Geração Automática de Visões de Objeto de Dados Relacionais

Vânia Vidal, Lineu Santos, Valdiana Araujo, Fernando Lemos

Departamento de Computação – Universidade Federal do Ceará (UFC)
Fortaleza – CE – Brazil
{vvidal,lineu,valdiana,fernandocl}@lia.ufc.br

Abstract. *Views are used to publish data from a database into a new structure or schema. The definition of object views over relational data allows this data to be published into an object oriented structure, without the need for changing the database schema. This paper proposes a three-step approach to generate object views of relational data. First, the user defines the view schema. Second, the user identifies the view correspondence assertions which formally specify the relationships between the view schema and the base relational schema. A simple graphical interface helps the user define the view correspondence assertions. Finally, based on the view assertions, it is generated the SQL query that carries out the mapping defined by the assertions. In this work, we present an algorithm that automatically generates the SQL view definition.*

Resumo. *Visões são usadas para publicar dados de uma base de dados em uma nova estrutura ou esquema. A definição de Visões de Objeto sobre dados relacionais permite que esses dados possam ser disponibilizados em uma estrutura orientada a objetos, sem que seja necessário mudar o esquema da base de dados. Neste trabalho, propomos um enfoque de três passos para geração de visões de objeto sobre base de dados relacional. Primeiro, o usuário define o esquema da visão. Em seguida, o usuário define as Assertivas de Correspondência que especificam formalmente os relacionamentos entre o esquema da base de dados e o esquema da visão. Uma interface gráfica facilita o processo de edição do esquema da visão e suas assertivas. O terceiro passo, então, produz a consulta SQL que realiza o mapeamento definido pelas assertivas. Neste trabalho, apresentamos um algoritmo que gera automaticamente a definição SQL da visão baseado nas assertivas de correspondência da visão.*

1. Introdução

Na arquitetura de três-níveis de esquemas, as visões constituem as interfaces através das quais os usuários consultam e atualizam a base de dados. Além das vantagens já conhecidas das visões relacionais, as visões de objeto [4, 7, 12] permitem a coexistência e o compartilhamento de dados armazenados em base de dados relacional com diversas aplicações usando tecnologia de objeto. Assim, visões de objeto definidas sobre dados relacionais provêem uma técnica poderosa para se impor uma ou mais visões lógicas, ricamente estruturadas, sobre base de dados já existente. Dessa forma, dados relacionais podem ser disponibilizados em uma estrutura orientada a objetos, sem que seja

necessário mudar o esquema da base de dados. Outra vantagem de visões de objeto é que os dados resultantes de uma consulta a uma visão de objeto podem ser automaticamente publicados como um documento XML [3, 8, 10], usando tecnologias como XSQL Pages [7], permitindo assim a integração de aplicações *Web* [1, 2] com bases de dados relacionais.

Para criar uma visão de objeto, primeiro deve-se criar o tipo dos objetos da visão. Depois, então, define-se a visão através de uma consulta SQL, a qual especifica como os objetos da visão são sintetizados a partir dos dados da base de dados. Definir um consulta que realize o mapeamento de tuplas de tabelas relacionais em objetos de tipos com estrutura complexa é tarefa que exige conhecimentos avançados de SQL:1999 [7, 13]. No caso de modificações no esquema da base de dados, as visões de objeto precisam ser redefinidas. Atualmente, a criação e a manutenção das visões de objeto são feitas manualmente, e não temos conhecimento de nenhum trabalho ou ferramenta que auxilie na geração e manutenção de visões de objeto.

Neste trabalho propomos um enfoque de três passos para geração de visões de objeto sobre bases de dados relacionais. Primeiro, o usuário define o esquema da visão. Em seguida, o usuário define as Assertivas de Correspondência (ACs) que especificam formalmente o mapeamento [6, 8, 9] entre o esquema da visão e o esquema da base de dados. O terceiro passo, então, produz a consulta SQL que realiza o mapeamento definido pelas ACs. As principais contribuições do artigo são as seguintes:

- Propomos um formalismo que permite especificar o mapeamento de tuplas de tabelas relacionais em objetos de tipos complexos. Com base no formalismo proposto, especificamos formalmente as condições sobre as quais um conjunto de ACs especifica completamente uma visão de objetos em termos de um esquema relacional e, neste caso, mostramos que o mapeamento definido pelas ACs pode ser traduzido diretamente em uma definição SQL da visão. É importante salientar que outros formalismos de mapeamentos propostos ou são ambíguos [5] ou requerem que o usuário declare mapeamentos lógicos complexos [14], os quais não podem ser definidos graficamente. Como mostramos adiante, o formalismo proposto lida com o problema da heterogeneidade semântica [13], e que permite que mapeamentos complexos possam ser especificados de forma simples.
- Apresentamos o algoritmo GeraConstrutorSQL que, baseado nas ACs da visão, gera o construtor SQL que constrói os objetos da visão a partir de tuplas de tabelas relacionais conforme o mapeamento especificado pelas ACs.
- Propomos a ferramenta VBA (View-By-Assertions) para facilitar a tarefa de criação de visões de objeto. Uma seção típica com VBA começa com o usuário definindo, através de uma interface gráfica, o tipo dos objetos da visão. O usuário deve então carregar o esquema da base de dados e, através de uma interface gráfica, definir as ACs do esquema da visão com o esquema da base de dados. Com base no conjunto de ACs da visão, VBA gera automaticamente a consulta SQL, que realiza o mapeamento definido pelas ACs.

Este artigo está organizado como se segue. Seção 2 discute os principais elementos do esquema Objeto-Relacional do SGBD Oracle 9i. Seção 3 apresenta o formalismo de mapeamento proposto. Seção 4 cobre o uso de assertivas para especificação de visões de objeto, e apresenta a interface gráfica da ferramenta VBA

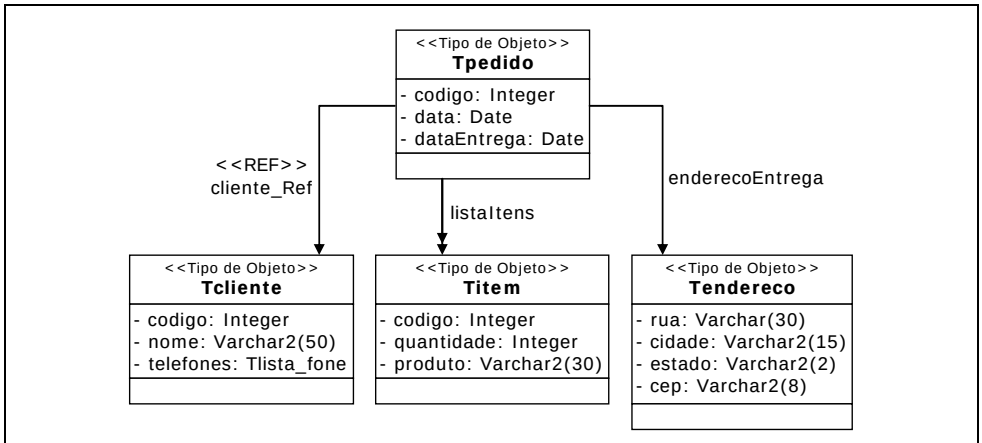
para entrada das assertivas. Seção 5 apresenta o algoritmo GeraConstrutorSQL. Finalmente, Seção 6 apresenta as conclusões.

2. Esquema Objeto-Relacional

Um esquema objeto-relacional é uma tripla S=(T, R, I), onde T é um conjunto de definições de tipos, R é um conjunto de tabelas e I é um conjunto de restrições de integridade. Uma das principais características do modelo objeto-relacional é a capacidade de estender o sistema de tipos do SGBD, ou seja, novos tipos podem ser definidos pelos usuários. Esses tipos são chamados de tipos abstratos de dados (TAD). Um tipo serve como molde para a criação de objetos (instâncias do tipo). Um tipo define a estrutura de dados (atributos) e as operações (métodos) comuns às instâncias do tipo. Nesse trabalho nos baseamos no SGBD Objeto-Relacional Oracle 9i.

O Oracle 9i suporta dois tipos de tabela: tabela de tuplas e tabela de objetos. As tabelas de tuplas são as tabelas do modelo relacional. Uma tabela de objetos possui associado um tipo, e os objetos inseridos na tabela possuem um identificador único (OID), permitindo, assim, que os objetos possam ser referenciados por outros objetos através de atributos de tipo referência, como ilustramos a seguir.

Considere, por exemplo, o esquema objeto-relacional EOR na Figura 1. A tabela clientes é uma tabela de objetos, cujos objetos são instâncias do tipo $T_{cliente}$. Os atributos de um tipo podem ser classificados em *monovalorados* ou *multivalorados*. Um atributo monovalorado pode ter tipo *atômico*, *estruturado* ou *referência*. Um atributo multivalorado tem tipo coleção cujos objetos podem ter tipo *atômico*, *estruturado* ou




```
CREATE TYPE Tlista_fone AS VARRAY(10) OF
      VARCHAR(15);
CREATE TYPE Tcliente AS OBJECT(
  codigo INTEGER,
  nome VARCHAR2(50),
  telefones Tlista_fone);
CREATE TYPE Titem AS OBJECT(
  codigo INTEGER,
  quantidade INTEGER,
  produto VARCHAR2(30));
CREATE TYPE Tendereco AS OBJECT(
  rua VARCHAR2(30),
  cidade VARCHAR2(15),
  estado VARCHAR2(2),
  cep VARCHAR2(8) );
```

```
CREATE TYPE Tlista_item AS TABLE OF Titem;
CREATE TYPE Tpedido AS OBJECT(
  codigo INTEGER,
  data DATE,
  dataEntrega DATE,
  enderecoEntrega Tendereco,
  cliente_ref REF Tcliente,
  listaItens Tlista_item);
CREATE TABLE Clientes OF Tcliente(
   codigo PRIMARY KEY );
CREATE TABLE Pedidos OF Tpedido(
  codigo PRIMARY KEY,
  cliente_ref REFERENCES Clientes)
 NESTED TABLE listaItens STORE AS listaItens_nt_tab;
```

Figura 1. Esquema Objeto-Relacional EOR

referência. Em um tipo referência, o tipo dos objetos referenciados pode ser um tipo atômico ou estruturado. Na representação gráfica da Figura 1, usa-se setas simples para atributos monovalorados, setas duplas para atributos multivalorados, e <<REF>> indica um atributo de referência, que é um atributo de tipo referência ou coleção de referência. No Oracle 9i, um atributo multivalorado pode ser implementado como uma *Nested Table* ou *Varray*. No esquema da Figura 1, temos que: o atributo nome do tipo $T_{cliente}$ é um atributo monovalorado de tipo atômico; o atributo enderecoEntrega do tipo T_{pedido} é um atributo monovalorado de tipo estruturado $T_{endereco}$; o atributo listaItens do tipo T_{pedido} é um atributo multivalorado (*nested table*) de tipo coleção T_{lista_item}, cujos objetos têm tipo estruturado T_{item}; o atributo cliente_ref do tipo T_{pedido} é um atributo monovalorado de tipo REF $T_{cliente}$ (referência para uma instância do tipo $T_{cliente}$). O escopo de um atributo de referência (tabela ou visão que contém os objetos referenciados) pode ser especificado através de restrições referenciais ou restrições de escopo [7]. Considere, por exemplo, a tabela Pedidos no esquema EOR da Figura 1. A restrição referencial "cliente_ref REFERENCES Clientes" especifica que o escopo do atributo de referência cliente_ref é a tabela Clientes.

3. Terminologia

Sejam R_1 e R_2 tabelas de uma base de dados relacional. Seja fk: $R_1[a_1,...,a_n] \subseteq R_2[b_1,...,b_n]$ uma chave estrangeira de R_1 que referencia R_2. Então, temos: (i) fk é uma *ligação* de R_1 para R_2 e para qualquer tupla r_1 de R_1 temos $r_1.fk = \{ r_2 \mid r_2 \in R_2 \text{ e } r_1.a_i = r_2.b_i,\ 1 \le i \le n \}$; e (ii) a ligação inversa de fk, denotada por fk^{-1}, é uma ligação de R_2 para R_1, e para qualquer tupla r_2 de R_2 temos $r_2.fk^{-1} = \{ r_1 \mid r_1 \in R_1 \text{ e } r_1.a_i = r_2.b_i,\ 1 \le i \le n \}$.

Seja ℓ uma ligação R_1 para R_2, e r_1 e r_2, tuplas de R_1 e R_2, respectivamente. Se $r_2 \in r_1.\ell$ então r_1 *referencia* r_2 *através de* ℓ. ℓ é uma *ligação monovalorada* se uma tupla de R_1 pode referenciar no máximo uma tupla de R_2 através de ℓ, e *multivalorada* se uma tupla de R_1 pode referenciar várias tuplas de R_2 através de ℓ.

Sejam $R_1,...,R_n$ tabelas de uma base de dados relacional, tal que existe uma ligação ℓ_i de R_i para R_{i+1} onde ℓ_i é uma chave estrangeira dada por $R_i[a_1^{\ell_i},...,a_{m_i}^{\ell_i}] \subseteq R_{i+1}[b_1^{\ell_i},...,b_{m_i}^{\ell_i}]$ ou inversa da chave estrangeira dada por $R_{i+1}[b_1^{\ell_i},...,b_{m_i}^{\ell_i}] \subseteq R_i[a_1^{\ell_i},...,a_{m_i}^{\ell_i}]$, $1 \le i \le n-1$. Então, $\varphi = \ell_1. \ ... \ .\ell_{n-1}$ é um *caminho* de R_1 para R_n, e para qualquer tupla r_1 de R_1 temos $r_1.\varphi = \{ r_n \mid \exists r_2 \in R_2 \wedge ... \wedge \exists r_n \in R_n \wedge r_i.a_k^{\ell_i} = r_{i+1}.b_k^{\ell_i},\ 1 \le k \le m_i,\ 1 \le i \le n-1 \}$. Assim, temos que as tuplas de R_1 referenciam tuplas de R_n através do caminho φ. Se as ligações $\ell_1,...,\ell_{n-1}$ são monovaloradas, então φ é um *caminho monovalorado*, caso contrário φ é um *caminho multivalorado*.

Uma *assertiva de correspondência* é uma expressão da forma $[T/\ c] \equiv [R/\ exp]$ onde c é um atributo do tipo T, R é uma tabela de uma base de dados relacional, e *exp* pode ter uma das seguintes formas:

(i) a, onde a é um atributo de R;

(ii) φ, onde φ é um caminho de R;

(iii) φ / a, onde φ é um caminho de R para R', e a é um atributo de R';

(iv) $\{a_1,...,a_n\}$, onde $a_1,...,a_n$ são atributos de R;

(v) φ / $\{a_1,...,a_n\}$, onde φ é um caminho de R para R', e $a_1,...,a_n$ são atributos de R';

(vi) NULL.

No resto desta seção, seja T um tipo, R uma tabela de uma base de dados relacional, e $\mathcal{A}$ um conjunto de Assertivas de Correspondência.

Dizemos que $\mathcal{A}$ *especifica* T *em termos de* R *sss* para qualquer atributo c de T onde c tem tipo T_c, existe uma assertiva de correspondência ψ em $\mathcal{A}$, tal que:

(i) Se T_c é um tipo atômico então ψ tem uma das seguintes formas: [T/ c] ≡ [R/ a], ou [T/ c] ≡ [R/ φ/ a], onde φ é um caminho monovalorado.

(ii) Se T_c é um tipo coleção de objetos de tipo atômicos então ψ tem uma das seguintes formas: [T/ c] ≡ [R/ φ/ a] onde φ é um caminho multivalorado, [T/ c] ≡ [R/ $\{a_1,...,a_n\}$], ou [T/ c] ≡ [R/ φ/ $\{a_1,...,a_n\}$] onde φ é um caminho monovalorado.

(iii) Se T_c é um tipo referência então ψ é dada por [T/ c] ≡ [R/ φ] onde φ é um caminho monovalorado.

(iv) Se T_c é um tipo estruturado então ψ é dada por [T/ c] ≡ [R/ φ] onde φ é um caminho monovalorado, ou [T/ c] ≡ [R/ NULL].

(v) Se T_c é um tipo coleção de objetos de tipo estruturado ou referência então ψ é dada por [T/ c] ≡ [R/ φ], onde φ é um caminho multivalorado.

Dizemos que $\mathcal{A}$ *especifica completamente* T *em termos de* R *sss* $\mathcal{A}$ especifica T em termos de R, e para qualquer assertiva em $\mathcal{A}$ da forma [T/ c] ≡ [R/ φ] tal que φ é um caminho de R para R', e c tem tipo T_c, ou REF T_c, ou coleção de objetos de tipo T_c ou REF T_c, onde T_c é um tipo estruturado, então $\mathcal{A}$ especifica T_c em termos de R'.

Seja $\mathcal{A}$ um conjunto de assertivas de correspondência tal que $\mathcal{A}$ especifica completamente T em termos de R. Seja t uma instância de T e r uma tupla de R. Dizemos que t é *semanticamente equivalente a* r *como especificado por* $\mathcal{A}$, denotado t $\equiv_{\mathcal{A}}$ r, sss para qualquer atributo c de T tal que T_c é o tipo de c, então:

(i) Se T_c é um tipo atômico e [T/ c] ≡ [R/ a] então t.c = r.a;

(ii) Se T_c é um tipo atômico e [T/ c] ≡ [R/ φ/ a] então t.c = r'.a, onde r'∈r.φ;

(iii) Se T_c é um tipo coleção de objetos de tipo atômico e [T/ c] ≡ [R/ $\{a_1,...,a_n\}$] então t.c = $\{r.a_1,...,r.a_n\}$;

(iv) Se T_c é um tipo coleção de objetos de tipo atômico e [T/ c] ≡ [R/ φ/ $\{a_1,...,a_n\}$] então t.c = $\{r'.a_1,...,r'.a_n\}$ onde r'∈r.φ;

(v) Se T_c é um tipo coleção de objetos de tipo atômico e [T/ c] ≡ [R/ φ/ a], então t.c = {o | o=r'.a onde r'∈r.φ};

(vi) Se T_c é um tipo referência para objetos do tipo T_v (REF T_v) onde [T/ c] ≡ [R/ φ] então t.c = REF(v), onde v $\equiv_{\mathcal{A}}$ r' e r'∈r.φ. (REF(v) retorna o identificador de v)

(vii) Se T_c é um tipo coleção de referências para objetos do tipo T_v, e [T/ c] ≡ [R/ φ], então t.c = {REF(v) | v $\equiv_{\mathcal{A}}$ r' onde r'∈r.φ};

(viii) Se T_c é um tipo estruturado e [T/ c] ≡ [R/ φ], então t.c $\equiv_{\mathcal{A}}$ r', onde r'∈r.φ;

(ix) Se T_c é um tipo coleção de objetos de tipo estruturado e [T/ c] ≡ [R/ φ], então t.c = {o | o $\equiv_{\mathcal{A}}$ r' onde r'∈r.φ};

(x) Se T_c é um tipo estruturado e [T/ c] ≡ [R/ NULL], então t.c $\equiv_{\mathcal{A}}$ r.

Observe que em (ii), (iv), (vi) e (viii), o caminho φ é monovalorado, de forma que r.φ é um conjunto vazio ou unitário (|r.φ|≤1), e no caso em que r.φ é vazio, então t.c tem valor nulo (t.c = NULL). Nos casos (v), (vii) e (ix) onde φ é multivalorado, se r.φ é vazio então o valor de t.c é uma coleção vazia (t.c = ∅).

4. Uso de Assertivas para Especificar Visões de Objeto

Seja S um esquema relacional. Uma *visão de objeto* sobre S é uma quádrupla V=<T,C,R,$\mathcal{A}$>, onde T é o tipo dos objetos da visão, C é o conjunto de tipos aninhados[I] de T, R é uma tabela ou visão relacional em S, denominada tabela(visão) pivô, e $\mathcal{A}$ é um conjunto de assertivas que especificam completamente T em termos de R. As assertivas de correspondência da visão especificam de forma axiomática como os objetos da visão são sintetizados a partir das tuplas da tabela pivô. Como mostraremos a seguir, as ACs da visão especificam completamente a visão V em termos do esquema base S, no sentido que define um mapeamento funcional, denominado DEFv, que associa com cada estado σ_S de S um estado da visão DEFv(σ_S). O estado da visão V em um estado σ_S é de S é dado por: DEFv(σ_S) = { t | $\exists r \in R$ e $t \equiv_{\mathcal{A}} r$}.

A definição da visão V em SQL é dada por:

```
CREATE VIEW V OF T
 WITH OBJECT IDENTIFIER (k1,...,km) AS
 SELECT τ [R→T](r) FROM R r.
```

Na cláusula SELECT, a função construtora τ[R→T] constrói uma instância t de T a partir de uma tupla r de R, tal que $t \equiv_{\mathcal{A}} r$. Suponha que o tipo T tem atributos $c_1,...,c_m$. Então, o construtor τ[R→T](r) tem a seguinte forma: $T(Q_{c1}(r),...,Q_{cm}(r))$, onde $Q_{ci}(r)$ é uma subconsulta SQL que gera o valor do atributo c_i de t conforme especificado pela assertiva de correspondência de c_i, para $1 \le i \le m$. Na Seção 4 apresentamos o Algoritmo GeraConstrutorSQL que recebe como entrada um tipo T, uma relação R e um conjunto $\mathcal{A}$ de ACs que especificam completamente T em termos de R, e gera a função construtora τ[R→T] tal que dada uma tupla r de R, τ[R→T](r) constrói uma instância t de T tal que $t \equiv_{\mathcal{A}} r$. O identificador da visão especificado na cláusula WITH OBJECT IDENTIFIER é composto dos atributos de V que têm correspondência com os atributos da chave primária da tabela pivô R. Assim dado que $\{d_1,...,d_m\}$ é a chave primária de R, e $[T/ k_i] \equiv [R/ d_i]$, $1 \le i \le m$, então $\{k_1,...,k_m\}$ é o identificador de V.

Neste trabalho só tratamos de visões preservadoras de objeto [4], de forma que existe um mapeamento 1-1 entre as tuplas da tabela(visão) pivô e os objetos da visão. Assim como em [4], para outros tipos de visão, deve-se criar uma visão relacional de forma que exista um mapeamento 1-1 entre as tuplas da visão relacional e os objetos da visão de objeto. Assim, a visão de objeto é definida sobre a visão relacional criada.

O processo de criação de uma visão de objeto consiste de três passos e é suportada pela ferramenta VBA. (i) Primeiro, o usuário define o esquema da visão o qual consiste do tipo dos objetos da visão e demais tipos usados na definição do tipo da

---

[I] T´ é um tipo aninhado do tipo T sss T' é o tipo de um atributo de T ou T' é o tipo de um atributo a, onde a é um atributo de um tipo aninhado de T.

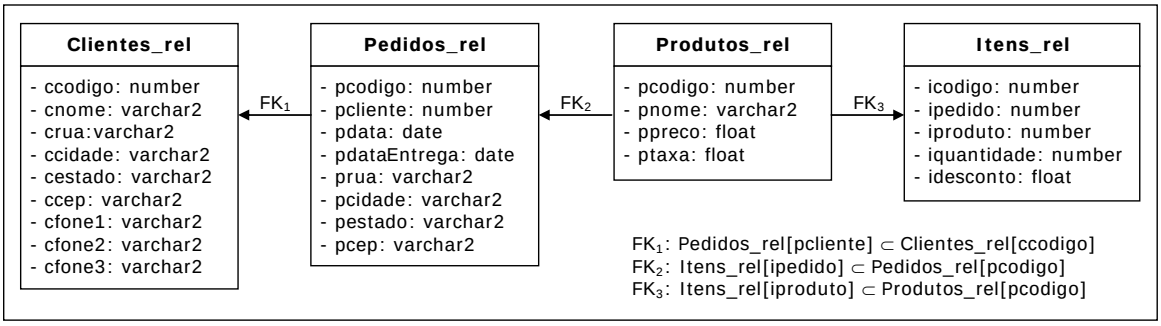


Figura 2. Esquema Relacional ER

visão. No caso de um atributo de tipo de referência, deve-se especificar o escopo dos objetos referenciados, o qual pode ser uma tabela de objetos ou visão de objetos; (ii) em seguida, o usuário deve escolher a tabela pivô e as ACs que especificam completamente o tipo da visão em termos da tabela pivô; (iii) no terceiro passo, é gerada a consulta SQL que realiza o mapeamento definido pelas ACs.

A seguir, como estudo de caso, considere o esquema relacional ER, mostrado na Figura 2. Suponha que o usuário deseja definir sobre o esquema ER as seguintes visões de objeto: a visão Pedidos_v de tipo T_{pedido} e a visão Clientes_v de tipo $T_{cliente}$, onde T_{pedido} e $T_{cliente}$ são tipos do esquema EOR na Figura 1. Considere que na visão Pedidos_v o escopo do atributo cliente_ref é a visão Clientes_v, e que as tabelas pivô de Pedidos_v e Clientes_v são as tabelas Pedidos_rel e Clientes_rel do esquema ER.

É importante notar que a criação das visões Pedidos_v e Clientes_v permite o desenvolvimento de aplicações orientadas a objeto como se Pedidos_v e Clientes_v fossem tabelas de objeto. Dessa forma, os dados armazenados na estrutura do esquema relacional ER são disponibilizados na estrutura do esquema objeto-relacional EOR sem que seja necessário mudar o esquema da base de dados.

Figura 3 mostra as ACs que especificam completamente o tipo T_{pedido} em termos de Pedidos_rel. A definição das ACs é realizada através de uma interface gráfica da ferramenta VBA. Para isso, VBA exibe uma tela contendo, em formato de árvore de diretório, a estrutura do esquema da visão e a estrutura do esquema da base de dados conforme mostrado na Figura 4. Observe que, no esquema da visão, o tipo de cada atributo é especificado entre parênteses, e no esquema base, além dos atributos das tabelas, são mostradas também as ligações de chaves estrangeiras e inversas de chave

ACs de T_{pedido} & Pedidos_rel
ψ_1: [T_{pedido}/ codigo] ≡ [Pedidos_rel/ pcodigo]
ψ_2: [T_{pedido}/ data] ≡ [Pedidos_rel/ pdata]
ψ_3: [Tpedido/ dataEntrega] ≡ [Pedidos_rel/ pdataEntrega]
ψ_4: [T_{pedido}/ enderecoEntrega] ≡ [Pedidos_rel/ NULL]
ψ_5: [T_{pedido}/ cliente_ref] ≡ [Pedidos_rel/ fk_1]
ψ_6: [T_{pedido}/ listaItens] ≡ [Pedidos_rel/ fk_2^{-1}]

ACs de $T_{endereco}$ & Pedidos_rel
ψ_7: [$T_{endereco}$/ rua] ≡ [Pedidos_rel/ prua]
ψ_8: [$T_{endereco}$/ cidade] ≡ [Pedidos_rel/ pcidade]
ψ_9: [$T_{endereco}$/ estado] ≡ [Pedidos_rel/ pestado]
ψ_{10}: [$T_{endereco}$/ cep] ≡ [Pedidos_rel/ pcep]

ACs de T_{item} & Itens_rel
ψ_{11}: [T_{item}/ codigo] ≡ [Itens_rel/ icodigo]
ψ_{12}: [T_{item}/ quantidade] ≡ [Itens_rel/ quantidade]
ψ_{13}: [T_{item}/ produto] ≡ [Itens_rel/ fk_3.pnome]

ACs de $T_{cliente}$ & Clientes_rel
ψ_{14}: [$T_{cliente}$/ codigo] ≡ [Clientes_rel/ ccodigo]
ψ_{15}: [$T_{cliente}$/ nome] ≡ [Clientes_rel/ cnome]
ψ_{16}: [$T_{cliente}$/ telefone] ≡ [Clientes_rel,{cfone1, cfone2, cfone3}]

Figura 3. Assertivas de Correspondência de Pedidos_v

estrangeira juntamente com a estrutura das tabelas referenciadas por estas ligações. Desse modo, o usuário pode navegar para todas as tabelas relacionadas com a tabela pivô. Para definir uma AC, o usuário relaciona graficamente o atributo do esquema da visão com um atributo ou caminho de uma tabela base como indicado na Figura 4.

O processo de definição das ACs com VBA é *top down* e consiste de dois passos: (i) Primeiro o usuário define as ACs para os atributos do tipo da visão; (ii) em seguida, define, de forma recursiva, as ACs para os atributos dos tipos aninhados no tipo da visão. Assim, na definição das assertivas de Pedidos_v, o usuário primeiro define as correspondências para os atributos de T_{pedido}. Como exemplificado a seguir:

- Para definir a assertiva do atributo codigo, o usuário seleciona codigo no esquema da visão e pcodigo no esquema da base de dados. Ao clicar no botão Salvar, VBA mostra a assertiva gerada ψ_1: [T_{pedido}/ codigo] ≡ [Pedidos_rel/ pcodigo];
- Para definir a assertiva do atributo listaItens, o usuário seleciona listaItens no esquema da visão e fk_2^{-1} no esquema da base de dados.
  A assertiva gerada é ψ_6:[T_{pedido}/ listaItens] ≡ [Pedidos_rel/ fk_2^{-1}];
- Para definir a assertiva do atributo enderecoEntrega, o qual não tem correspondência direta com nenhum atributo de Pedidos_rel, o usuário seleciona o atributo enderecoEntrega e a própria tabela Pedidos_rel, e, então, salva. A assertiva gerada é ψ_4:[T_{pedido}/ enderecoEntrega] ≡ [Pedidos_rel/ NULL].

Em seguida, o usuário deve então definir as ACs para os atributos dos tipos estruturados T_{Item}, $T_{endereco}$ e $T_{cliente}$, os quais são tipos aninhados de T_{pedido}.

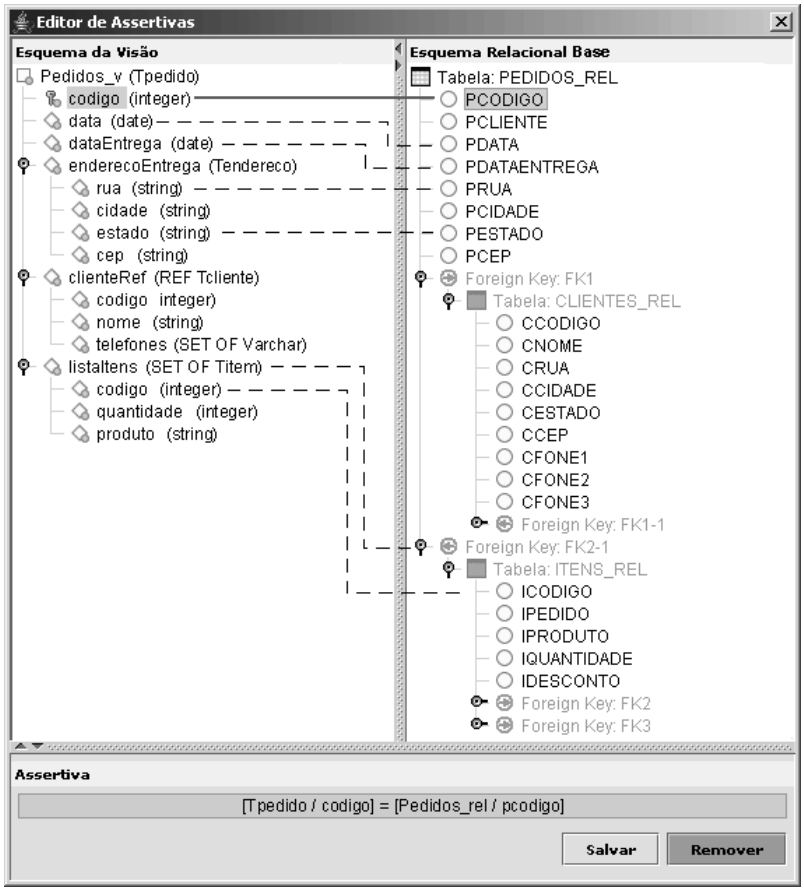


Figura 4. Editor de Assertivas do VBA

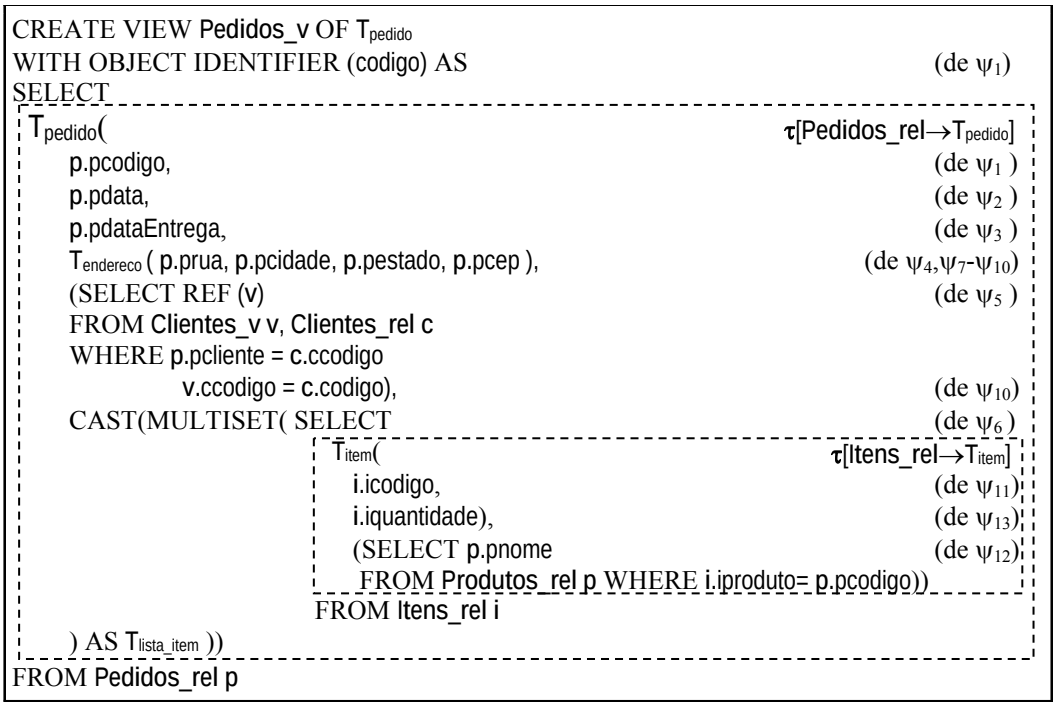


Figura 5. Definição SQL da Visão Pedidos_v

Após definir as assertivas da visão, a definição SQL da visão pode ser gerada automaticamente. A definição da visão de objetos Pedidos_v é mostrada na Figura 5. O construtor τ[Pedidos_rel$\rightarrow T_{pedido}$] da visão Pedidos_v é gerado automaticamente pelo Algoritmo GeraConstrutorSQL com base nas assertivas de correspondência de Pedidos_v. A Figura 5 mostra as assertivas de correspondência que geraram cada subconsulta SQL do construtor. Na próxima seção, discutimos detalhadamente a geração de cada subconsulta de τ[Pedidos_rel$\rightarrow T_{pedido}$] e mostrados que, dada uma tupla r de Pedidos_rel, τ[Pedidos_rel$\rightarrow T_{pedido}$](r) constrói uma instância t de T_{pedido} de forma que t é semanticamente equivalente a r conforme especificado pelas assertivas de Pedidos_v.

5. Algoritmo GeraConstrutorSQL

Figura 6 mostra o Algoritmo GeraConstrutorSQL que recebe como entrada um tipo T, uma tabela R, um pseudônimo r para tabela R, e um conjunto $\mathcal{A}$ de Assertivas de Correspondência que especificam completamente T em termos de R, e gera a função construtora τ[R$\rightarrow$T] tal que dada uma tupla r de R, τ[R$\rightarrow$T](r) constrói uma instância t de T tal que $t \equiv_{\mathcal{A}} r$. No Algoritmo GeraConstrutorSQL, temos que:

(i) φ é um caminho da forma $\ell_1 . \ldots . \ell_{n-1}$ onde:

- ℓ_1 é uma ligação de chave estrangeira dada por $R[a_1^{\ell 1},...,a_{m1}^{\ell 1}] \subseteq R_1[b_1^{\ell 1},...,b_{m1}^{\ell 1}]$ ou inversa da chave estrangeira dada por $R_1[b_1^{\ell 1},..., b_{m1}^{\ell 1}] \subseteq R[a_1^{\ell 1},...,a_{m1}^{\ell 1}]$;
- ℓ_i, é uma ligação de chave estrangeira dada por $R_{i-1}[a_1^{\ell i},..., a_{mi}^{\ell i}] \subseteq R_i[b_1^{\ell i},...,b_{mi}^{\ell i}]$ ou inversa da chave estrangeira dada por $R_i[b_1^{\ell i},.., b_{mi}^{\ell i}] \subseteq R_{i-1}[a_1^{\ell i}, ..., a_{mi}^{\ell i}]$, $2 \leq i \leq n$.

Assim, dado uma tupla r de R, temos que:
$r.\varphi = \{r_n \mid \exists r_2 \in R_2 \wedge ... \wedge \exists r_n \in R_n \wedge r.a_k^{\ell 1} = r_1.b_k^{\ell 1}, 1 \leq k \leq m_1 \wedge r_{i-1}.a_k^{\ell i} = r_i.b_k^{\ell i}, 1 \leq k \leq m_i, 2 \leq i \leq n\}$.

Input: Tipo T, tabela R, pseudônimo r para a tabela R, um conjunto $\mathcal{A}$ de Assertivas de Correspondência que especificam completamente T em termos de R.

Output: Função $\tau[R \rightarrow T]$ tal que dada uma tupla r de R, $\tau[R \rightarrow T](r)$ constrói uma instância t de T e $t \equiv_{\mathcal{A}} r$.

$\tau[R \rightarrow T]$ = "T ("

Para cada atributo c de T faça

Seja T_c o tipo do atributo c;

No caso de:

Caso 1: Se T_c é um tipo atômico e $[T/ c] \equiv [R/ a]$ então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "r.a ,";

Caso 2: Se T_c é um tipo atômico e $[T/ c] \equiv [R/ \varphi/ a]$ então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "(SELECT r_n.a FROM Juncaoφ(r)), ";

Caso 3: Se T_c é um tipo coleção, cujos objetos têm tipo atômico e $[T/ c] \equiv [R/ \{a_1,...,a_n\}]$ então,

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + " T_c ($r.a_1$,...,$r.a_n$) ,";

Caso 4: Se T_c é um tipo coleção, cujos objetos têm tipo atômico e $[T/ c] \equiv [R/ \varphi/ \{a_1,...,a_n\}]$ então,

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + " (SELECT T_c ($r_n.a_1$, ..., $r_n.a_n$) FROM Juncaoφ(r)) ,";

Caso 5: Se T_c é um tipo coleção, cujos objetos têm tipo atômico e $[T/ c] \equiv [R/ \varphi/ a]$, então,

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "(CAST (MULTISET (SELECT r_n .a FROM Juncaoφ(r)) AS T_c) ,";

Caso 6: Se T_c é um tipo referência para a visão V cujos objetos são do tipo T_v, { k_1,...,k_m } é o identificador de V, $[T/ c] \equiv [R/ \varphi]$, e $[T_v/ k_i] \equiv [R_n/ d_i]$, $1 \leq i \leq m$ então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "(SELECT REF(v)

FROM V v, Juncaoφ(r) AND $r_n.d_1 = v.k_1$ AND ... AND $r_n.d_m = v.k_m$) ,";

Caso 7: Se T_c é um tipo coleção, cujos objetos têm tipo referência para a visão V de tipo T_v, onde { k_1,...,k_m } é o identificador de V, $[T/ c] \equiv [R/ \varphi]$ e $[T_v/ k_i] \equiv [R_n/ d_i]$, $1 \leq i \leq m$ então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "CAST (MULTISET (SELECT REF (v)

FROM V v, Juncaoφ(r) AND

$v.k_1 = r_n.d_1$ AND ... AND $v.k_m = r_n.d_m$) AS T_c),";

Caso 8: Se T_c é um tipo estruturado e $[T/ c] \equiv [R/ \varphi]$, então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "(SELECT"+ GeraConstrutorSQL(T_c, R_n, , r_n, $\mathcal{A}$) +

"FROM Juncaoφ(r)) ,";

Caso 9: Se T_c é um tipo coleção, cujos objetos têm tipo estruturado T_{item_c} e $[T/ c] \equiv [R/ \varphi]$ então,

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + "CAST(MULTISET(SELECT" + GeraConstrutorSQL(T_{item_c}, R_n, r_n, $\mathcal{A}$) +

"FROM Juncaoφ(r)) AS T_c) ,";

Caso 10: Se T_c é um tipo estruturado e ψ_c: $[T/ c] \equiv [R/ NULL]$ então

$\tau[R \rightarrow T] = \tau[R \rightarrow T]$ + GeraConstrutorSQL(T_c, R, r, $\mathcal{A}$) + " ,";

fim caso;

fim para;

retorne $\tau[R \rightarrow T]$;

Figura 6. Algoritmo GeraConstrutorSQL

(ii) $\text{Juncao}\varphi(r)$ é definida pelo seguinte fragmento SQL:

$R_1\ r_1,...,\ R_n\ r_n$ WHERE $r.a_1^{\ell_1} = r_1.b_1^{\ell_1}$ AND...AND $r.a_{m_1}^{\ell_1} = r_1.b_{m_1}^{\ell_1}$ AND...
AND $r_{n-1}.a_1^{\ell_n} = r_n.b_1^{\ell_n}$ AND... AND $r_{n-1}.a_{m_n}^{\ell_n} = r_n.b_{m_n}^{\ell_n}$.

De forma que, dado uma tupla r de R, então $r.\varphi$ = SELECT r_n FROM $\text{Juncao}\varphi(r)$.

(iii) Nos casos 7 e 9, o operador MULTISET tem como parâmetro uma subconsulta Q de forma que MULTISET(Q) indica que o resultado da subconsulta Q é uma coleção de objetos (*nested table*); e o operador CAST tem como parâmetro uma subconsulta Q e um tipo T, de forma que CAST((Q) AS T) transforma o resultado da subconsulta Q em um objeto do tipo T

A seguir mostramos que dado um construtor $\tau[R \rightarrow T]$ gerado pelo algoritmo GeraConstrutorSQL, então para qualquer tupla r de R, $\tau[R \rightarrow T](r)$ constrói uma instância t de T tal que $t \equiv_{\mathcal{A}} r$.

No resto desta seção, seja r uma tupla de R e t uma instância de T tal que $t = \tau[R \rightarrow T](r)$. Suponha que o tipo T tem atributos $c_1,...,c_m$. Assim, o construtor $\tau[R \rightarrow T](r)$ tem a seguinte forma: $T(Q_{c_1}(r),..., Q_{c_m}(r))$, onde $Q_{c_i}(r)$ é uma subconsulta SQL que gera o valor do atributo c_i de t para $1 \leq i \leq m$. A prova consiste em mostrar que para qualquer atributo c_i de T temos que o valor de $t.c_i$ definido por $Q_{c_i}(r)$ satisfaz a condição requerida para que $t \equiv_{\mathcal{A}} r$ conforme definido na Seção 3. A seguir, seja c_i um atributo de T tal que T_{c_i} é o tipo de c_i.

(i) Suponha que T_{c_i} é um tipo atômico e $[T/c_i] \equiv [R/a]$. Pelo Caso 1 do algoritmo, temos que $Q_{c_i}(r) = r.a$. Então, $t.c_i = r.a$.

(ii) Suponha que T_{c_i} é um tipo atômico e $[T/c_i] \equiv [R/\varphi/a]$. Pelo Caso 2 do algoritmo, temos que $Q_{c_i}(r)$ = (SELECT $r_n.a$ FROM $\text{Juncao}\varphi(r)$). Pela definição de $\text{Juncao}\varphi(r)$ temos que $r_n \in r.\varphi$. Como c_i é monovalorado, então: $t.c_i = r_n.a$, onde $r_n \in r.\varphi$.

(iii) Suponha que T_{c_i} é um tipo coleção cujos objetos têm tipo atômico e $[T/c_i] \equiv [R/\{a_1,...,a_n\}]$. Pelo Caso 3 do algoritmo temos que $Q_{c_i}(r) = T_{c_i}\ (r.a_1,...,\ r.a_n)$. Então, $t.c_i = T_{c_i}\ (r.a_1,...,\ r.a_n)$.

(iv) Suponha que T_{c_i} é um tipo coleção cujos objetos têm tipo atômico e $[T/c_i] \equiv [R/\varphi/\{a_1,...,a_n\}]$. Pelo Caso 4 do algoritmo temos que:

$Q_{c_i}(r)$ = (SELECT $T_{c_i}\ (r_n.a_1,...\ ,\ r_n.a_n)$ FROM $\text{Juncao}\varphi(r)$)

De $\text{Juncao}\varphi(r)$ e dado que φ é monovalorado, então temos:

$t.c_i = \{\ r_n.a_1,\ ...\ ,\ r_n.a_n\ \}$ onde $r_n \in r.\varphi$

(v) Suponha que T_{c_i} é um tipo coleção cujos objetos têm tipo atômico e $[T/c_i] \equiv [R/\varphi/a]$. Pelo Caso 5 do algoritmo, temos que:

$Q_{c_i}(r)$ = CAST (MULTISET (SELECT $r_n.a$ FROM $\text{Juncao}\varphi(r)$) AS T_{c_i}).

De $\text{Juncao}\varphi(r)$ e dado que c_i é multivalorado, temos: $t.c_i = \{\ o \mid o = r_n.a$ onde $r_n \in r.\varphi\}$.

(vi) Suponha que T_{c_i} é um tipo referência para a visão V do tipo T_v, $[T/c_i] \equiv [R/\varphi]$, $\{k_1,...,k_m\}$ é o identificador de V, e $[T_v/k_i] \equiv [R_n/d_i]$, $1 \leq i \leq m$. Pelo Caso 6 do algoritmo, temos:

$Q_{c_i}(r)$ = SELECT REF(v) FROM V v, Juncaoφ(r) AND $r_n.d_1$=v.k_1 AND…AND $r_n.d_m$=v.k_m

De Juncaoφ(r), e dado que c_i é monovalorado, temos:

$t.c_i$ = REF(v), onde v ∈ V, $r_n \in r.\varphi$ e $r_n.d_i$ = v.k_i, $1 \leq i \leq m$.

Como R_n é a tabela pivô de V então para qualquer v (V existe r' (Rn e v (A r'. Como {k1,...,km} é o identificador de V, e [Tv/ki] ([Rn/di], $1 \leq i \leq m$, e dado que existe um mapeamento 1-1 de Rn em V, então {d1,...,dm} é chave primária de Rn. Assim, como rn.di = v.ki, $1 \leq i \leq m$, então v (A rn. Logo, temos que

$t.c_i$ = REF(v) onde v (V, rn (r.(e $v \equiv_{\mathcal{A}} r_n$.

(vii) Suponha que T_{c_i} é um tipo coleção cujos objetos têm tipo referência para a visão V do tipo T_v, cujos objeto são do tipo T_v, $\{k_1,...,k_m\}$ é o identificador de V, $[T/c_i] \equiv [R/\varphi]$ e $[T_v/k_i] \equiv [R_n/d_i]$, $1 \leq i \leq m$. Pelo caso 7 do algoritmo, temos que:

$Q_{c_i}(r)$ = CAST (MULTISET (SELECT REF (v) FROM V v, Juncaoφ(r)
AND v.k_1 = $r_n.d_1$ AND … AND v.k_m= $r_n.d_m$) AS T_{c_i}).

De forma similar ao caso anterior, por Juncaoφ(r) e dado que c_i é multivalorado, então temos que $t.c_i$ = { REF(v) | v ∈ V, $v \equiv_{\mathcal{A}} r_n$ onde $r_n \in r.\varphi$}.

(viii) Suponha que T_{c_i} é um tipo estruturado e $[T/c_i] \equiv [R/\varphi]$. Pelo caso 8 do algoritmo, temos que $Q_{c_i}(r)$ = SELECT $\tau[R_n \rightarrow T_{c_i}](r_n)$ FROM Juncaoφ(r).

De Juncaoφ(r), e dado que c_i é monovalorado, então $t.c_i = \tau[R_n \rightarrow T_{c_i}](r_n)$, onde $r_n \in r.\varphi$. Neste caso assumimos que, dado $\tau[R_n \rightarrow T_{c_i}]$ o construtor retornado por GeraConstrutorSQL(T_{c_i}, R_n, r_n, $\mathcal{A}$), temos que para qualquer tupla r_n de R_n, $\tau[R_n \rightarrow T_{c_i}](r_n)$ constrói uma instância t_{c_i} de T_{c_i} tal que $t_{c_i} \equiv_{\mathcal{A}} r_n$. Essa suposição pode ser provada por indução na profundidade de aninhamento de T_{c_i} em T.

Logo, temos que $t.c_i = t_{c_i}$ onde $t_{c_i} \equiv_{\mathcal{A}} r_n$ e $r_n \in r.\varphi$.

(ix) Suponha que T_{c_i} é um tipo coleção cujos objetos têm tipo estruturado T_{item_c} e $[T/c_i] \equiv [R/\varphi]$. Pelo Caso 9 do algoritmo, temos que:

$Q_{c_i}(r)$ = CAST(MULTISET(SELECT $\tau[R_n \rightarrow T_{item\_c}](r_n)$ FROM Juncaoφ(r)) AS T_{c_i})

De Juncaoφ(r), e dado que c_i é multivalorado, temos:

$t.c_i = \{ t_{item\_c} \mid t_{item\_c} = \tau[R_n \rightarrow T_{item\_c}](r_n)$ onde $r_n \in r.\varphi\}$.

Neste caso, assumimos que dado $\tau[R_n \rightarrow T_{item\_c}](r_n)$, o construtor retornado por GeraConstrutorSQL(T_{item_c}, R_n, r_n, $\mathcal{A}$), temos que para qualquer tupla r_n de R_n, $\tau[R_n \rightarrow T_{item\_c}](r_n)$ constrói uma instância t_{item_c} de T_{item_c} tal que $t_{item\_c} \equiv_{\mathcal{A}} r_n$.

Logo, temos que $t.c_i = \{ t_{item\_c} \mid t_{item\_c} \equiv_{\mathcal{A}} r_n$ onde $r_n \in r.\varphi\}$.

(x) Suponha que T_{c_i} é um tipo estruturado e $[T/c_i] \equiv [R/NULL]$. Pelo Caso 10 do algoritmo, temos que $Q_{c_i}(r) = \tau[R \rightarrow T_{c_i}](r)$. Neste caso, assumimos que o construtor $\tau[R \rightarrow T_{c_i}](r)$, retornado por GeraConstrutorSQL(T_{c_i}, R, r, $\mathcal{A}$) constrói, para qualquer tupla r de R, uma instância t_{c_i} de T_{c_i} tal que $t_{c_i} \equiv_A r$.

Logo, temos $t.c_i = t_{c_i}$ onde $t_{c_i} \equiv_{\mathcal{A}} r$.

Portanto, de (i) – (x) temos que $t \equiv_{\mathcal{A}} r$. Mostramos assim, que para qualquer tupla r de R, $\tau[R \rightarrow T](r)$ constrói uma instância t de T tal que t é semanticamente equivalente a r conforme especificado pelas assertivas em $\mathcal{A}$.

A seguir, ilustramos o uso do algoritmo para a geração do construtor τ[Pedidos_rel$\rightarrow T_{pedido}$] usado na definição SQL da visão Pedidos_v mostrada na Figura 5. Para gerar o construtor τ[Pedidos_rel$\rightarrow T_{pedido}$], o algoritmo GeraConstrutorSQL é chamado com os seguintes argumentos: o tipo T_{pedido}, a relação Pedidos_rel, o pseudônimo p para tabela Pedidos_rel, e o conjunto de assertivas de correspondência de Pedidos_v (vide Figura 3). O construtor gerado tem a forma:

$T_{pedido}(Q_{codigo}(r), Q_{data}(r), Q_{enderecoEntrega}(r), Q_{cliente\_ref}(r), Q_{listaItens}(r))$,

o qual constrói uma instância t de T_{pedido} a partir de uma tupla r de Pedidos_rel, tal que $t \equiv_{\mathcal{A}} r$. As subconsultas $Q_{codigo}(r)$, $Q_{data}(r)$, $Q_{enderecoEntrega}(r)$, $Q_{cliente\_ref}(r)$, $Q_{listaItens}(r)$ geram, a partir da tupla r, os valores dos atributos codigo, data, enderecoEntrega, cliente_ref, e listaItens de t, respectivamente. Como mostraremos a seguir, cada subconsulta é definida conforme especificado pela assertiva de correspondência do atributo correspondente:

- O atributo codigo tem tipo atômico e assertiva ψ_1:[T_{pedido}/ codigo] $\equiv$ [Pedidos_rel/ pcodigo]. Do caso 1 do algoritmo, temos que:

  $Q_{codigo}(p) = p.pcodigo$.

- O atributo data tem tipo atômico e assertiva ψ_2:[T_{pedido}/ data] $\equiv$ [Pedidos_rel/ pdata]. Do caso 1 do algoritmo, temos que:

  $Q_{data}(p) = p.pdata$.

- O atributo enderecoEntrega tem tipo estruturado $T_{endereco}$ e assertiva ψ_4:[T_{pedido}/ enderecoEntrega] $\equiv$ [Pedidos_Rel/ NULL]. Do caso 10 do algoritmo, temos que $Q_{enderecoEntrega}(p)=\tau$[Pedidos_rel$\rightarrow T_{endereco}$], onde o construtor τ[Pedidos_rel$\rightarrow T_{endereco}$] é gerado pela chamada recursiva do algoritmo com os seguintes argumentos: o tipo $T_{endereco}$, a tabela Pedidos_rel, o pseudônimo p para a tabela Pedidos_rel, e o conjunto de assertivas de correspondência de Pedidos_v. O construtor gerado tem a forma:

  $T_{endereco}(Q_{rua}(p), Q_{cidade}(p), Q_{estado}(p), Q_{cep}(p))$.

  Das assertivas ψ_7 - ψ_{10} e do caso 1 do algoritmo temos:

  $Q_{rua}(p) = p.prua$; $Q_{cidade}(p) = p.pcidade$; $Q_{estado}(p) = p.pestado$; e $Q_{cep}(p) = p.pcep$.

  Assim temos que:

  $Q_{enderecoEntrega}(p) = T_{endereco}$ (p.prua, p.pcidade, p.pestado, p.pcep).

- O atributo cliente_ref tem tipo referência e assertiva ψ_5: [T_{pedido}/ cliente_ref] $\equiv$ [Pedidos_rel/ fk_1] onde Clientes_v é a visão referenciada por cliente_ref, {codigo} é o identificador de Clientes_v, e [$T_{cliente}$/ codigo] $\equiv$ [Pedidos_rel/ ccodigo]. Do caso 6 do algoritmo, temos:

  $Q_{cliente\_ref}(p)$ = (SELECT REF(v) FROM Clientes_v v, Juncaoφ(p)
  AND v.codigo = p.ccodigo).

  Dado que $\varphi=fk_1$ onde fk_1 é a chave estrangeira Pedidos_rel[pcliente] $\subseteq$ Clientes_rel[ccodigo], temos:

  Juncaoφ(p) = Clientes_rel c WHERE p.pcliente= c.ccodigo.

  Então, temos que:

  $Q_{cliente\_ref}(p)$ = (SELECT REF(v) FROM Clientes_v v, Clientes_rel c
  WHERE p.pcliente = c.ccodigo AND v.codigo = c.ccodigo)

- O atributo listaItens é do tipo coleção $T_{listaItem}$ cujos objetos têm tipo estruturado T_{item} e assertiva ψ_6:[T_{pedido}/ listaItens] ≡ [Pedidos_rel/ fk_2^{-1}]. Assim, do caso 9 do algoritmo temos que:

  $Q_{listaItens}(p)$=CAST(MULTISET(SELECT τ[Itens_rel→T_{item}](p) FROM Juncaoφ(p)) AS $T_{listaItem}$).

  Dado que $\varphi=fk_2^{-1}$, onde fk_2 é a chave estrangeira Itens_rel[ipedido] ⊆ Pedidos_rel[pcodigo], temos:

  Juncaoφ(p)= Itens_rel i WHERE p.pcodigo = i.ipedido.

  O construtor τ[Itens_rel→T_{item}] é gerado pela chamada recursiva do algoritmo com os seguintes argumentos: o tipo T_{item}, a relação Itens_rel, o pseudônimo i para a tabela Itens_rel, e o conjunto de assertivas de correspondência de Pedidos_v. O construtor gerado tem a forma:

  T_{item} (Q_{codigo}(i), $Q_{produto}$(i), $Q_{quantidade}$(i)).

- Das assertivas ψ_{11} e ψ_{12} e do caso 1 do algoritmo, temos que:

  Q_{codigo}(i)= i.icodigo;

  $Q_{quantidade}$(i)= i.iquantidade.

- O atributo produto tem tipo atômico definido pela assertiva ψ_{13}:[T_{item}/ produto] ≡ [Titens_rel/ fk_3/ pnome]. Assim, do caso 2 do algoritmo temos que:

  $Q_{produto}$(i)= (SELECT p.pnome FROM Juncaoφ(i)).

  Dado que $\varphi=fk_3$, onde fk_3 é a chave estrangeira, Itens_rel[iproduto] ⊆ Produtos_rel[pcodigo], temos:

  Juncaoφ(i) = Produtos_rel p WHERE i.iproduto= p.pcodigo

  Assim temos

  $Q_{produto}$(i)=(SELECT p.pnome FROM Produtos_rel p WHERE i.iproduto= p.pcodigo).

Logo, temos que:

$Q_{listaItens}$(p) = CAST(MULTISET(SELECT T_{item} (i.icodigo, i.iquantidade,
(SELECT p.pnome FROM Produtos_rel p
WHERE i.iproduto= p.pcodigo))
FROM Itens_rel i WHERE p.pcodigo = i.ipedido
) AS $T_{listaItem}$).

6. Conclusões

Neste trabalho propomos o uso de Assertivas de Correspondência para especificar o mapeamento entre o esquema de uma visão de objetos e um esquema relacional. Nós mostramos que as ACs da visão especificam completamente a visão em termos do esquema relacional base, no sentido que define um mapeamento funcional de instâncias do esquema base em instâncias da visão.

O algoritmo GeraConstrutorSQL apresentado, gera, baseado nas ACs da visão, o construtor SQL que constrói os objetos da visão a partir de tuplas da base de dados relacional. Como mostramos, o formalismo proposto permite provar formalmente, que um construtor gerado pelo algoritmo realiza corretamente o mapeamento especificado pelas ACs da visão. É importante salientar que o algoritmo pode ser facilmente

adaptado para outros SGBDs objeto-relacional. Como trabalhos futuros pretendemos estender o formalismo de mapeamento proposto para tratar outros tipos de visão de objeto.

Referências

1. Atzeni, P., Mecca, G., Merialdo, P., *Design and Maintenance of Data-Intensive Web Sites*, In: Proceedings of the 6th International Conference on Extending Database Technology, London, UK, 1998, p. 436-450.
2. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M., *Designing Data-Intensive Web Applications*, Morgan-Kaufmann, 2002.
3. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L., *Data Exchange: Semantics and Query Answering*, In: Lecture Notes in Computer Science, vol. 2572, 2003, p. 207-224
4. Fahl, G., Risch, T., *Query processing over object views of relational data*, The VLDB Journal, vol. 6, issue 4, Nov 1997, p. 261-281.
5. Hernandez, M.A., Miller, R.J., Haas, L., Yan, L., Ho, C.T.H., Tian, X., *Clio: A semi-automatic tool for schema mapping*. In: Proceedings of the 2001 ACM SIGMOD, p. 607, California, USA, 2001.
6. Madhavan, J., Bernstein, P., Rahm, E., *Generic Schema Matching with Cupid*, In: Proceedings of the 27th International Conference on Very Large Databases, 2001, p. 49-58.
7. *Oracle Corporation*. Disponível em: http://technet.oracle.com. (Acessado em 24 de abril 2005)
8. Popa, L., Velegrakis, Y., Miller, R. J., Hernandez, M. A., Fagin, R., *Translating Web Data*. In: Proceedings of the International Conference on Very Large Data Bases, 2002, p. 598–609.
9. Rahn, E., Bernstein, P. A., *A survey of approaches to automatic schema matching*, The VLDB Journal, vol.10, num. 4, Dec. 2001, p. 334-350.
10. Santos, L. A. L., *XML Publisher: Um framework para Publicar Dados Objeto Relacional como XML*, Dissertação de Mestrado, Universidade Federal do Ceará, 2004.
11. Stonebraker, M., Moore, D., *Object Relational DBMSs: The Next Great Wave*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1995.
12. Takahashi, T., Keller, A. M., *Implementation of Object View Query on a Relational Database*, In: Proceeding of International Conference on Data and Knowledge Systems for Manufacturing and Engineering, 1994, p. 192-197.
13. Vidal, V. M. P., Lóscio, B.F., *Solving the Problem of Semantic Heterogeneity in Defining Mediator Update Translators*, In: Proceedings of 18th International Conference on Conceptual Modeling, Paris, France, 1999, p. 293-308.
14. Yu, C., Popa, L., *Constraint-Based XML Query Rewriting For Data Integration*. In: International Conference on Management of Data, Paris, France, 2004, p. 371–382,