous approach to schema restructuring," in *Proceedings of The 14th International Conference on Object-Oriented and Entity-Relationship Modeling*, 1995.

[57] V. M. P. Vidal, "Definindo traduções corretas de atualizações de visões na presença de efeitos colaterais," in *XI Simpósio Brasileiro de Banco de Dados*, Outubro 1996.

[58] V. M. P. Vidal and B. F. Lóscio, "Especificação de mediadores para acesso e atualização de múltiplas bases de dados," in *Proceedings of XII Simpósio Brasileiro de Banco de Dados*, 1997.

[59] W. W. W. Consortium, "Xquery 1.0: An xml query language," W3C Working Draft, 2002.

[60] J. Zendulka, "Object-relational modeling in uml," in *Proceedings of the Conference Information Systems Modelling*, pp. 17–24, MARQ, 2001.

[61] B. F. Lóscio, "Atualização de múltiplas bases de dados através de mediadores," Master's thesis, Universidade Federal do Ceará, 1998.

[62] W. G. C. Oliveira, "Atualizando banco de dados objeto-relacional através de visões xml (em andamento)," Master's thesis, Universidade Federal do Ceará, 2002.

[63] V. M. Pequeno, "Auto-manutenção de classes de fusão em visões de integração de dados," Master's thesis, Universidade Federal do Ceará, 2000.

[64] R. M. de Figueirêdo Vilas Boas, "Xmls+ matcher: Um método para matching de xml schemas semânticos," Master's thesis, Universidade Federal do Ceará, Agosto 2002.

[65] A. M. Keller, *Updating Relational Databases Through Views*. PhD thesis, Stanford University, Junho 1995.