

TÍTULO: PROJETO E MANUTENÇÃO DE REPRESENTAÇÕES
RELACIONAIS OTIMIZADAS PARA ESQUEMAS
ENTIDADE-RELACIONAMENTO

AUTORES: Marco Antonio Casanova e Luiz Tucheran

ENDEREÇO PARA CONTATO: Centro Científico Rio - IBM Brasil
Caixa Postal 4624
20.001, Rio de Janeiro, R.J.
Tel. (021) 271-3784

CURRICULUM VITAE:

MARCO ANTONIO CASANOVA é pesquisador do Centro Científico-Rio da IBM Brasil. Formou-se em Engenharia Eletrônica pelo IME em 1974, obteve o grau de Mestre em Ciências da Computação pela PUC-RJ em 1976, e os graus de Mestre em Ciências (1977) e Doutor em Filosofia (1979), ambos em Matemática Aplicada, pela Universidade de Harvard. Suas áreas de interesse são Projeto de Banco de Dados e Programação em Lógica.

LUIZ TUCHERMAN é pesquisador do Centro Científico-Rio da IBM Brasil. Formou-se em Engenharia Civil-Eletrotécnica pela Universidade Federal de Juiz de Fora, obteve o grau de Doutor em Ciência da Computação pela PUC-RJ em 1989 e o grau de Mestre em Ciências da Computação em 1983 pela mesma universidade. Suas áreas de interesse são Projeto de Banco de Dados e Sistemas de Gerência de Banco de Dados.

TOTAL DE PÁGINAS: 16

RESUMO:

Este trabalho propõe inicialmente um algoritmo para obter representações otimizadas no modelo relacional de esquemas conceituais de banco de dados descritos em um modelo entidade-relacionamento estendido. O algoritmo associa ainda a cada estrutura relacional uma explicação indicando que conceitos ela representa. Em seguida, descreve um algoritmo de reprojeto que, dadas mudanças sobre um esquema conceitual, gera um plano para modificar a representação relacional original e reorganizar o estado do banco de dados para refletir o novo esquema conceitual.

PALAVRAS-CHAVE: modelo entidade-relacionamento, projeto lógico de banco de dados, otimização de esquemas.

1. INTRODUÇÃO

O projeto de bancos de dados relacionais tipicamente parte de uma primeira descrição utilizando uma variante do modelo entidade-relacionamento. Chamaremos esta descrição simplesmente de esquema conceitual. Esta fase do projeto é relativamente simples, embora as otimizações empregadas continuem sendo aplicadas de forma manual e empírica. A primeira contribuição deste trabalho consiste na descrição de um algoritmo de projeto que aceita como entrada um esquema conceitual e gera como saída uma representação relacional otimizada, acrescida de explicações indicando quais objetos do esquema conceitual cada tabela representa e porque. O algoritmo captura heurísticas de otimização comumente adotadas, que podem ser explicadas exclusivamente em termos do esquema conceitual dado como entrada.

Este trabalho considera também o problema de manutenção de bancos de dados projetados da forma descrita acima. Neste caso, a dificuldade reside em que qualquer modificação no esquema conceitual deve ser traduzida em modificações na representação relacional e na reestruturação das tabelas que compõem o estado atual do banco de dados relacional. Além disto, uma mudança no esquema conceitual pode invalidar otimizações efetuadas ou criar oportunidade para novas otimizações. A contribuição deste trabalho à solução deste problema consiste na descrição de um algoritmo de reprojeto que aceita como entrada um esquema conceitual, uma representação relacional produzida pelo algoritmo de projeto e uma seqüência de modificações e produz como saída o novo esquema conceitual e um plano para criar a nova representação relacional e reestruturar o banco de dados.

O problema de mapeamento automático otimizado de esquemas conceituais entidade-relacionamento em representações relacionais, embora importante, tem sido pouco estudado de forma rigorosa [6,8,9]. Já o uso do modelo entidade-relacionamento e suas extensões para projeto conceitual possui vasta literatura [3,4,5,6,7,10,11]. Este trabalho é parte de uma investigação mais ampla sobre ferramentas para projeto automatizado de banco de dados, descrita em parte em [1,2].

Este trabalho está dividido da seguinte maneira. A seção 2 descreve a extensão do modelo entidade-relacionamento adotada e alguns conceitos do modelo relacional. A seção 3 aborda o algoritmo de projeto, enquanto a seção 4, o algoritmo de reprojeto. Finalmente a seção 5 contém as conclusões.

2. PRELIMINARES

2.1 O Modelo Entidade-Relacionamento Estendido

Esta seção resume a extensão do modelo entidade-relacionamento (ou modelo-ERE) adotada. Uma descrição completa encontra-se em [1].

Um *esquema conceitual ERE* ou, simplesmente, um *esquema ERE*, contém descrições de conjuntos de entidades e conjuntos de relacionamentos, possivelmente organizando os conjuntos de entidades em um grafo de especializações.

A definição de um conjunto de entidades é bastante padronizada. Permitimos a especificação de um conjunto de *chaves alternativas*, em adição a uma *chave primária*. A chave primária, as chaves alternativas e mesmo a lista de atributos são em princípio opcionais. Mas, assumindo que desejamos selecionar cada entidade por um dos seus valores de chave, esta liberalidade somente faz sentido quando um esquema de entidade especializa um outro esquema de entidade do qual ele herda suas chaves e a lista de atributos (veja o conceito de especialização abaixo).

Portanto, introduzimos uma *declaração de esquema de entidade* como uma expressão da seguinte forma (as expressões entre chaves podem ser omitidas):

```
defina entidade E
    [atributos  $A_1 D_1, \dots, A_n D_n$ ]
    [chave  $K_0$ ] ... [chave  $K_p$ ]
```

Dizemos que *E* é o nome, A_1, \dots, A_n é a lista de nomes de atributos, K_0 é a chave primária e K_1, \dots, K_p são as chaves alternativas do esquema. Dizemos também que D_i é o tipo do domínio de A_i , para $i = 1, \dots, n$, e que A_i aceita valores nulos se D_i também aceitar.

Somente permitimos definir conjuntos de relacionamentos sobre conjuntos de entidades, mas deixamos que um conjunto de entidades participe mais de uma vez de um conjunto de relacionamentos, desde que um papel distinto seja designado para cada ocorrência do conjunto de entidade participante. Permitimos também que um conjunto de relacionamento seja opcionalmente definido como *total* [11] em relação ao papel de um conjunto de entidade. Introduzimos também a noção de identificador para os conjuntos de relacionamentos como um correspondente às chaves. Escolhemos o termo 'identificador' para chamar a atenção para o fato de que agora podemos ter uma lista de participantes no relacionamento, e não uma lista de atributos. Por último, permitimos

que um relacionamento seja definido com mais de um identificador, uma liberalidade que é necessária, por exemplo, para a definição de relacionamentos binários 1-1.

Os identificadores capturam a anotação usual que rotula com "1" ou "n" os arcos que deixam o losango que representa um conjunto de relacionamento em um diagrama ERE, mas eles são mais gerais do que esta convenção de anotação. O identificador primário é sempre o primeiro a ser especificado.

Se o identificador for constituído por um único papel, então o relacionamento é dito *funcional* em relação a este identificador.

Uma *declaração de um esquema de relacionamento* é uma expressão da forma:

defina relacionamento R
sobre O_1 [como N_1] [total], ..., O_m [como N_m] [total]
[atributos $A_1 D_1, \dots, A_n D_n$]
[identificador I_0] ... [identificador I_p]

Dizemos que R é o *nome*, A_1, \dots, A_n é a lista de *nomes de atributos*, I_0 é o *identificador primário* e I_1, \dots, I_p são os *identificadores alternativos* do esquema. Dizemos também que o esquema de relacionamento R tem m *papéis*. O esquema no $i^{\text{ésimo}}$ papel é O_i e o nome do $i^{\text{ésimo}}$ papel é N_i , se especificado, ou o próprio O_i , caso contrário. Quando "total" é especificado para " O_i [como N_i]", dizemos que R é *total* no $i^{\text{ésimo}}$ papel do esquema.

Permitimos também o projetista do banco de dados organizar os esquemas de entidades de um dado esquema conceitual como um grafo de especialização, que deve ser sempre acíclico. Note que o grafo de especialização não necessita então ser uma árvore, isto é, uma hierarquia estrita. Se um esquema de entidade E for definido como uma *especialização* de F , então E deve sempre denotar um subconjunto do conjunto de entidades associada com F . Por fim, um esquema de entidades F herda os atributos e as chaves de todos os esquemas que especializa, direta ou transitivamente.

Mais precisamente, introduzimos uma *declaração de especialização* como uma expressão da forma:

especialize O em O_1, \dots, O_m

Dizemos que O é o esquema *especializado* e que O_i é uma *especialização* de O . Também dizemos que O é uma *generalização* de O_1, \dots, O_m .

O *grafo* de um esquema conceitual E , $g(E) = (V, A)$, é um grafo dirigido tal que V é o conjunto de nomes de esquemas de entidades e de relacionamentos em E e um par (O, P) está em A se e somente se O é uma especialização de P em E ou O é um relacionamento sobre P em E .

2.2 Conceitos do Modelo Relacional

O esquema relacional resultado do mapeamento do esquema conceitual ERE está em conformidade com o modelo relacional tradicional, isto é, é definido através de esquemas relacionais em primeira forma normal, chaves e uma classe de dependência de inclusão que é suficientemente poderosa para capturar especializações e para auxiliar na definição de relacionamentos.

Mais precisamente um *esquema de relação* é uma expressão da forma

defina tabela T
[atributos $A_1 D_1, \dots, A_n D_n$]
[chave K_0] ... [chave K_p]

onde T é o *nome* e A_1, \dots, A_n é a lista de *nomes de atributos* do esquema, K_0 é a *chave primária* e K_1, \dots, K_p são as *chaves alternativas* do esquema. Dizemos também que D_i é o tipo do domínio de A_i , para $i = 1, \dots, n$, e que A_i *aceita valores nulos* se D_i também aceitar.

Frequentemente usaremos o termo *tabela* como sinônimo de esquema de relação.

Seja R um conjunto de esquemas de relação. Uma *dependência de inclusão* é uma expressão da forma $T_1[X_1] \subseteq T_2[X_2]$ onde, para $i = 1, 2$, T_i é o nome de um esquema de relação em R e X_i é uma sequência de atributos distintos de T_i tal que X_1 e X_2 tem o mesmo comprimento e X_2 é definido como a chave de T_2 .

Uma *definição de visão* sobre R é uma tripla (V, Q, E) onde V é um esquema de relação cujo nome é distinto dos nomes de esquemas em R , Q é uma expressão relacional n -ária sobre R , E é a especificação das traduções corretas para as operações sobre V em operações sobre os esquemas em R . Dizemos que V é o *nome* da visão e que Q é a *expressão da definição* da visão.

Finalmente, um *esquema relacional* é uma tripla $S = (R, V, I)$ onde R é um conjunto de esquemas de relação com nomes distintos, V é um conjunto de definições de visões sobre R com nomes distintos e I é um conjunto de dependências de inclusão sobre R .

3. PROJETO

3.1 Colapsamento de Esquemas

O algoritmo de projeto sistematiza uma heurística bastante difundida para criação de representações relacionais de esquemas conceituais (no modelo ERE), qual seja, representar em uma única tabela T:

- um esquema de entidade E e um esquema de relacionamento R tal que R é funcional com relação a E ;
- um esquema de entidade E e uma sua especialização F .

Por exemplo, suponha que *TRABALHA* seja um esquema de relacionamento sobre *EMPREGADO* e *DEPARTAMENTO*, funcional sobre *EMPREGADO*. Então, podemos representar *TRABALHA* e *EMPREGADO* por uma única tabela *EMPREGADO** contendo os atributos de ambos e da chave de *DEPARTAMENTO* e tendo como chave primária aquela de *EMPREGADO*. A representação relacional de *EMPREGADO* será então através de uma visão, que podemos chamar também de *EMPREGADO*, definida como a projeção de *EMPREGADO** sobre os atributos do esquema de entidade *EMPREGADO*. Similarmente, a representação relacional de *TRABALHA* será através da visão *TRABALHA*, definida como a projeção de *EMPREGADO** sobre os atributos do esquema de entidade *TRABALHA* e das chaves primárias de *EMPREGADO* e *DEPARTAMENTO*, restrita às tuplas em que a chave de *DEPARTAMENTO* não é nula.

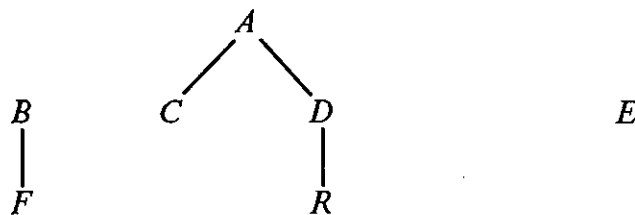
Como um segundo exemplo, suponha que *SECRETARIA* seja uma especialização de *EMPREGADO*. Então, podemos representar *EMPREGADO* e *SECRETARIA* em uma única tabela *EMPREGADO** contendo os atributos de ambos e tendo como chave primária aquela de *EMPREGADO*. Neste caso, devemos incluir um atributo especial em *EMPREGADO**, ou utilizar um atributo que não admite valores nulos em *SECRETARIA*, para indicar quando uma tupla de *EMPREGADO** representa um empregado que também é uma secretária. Digamos, por exemplo, que *SECRETARIA* tenha *Velocidade* como atributo que não admite valores nulos. Então, a representação relacional de *SECRETARIA* será através de uma visão, também chamada de *SECRETARIA*, definida como a projeção de *EMPREGADO** sobre os atributos do esquema de entidade *SECRETARIA* e da chave primária de *EMPREGADO* (para identificar uma secretária como um empregado), restritas às tuplas com *Velocidade* diferente de nulo (que identificam os empregados que são secretárias). A representação do esquema de entidades *EMPREGADO* é semelhante à do exemplo anterior. De fato, os dois exemplos

podem ser combinados, representando em uma única tabela EMPREGADO* os três esquemas, EMPREGADO, SECRETÁRIA e TRABALHA.

Para sistematizar o processo de otimização ilustrado pelos dois exemplos acima, introduziremos inicialmente o conceito de arco colapsável. Seja E um esquema conceitual no modelo ERE e seja $g(E) = (V, A)$ o grafo de E . Um arco (F, G) em A é *colapsável* se e somente se F for um esquema de relacionamento funcional em G ou F for uma especialização de G . Um nó F é colapsável se e somente se existir um arco colapsável em $g(E)$ saindo de F . Se (F, G) for um arco colapsável é possível então representar F e G em uma única tabela, como ilustrado pelos dois exemplos anteriores.

Uma floresta f é uma *floresta de colapsamento* para E se e somente se os nós de f são os de $g(E)$ e F é um filho de G se e somente se o arco (F, G) for colapsável. A floresta é *completa* se e somente se nenhuma de suas raízes for colapsável. Assim, uma floresta de colapsamento completa para E indica uma forma de representar todos os esquemas de E através de tabelas até que não seja possível colapsar mais nenhum esquema.

Adotaremos no que se segue uma representação por multilistas para as florestas. Por exemplo, a multilista $f = (A(C, D(R)), B(F), E)$ representa a floresta



3.2 Algoritmo de Projeto

Neste ponto estamos preparados para esboçar o algoritmo de projeto:

Entrada: um esquema conceitual E

Saída: uma representação relacional R para E

Passo 1:

- construa uma floresta de colapsamento completa f para E .

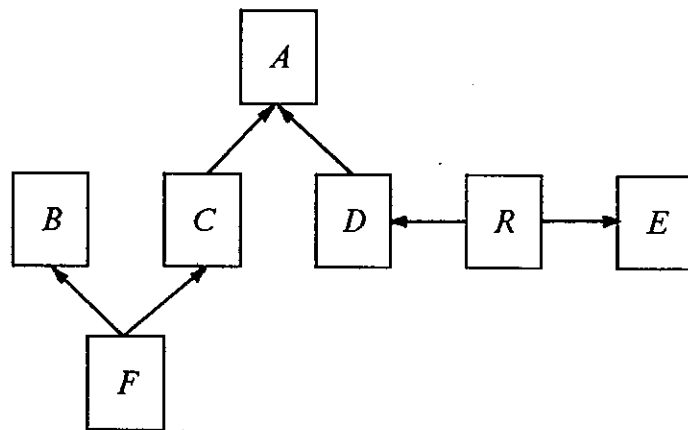
Passo 2:

- crie, a partir de f , uma representação relacional R para E de acordo com as seguintes linhas gerais. Para cada árvore v de f com raiz F :

- gere uma tabela básica F^* contendo os atributos de todos os esquemas correspondendo a nós da árvore ν e tendo como chaves aquelas de F ;
- para cada nó interior G (ou seja, que não é raiz) de ν , gere uma visão G sobre F^* contendo todos os atributos de G e da chave primária de F .

Note que o passo 1 é não-determinístico pois haverá mais de uma floresta de colapsamento completa para E se existir um esquema F de E tal que F possui mais de um arco saindo que seja colapsável. Isto ocorrerá quando F for especialização de mais de um esquema ou quando F for um esquema de relacionamento funcional em mais de um esquema de entidade.

Ilustraremos o processamento do algoritmo através de um exemplo detalhado. Seja E um esquema conceitual no modelo ERE abstraído pelo seguinte grafo:



O passo 1 do algoritmo constrói uma árvore de colapsamento completa para E da seguinte forma. Inicialmente a floresta conterá todos os esquemas de E como raízes. Em seguida, o algoritmo escolhe cada um dos nós (raiz ou interno) e nele colapsa todos os nós possíveis:

Floresta	Nó Escolhido	Nós Colapsados
$f_0 = (A, B, C, D, E, F, R)$	-	-
$f_1 = (A(C, D), B, E, F, R)$	A	C, D
$f_2 = (A(C, D), B(F), E)$	B	F
-	C	nenhum
$f_3 = (A(C, D(R)), B(F), E)$	D	R
-	E	nenhum
-	F	nenhum
-	R	nenhum

O passo 2 do algoritmo gera então uma representação relacional para E a partir de f_3 , descrita esquematicamente nas tabelas abaixo, onde K_U representa o conjunto dos atributos da chave primária e A_U o conjunto de atributos de uma esquema de entidade U (incluindo os das chaves). Note que associamos uma explicação a cada tabela T . Se T representa mais de um esquema, a explicação é a árvore t da floresta de colapsamento que corresponde a T pois t indica todos os colapsamentos que geraram T . Caso contrário, a explicação reduz-se ao esquema que T representa.

Tabelas Básicas			
Nome	Chaves	Atributos	Explicação
A*	chaves de A	A_A, A_C, A_D, A_R, K_E	$A(C, D(R))$
B*	chaves de B	A_B, A_F, K_C, K_A	$B(F)$
E	chaves de E	A_E	E

Visões			
Nome	Base	Atributos	Explicação
A	A*	A_A	A
B	B*	A_B	B
C	A*	A_C, K_A	C
D	A*	A_D, K_A	D
F	B*	A_F, K_B, K_C, K_A	F
R	A*	A_R, K_D, K_E	R

Além destas tabelas, a representação relacional de E contém as seguintes dependências de inclusão:

$$R[K_E] \subseteq E[K_E]$$

$$F[K_A] \subseteq C[K_A]$$

Completaremos esta seção com algumas observações sobre a implementação do algoritmo de projeto. Poderíamos configurar o passo 1 do algoritmo para enumerar todas as possíveis florestas de colapsamento completas do esquema conceitual E e escolher a floresta que levasse à representação ótima. Porém, dependendo do esquema conceitual,

isto poderia levar a um número muito grande de florestas, o que não tornaria o algoritmo muito prático. Além disto haveria o (sério) problema de definir quando uma representação é "melhor" do que outra. Associada a este problema há ainda a questão de que nem todos os possíveis colapsamentos são de fato desejáveis. Por exemplo, pode ser mais conveniente manter uma representação separada para um esquema que possui um grande número de atributos do que colapsá-lo em outro esquema que o generaliza.

Por estas razões, escolhemos implementar o algoritmo de projeto sem uma pesquisa exaustiva no passo 1, mas permitindo ao projetista controlar o processo, se desejar, através de comandos que indiquem quais colapsamentos efetuar, quando houver mais de uma possibilidade, e quais colapsamentos (possíveis) não efetuar, por razões de projeto.

4. REPROJETO

4.1 Comandos de Reprojeto

Podemos classificar os comandos de reprojeto em dois tipos: locais, que afetam apenas um esquema de entidade ou relacionamento, ou não-locais, que só podem ser validados observando-se o esquema conceitual. Os comandos locais são:

adicione/remova

- atributo a um esquema
- chave a um esquema de entidade
- identificador a um esquema de relacionamento
- totalidade para um papel de um esquema de relacionamento

modifique

- nome de um atributo
- domínio de um atributo

Os comandos de reprojeto não-locais por sua vez são os seguintes:

adicione/remova

- esquema de entidade ou de relacionamento
- um esquema como especialização de um outro esquema

modifique

- nome de um esquema

Não consideramos a possibilidade de adicionar ou remover um participante de um esquema de relacionamento por entender que esta mudança reflete uma alteração profunda no significado do relacionamento e, portanto, deve ser efetuada removendo o esquema de relacionamento antigo e adicionando um novo, modificado.

O algoritmo de reprojeto pode processar, de uma só vez, uma lista de mudanças. Porém, a lista deve ser especificada de tal forma que, após cada mudança, o novo esquema conceitual seja correto. Por exemplo, não é possível remover um esquema de entidade antes de remover todos os esquemas de relacionamento em que ele participa.

Alguns comandos de reprojeto exigem a carga de novos dados como, por exemplo, um comando que adiciona um novo atributo que não admite valores nulos. Nestes casos, o plano de reprojeto conterá também comandos que pedirão ao projetista a carga dos novos dados, acompanhados das condições que deverão satisfazer. De forma semelhante, certos comandos de reprojeto exigem que o estado corrente do banco de dados satisfaça certas condições a fim de que possa ser gerado um estado consistente do novo esquema. O plano de reprojeto conterá então testes que asseguram as condições necessárias.

4.2 ALGORITMO DE REPROJETO

Esta seção esboça o algoritmo de reprojeto, ilustrando o seu funcionamento através de um exemplo detalhado.

O algoritmo de reprojeto em linhas gerais é o seguinte:

Entradas:

- um esquema conceitual E no modelo ERE
- uma representação relacional R para E
- uma seqüência s de comandos de reprojeto para E

Saídas:

- o novo esquema conceitual E'
- uma representação relacional R' para E'
- um *plano de reprojeto* para criar R' a partir de R e mapear o estado corrente de R (suposto consistente) em um estado consistente de R'

Passo 1:

- analise os comandos de reprojeto em s , verificando se produzem um novo esquema conceitual correto a partir de E ;

- produza o novo esquema conceitual E' ;
- reorganize a floresta de colapsamento da representação antiga R para refletir os comandos de reprojeção.

Passo 2:

- reprocessse o passo 1 do algoritmo de projeto, continuando a reorganização da floresta de colapsamento.

Passo 3:

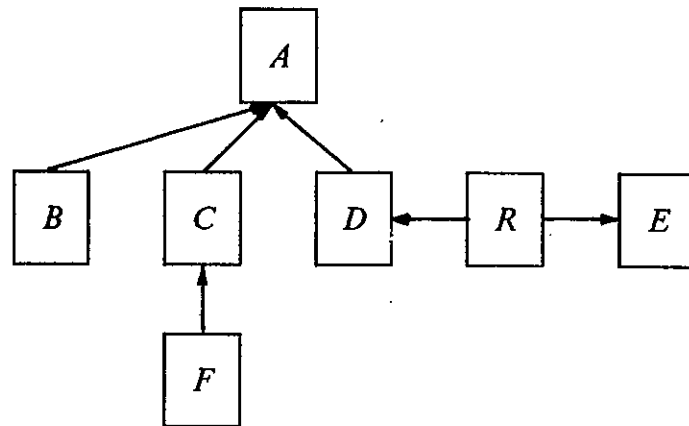
- compare a nova floresta de colapsamento com a antiga para gerar um plano de reprojeção contendo comandos para:
 - mapear a representação antiga R em uma representação R' para o novo esquema conceitual E' ;
 - pedir ao projetista para fornecer novos dados na medida do necessário;
 - verificar se os novos dados e o estado corrente são consistentes com as mudanças propostas;
 - mapear o estado corrente de R em um estado consistente de R' .

Para ilustrar o processamento do algoritmo de reprojeção, suponha que seja dado como entrada o esquema conceitual usado como exemplo na seção 3, a floresta de colapsamento produzida no exemplo e a seguinte lista de comandos de reprojeção:

- C_1 . remova F especialização de B
- C_2 . adicione B especialização de A
- C_3 . remova identificador D de R

Para estas entradas, o processamento do algoritmo de reprojeção se dará da seguinte forma.

O passo 1 aplicará os comandos de reprojeção, gerando um novo esquema conceitual, cujo grafo é o seguinte:



O passo 1 reorganiza ainda a floresta, gerando a seguinte seqüência de florestas:

$$f_0 = (A(C,D(R)), B(F), E)$$

$$f_1 = (A(C,D(R)), B, F, E)$$

$$f_2 = (A(C,D), R, B, F, E)$$

De fato, após processar o comando C_1 , o algoritmo transforma a floresta de colapsamento inicial f_0 na floresta f_1 pois F deixou de ser especialização de B e, portanto, não pode ser mais colapsado em B . O comando C_2 neste ponto não provoca nenhuma alteração na floresta. Já após processar o comando C_3 , o algoritmo transforma a floresta f_1 em f_2 pois D deixou de ser identificador de R , ou seja, R deixou de ser funcional em D . Portanto, R não pode ser mais colapsado em D .

O passo 2 continua a reorganizar a floresta na seguinte seqüência:

$$f_3 = (A(B,C,D), R, F, E)$$

$$f_4 = (A(B,C(F),D), R, E)$$

O algoritmo transforma a floresta f_2 em f_3 pois B passou a ser especialização de A e, portanto, pode ser colapsado em A . Para justificar f_4 , basta observar que F pode agora ser colapsado em C por ser especialização de C e não estar mais colapsado em B .

Na primeira fase do passo 3, o algoritmo compara a floresta inicial com a floresta final, detectando o movimento dos nós. Assim, comparando-se f_0 com f_4 :

$$f_0 = (A(C,D(R)), B(F), E)$$

$$f_4 = (A(B,C(F),D), R, E)$$

observa-se o seguinte movimento dos nós:

A, C, D, E - inalterados

B - transformado de raiz para nó interior

R - transformado de nó interior para raiz

F - mantido como nó interior, mas em árvore diferente

Iremos agora detalhar a geração do plano de reprojeto, analisando cada um dos nós que sofreu uma movimentação na floresta de colapsamento.

Começemos pelo nó *B*. Observe que, após o comando de reprojeto C_2 , *B* passou a ser especialização de *A*. Portanto é necessário identificar cada objeto em *B* com um objeto em *A*. Digamos, se *A* representasse pessoas e *B* empregados, seria necessário indicar, por exemplo, o CPF de cada empregado para identificá-lo como pessoa (assumindo que CPF é a chave de pessoa). Em termos da representação relacional, isto corresponde a iniciar o plano de reprojeto com as seguintes operações:

B1 defina K_A como uma chave de B^* e defina a dependência $B^*[K_A] \subseteq A^*[K_A]$;

B2 peça ao projetista para popular K_A em B^* , respeitando as dependências definidas em **B1**.

Note que B^* já contém os atributos da chave primária de *A*, portanto não é necessária nenhuma operação preliminar a **B1**.

Observe agora que *B* passou de raiz a nó interior da árvore cuja raiz é *A*. Isto significa que *B* deverá então ser colapsada em *A* ou, em termos da representação relacional, *B* deverá se tornar visão sobre a tabela básica A^* . Para tal é necessário acrescentar ao plano de reprojeto as seguintes operações:

B3 expanda A^* com os atributos de *B*;

B4 atualize a nova tabela A^* populando os novos atributos de cada tupla *t* de A^* com os valores que possuíam na tupla *u* de B^* , se existir uma tupla *u* de B^* tal que $t[K_A] = u[K_A]$, ou com nulos, se não existir tal tupla.

Para completar o colapsamento de *B* em *A* é necessário executar ainda as seguintes operações:

B5 remova a tabela B^* do banco de dados;

B6 remova B^* e todas as dependências sobre B^* da representação relacional;

B7 remova *B* como visão sobre B^* da representação relacional;

B8 acrescente *B* como visão sobre A^* à representação relacional.

As operações **B5** e **B6** deverão ocorrer apenas após o processamento de F pois F também é uma visão sobre B^* . Alternativamente, estas operações poderiam ser disparadas ao final do reprojeto por um processo que eliminasse todas as tabelas básicas sobre as quais não mais houvessem visões definidas.

Passemos agora ao nó F . Observe que F passou da árvore cuja raiz era B para a árvore cuja raiz é A . Isto significa que F deverá ser removido de B e colapsado em A ou, em termos da representação relacional, F deverá se tornar visão sobre a tabela básica A^* . Para tal é necessário acrescentar ao plano de reprojeto as seguintes operações:

F1 expanda A^* com os atributos de F ;

F2 atualize a nova tabela A^* populando os novos atributos de cada tupla t de A^* com os valores que possuíam na tupla u de F , se existir uma tupla u de F tal que $t[K_A] = u[K_A]$, ou com nulos, se não existir tal tupla.

Note que, como F já era uma especialização de C no esquema original, a visão F já continha a chave primária de A^* , por definição de representação relacional.

Para completar o colapsamento de F em A é necessário executar ainda as seguintes operações:

F3 remova F como visão sobre B^* da representação relacional;

F4 acrescente F como visão sobre A^* à representação relacional.

Finalmente, considere o nó R , que passou de nó interior da árvore cuja raiz é A para raiz. Isto significa que R deverá ser removido de A ou, em termos da representação relacional, a visão R deverá ser materializada:

R1 remova R como visão sobre A^* da representação relacional;

R2 acrescente R como tabela básica à representação relacional;

R3 popule a tabela R a partir da tabela A^* usando a definição original da visão R sobre A^* ;

R4 remova os atributos de R de A^* .

Com isto concluímos o exemplo de reprojeto.

5. CONCLUSÕES

Descrevemos neste trabalho, em linhas gerais, dois algoritmos que, em conjunto, formam a base de uma ferramenta para auxiliar o desenvolvimento de aplicações de banco de dados, cobrindo o projeto e a manutenção do esquema conceitual e de sua representação em um modelo de implementação. Adotamos uma extensão do modelo entidade-relacionamento para modelagem conceitual e o modelo relacional para modelo de implementação.

O algoritmo para obter representações otimizadas de esquemas conceituais está completamente implementado, com as características descritas no final da seção 3. O algoritmo para reprojeto encontra-se especificado.

Agradecimentos: agradecemos ao Prof. Alberto Laender pelas discussões que levaram à elaboração deste trabalho e a Anelise Pacheco pela implementação preliminar do algoritmo de projeto.

REFERÊNCIAS

- [1] Casanova, M.A., Tucherman, L., Gualandi, P.M., Pacheco, A. e Cavalcanti, M.R., "A Data Definition Language for an Extended Entity-Relationship Model", Relatório Técnico CCR072, Centro Científico Rio, IBM Brasil (junho 1989).
- [2] Casanova, M.A., Tucherman, L., Furtado, A.L. e Pacheco, A., "Optimization of Relational Schemas Containing Inclusion Dependencies", Proc. 15th Int'l. Conf. on Very Large Data Bases, Amsterdam, The Netherlands (Set. 1989).
- [3] Chen, P.P., "The entity-relationship model: toward a unified view of data", ACM Transactions on Database Systems 1:1 (1976), 9-36.
- [4] Dogac, A. e Chen, P.P., "Entity-Relationship Model in the ANSI/SPARC Framework", em *Entity-Relationship Approach to System Analysis and Design*, P.P. Chen (ed.), North-Holland (1981), 361-378.
- [5] Elmasri, R., Weeldreyer, J. e Heuner, A., "The Category Concept: an extension to the entity-relationship model", *Data and Knowledge Engineering 1*, North-Holland (1985), 75-116.
- [6] Kozaczynski, W. e Lillien, L., "An Extended Entity-Relationship (E²R) Database Specification and its Automatic Verification and Translation into the Logical Relational Design", Proc. of the Sixth Int. Conf. on Entity-Relationship Approach, New York (Nov. 1987), 497-513.
- [7] Lenzerini, M. e Santucci, G., "Cardinality constraints in the E-R model", em *Entity-Relationship Approach to Software Engineering*, North-Holland (1983).

- [8] Markowitz, V.M. e Shoshani, A., "On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model", Proc. ACM SIGMOD International Conference on the Management of Data, Portland, Oregon (Junho 1989), 430-439.
- [9] Pacheco, A., "Tradução Correta de Esquemas Entidade-Relacionamento em Esquemas Relacionais", Anais da 9º Congresso da Sociedade Brasileira de Computação, Uberlândia (julho 1989).
- [10] Scheuermann, P., Schiffner, G. e H. Weber, "Abstraction capabilities and invariant properties modelling within the Entity-Relationship approach" em *Entity-Relationship approach to System Analysis and Design*, P.P. Chen (ed.), North-Holland (1980).
- [11] Teorey, T.J., Yang, D. e Fry, J.P., "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", ACM Computing Survey 18:2 (junho 1986), 197-222.