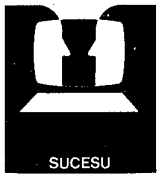
INFORMÁTICA 82

XV CONGRESSO NACIONAL DE INFORMÁTICA
II FEIRA INTERNACIONAL DE INFORMÁTICA

ANAIS



INFORMÁTICA 82

XV CONGRESSO NACIONAL DE INFORMÁTICA
II FEIRA INTERNACIONAL DE INFORMÁTICA

ANAIS

PROJETO DE BANCO DE DADOS ATRAVÉS DE EXEMPLOS

Claudio M.O. Moura / Marco A. Casanova
Instituto Latino Americano de Pesquisa em Sistemas IBM do Brasil Ltda. / Departamento de informática
Pontifícia Universidade Católica
Caixa Postal 1830 / Rua Marquês de São Vicente, 225 – Rio de Janeiro, RJ

Palavras-chave: Banco de Dados, projeto de banco de dados, modelo relacional, restrições, integridade, Query-by-Example, QBE, dependências de dados, semântica dos dados, inferência lógica.

Uma linguagem para definição de restrições, que apresenta uma notação uniforme para as dependências de dados mais comumente encontradas na literatura, é apresentada. As restrições de integridade são expressas nesta linguagem de maneira semelhante à especificação de consultas em Query-by-Example. Para administrar as restrições especificadas desta maneira, um dicionário auxiliar é descrito. Para concluir, é considerado o problema de se verificar se um determinado conjunto de restrições de integridade representa a semântica pretendida da empresa que se está modelando no Banco de Dados.

1. INTRODUÇÃO

A descrição de um banco de dados consiste em um conjunto de estruturas de dados e um conjunto de restrições de integridade determinando que valores os dados podem assumir, ou seja, que estados do banco são consistentes. Um estado de um banco de dado é dito consistente quando ele satisfaz todas as restrições de integridade. As transações do usuário são então obrigadas a preservar a consistência do banco de dados, isto é, mapear um conjunto de estados consistentes nele mesmo.

No modelo de dados relacional a estrutura básica consiste em tabelas ou relações. Várias classes de dependências têm sido estudadas em conexão com o modelo relacional. Exemplos destas dependências incluem dependências funcionais çCol, Ar], dependências multivaloradas [Fal, Za, BFH], dependências de junção [ABU, MMS] e dependências de inclusão [Fa2, CFP]. Contudo estas classes, e mesmo outras [YP, GJ], são casos especiais das dependências conhecidas como Dependências Implicacionais Embutidas e Estendidas (Extended Embedded Implicational Dependencies – XEID) [Fa3].

O objetivo deste trabalho é explorar as XEID's como ferramenta prática para modelar a semântica de um banco de dados. Nesta direção, uma linguagem de definição de restrições é inicialmente introduzida e, em seguida, o problema de verificar se um determinado conjunto de XEID's exprime a semântica pretendida da empresa é discutido.

A linguagem de definição segue o mesmo estilo do "Query-by-Example" (QBE) [Zℓ1, Zℓ2, Re] e, por esta razão, é chamada de Projeto de Banco de Dados Através de Exemplos, abreviadamente, Projeto-Através-de-Exemplos (PAE) ou, em inglês, "Design-by-Example" (DBE). XEIDs são expressas nesta linguagem de modo bastante semelhante a definição de consultas em QBE. Deste modo, a linguagem proporciona uma notação fácil de ser usada para exprimir as XEIDs e conseqüêntemente, as dependências de dados comumente encontradas na literatura.

A descrição do banco de dados modelará, então, a empresa através de um conjunto de tabelas, definidas usando QBE, e um conjunto de XEIDs sobre estas tabelas, especificadas através de nossa linguagem. O problema a que nos dirigimos em seguida é o de como verificar se o conjunto de restrições, expressas através das XEIDs, exprime a semântica pretendida da empresa. Não é nossa intenção, com isto, verificar se um conjunto de XEIDs é logicamente equivalente à alguma outra descrição formal da empresa, mas, em vez disso, verificar se o comportamento do banco de dados, conforme especificado pelas restrições dos dados, corresponde, intuitivamente, a representação do comportamento da empresa.

Concluimos esta introdução, descrevendo, brevemente, o conteúdo de cada seção. A Seção 2 revê as definições básicas do modelo relacional, define Dependências Implicacional Embutidas e Estendidas – (XEID) e mostra que várias dependências de dados são, na realidade, casos especiais de XEIDs. A Seção 3 descreve

a linguagem de definição de restrições, com o auxílio de uma série de exemplos. Seção 4 descreve o dicionário que auxilia o gerenciamento das restrições de integridade. Seção 5 é dedicada ao problema de verificar se um conjunto de XEIDs representa a semântica pretendida da empresa. Finalmente a Seção 6 contém conclusões e sugere direções para futuras pesquisas.

2. DEFINIÇÕES BÁSICAS

Um *esquema de relação* é uma expressão da forma R[U], onde R é o *nome da relação* e U= $(A_1, ...A_n)$ é uma seqüência finita de *atributos*. Uma *tupla* t em U sobre um conjunto D é uma seqüência $(a_1, ..., a_n)$ onde $a_1, ..., a_n \in D$. Uma *relação em* U sobre *D* é um conjunto de tuplas em U sobre *D*.

Se s é uma seqüência, | s | denota o comprimento de s. Se s= $(S_1, ..., s_n)$ é uma seqüência tal que | s |= |U |, e X= $(Ai_1, ..., A_i\downarrow)$, onde $i_1, ..., i_k$ são elementos distintos de (1, ..., n), então s[X] denota a seqüência $(Si_1, ..., S_{ik})$. Se s e s' são duas seqüências, então s= s' significa que s e s' têm o mesmo comprimento e os mesmos elementos.

Assim se r é uma relação em U, temos r[X] = (t [X] | tϵ).

Um *esquema do Banco de Dados* E= $(R_1[U_1], ..., R_m[U_m])$ é um conjunto de esquemas de relações. Um estado *I* do Banco de Dados E é um mapeamento que associa a cada esquema de relação $R_i[U_i]$ uma relação r_i em U_i. Algumas vezes representaremos um estado de E simplesmente por $r_1, ..., r_m$. Assumimos que $r_1, ..., r_m$ são relações sobre um dado conjunto *D*, o *domínio* de *I*.

Dado um esquema de Banco de Dados E= $(R1[U_1], ..., R_m[U_m])$, e um conjunto de variáveis $x_1, x_2, ...$, uma fórmula relacional em E é uma expressão da forma $R_i(x)$, onde $1< i <$ e x é uma seqüência de variáveis tal que $|x| = |U_i|$. Uma *igualdade* é uma expressão de forma x= y, onde x e y são variáveis. Uma *fórmula atômica* é uma igualdade ou uma fórmula relacional.

Fórmulas (envolvendo conectivos e quantificadores) e sentenças (fórmulas sem variáveis livres) são definidas como na lógica de primeira ordem [En, BM]:

Uma *dependência implicacional embutida e estendida (Extended Embedded Implicational Dependency – XEID* [Fa3] em E, é uma sentença F da forma $Ax_1, ..., Ax_m ((P_1 v ... vP_k) U E y_1 ... Ey_r (Q_1 v ... vQ_l))$ onde $P_1, ..., P_k$ são fórmulas relacionais em E, os *antecedentes* de F, e $Q_1, ..., Q_l$ são fórmulas atômicas, os *conseqüentes* de F, tal que x_i ocorra pelo menos uma vez em P_j.

Note que não necessitamos que F seja tipificada e não inter-relacional, diferentemente da definição original de XEID.

Dado um estado do banco de dados *I*= $(r_1, ..., r_m)$ de E com domínio *D* e variáveis $x_1, x_2, ...$, uma valorização em *D* é uma função v que atribui a cada variável x um elemento $v(x_1) \in D$. Dizemos que uma fórmula relacional $R_i(x_1, ..., x_m)$ em E é *verdadeira* em *I* para v sse (se e somente se) $(v(x_1), ..., v(x^\frown)) \in r_i$; dizemos ainda que x= y é *verdadeira* em *I* sse $(v(x) \Leftrightarrow v(y))$.

Então, dada uma XEID F em E da forma $Ax_1 ... Ax_m ((P_1 v ... v P_k) \prec Ey_1 ... Ey_r (Q_1 v ... vQ_l))$ dizemos que F é *verdadeira* no estado *I* do banco de dados se, para qualquer valorização v em *D* existe uma valorização v' em *D* tal que v' concorda com v em $x_1, ..., x_m$ e se $P_1, ..., P_k$ são *verdadeiras* em *I* para v', então $Q_1, ..., Q_l$ são *verdadeiras* em I para v'. Neste caso dizemos também que *I satisfaz* F.

Exemplificando, a XEID $Ax_1 ,,, Ax_5 (R_i(x_1, x_2, x_3)$ $vR_i(x_1, X_4, x_5) \Rightarrow x_2 = x_4)$ é verdadeira no estado do banco de dados $r_1, ..., r_m$ se quaisquer duas tuplas de r_i que concordem na primeira coluna devem também concordar na segunda. Como segundo exemplo, a XEID $Ax_1Ax_2Ax_3(R_i(x_1, x_2, x_3) \Rightarrow Ex4\ R_j(x_4, x_1, x_2))$, é verdadeira em $r_1, ..., r_m$ se a projeção de r_i nas duas primeiras colunas é um subconjunto da projeção de r_j nas suas duas últimas colunas.

Dizemos que uma XEID F é uma *conseqüência lógica* de um conjunto *F* de XEIDs se qualquer estado *I* do banco de dados que satisfaz todas as XEIDs em *F* também fatisfaz F.

Concluímos esta seção definindo algumas das dependências mais familiares, em termos de XEIDs.

Seja E= $(R_1[U_1], ..., R_m[U\frown])$ um esquema de um banco de dados.

Suponha que U_i= $(A_1, ..., A_k)$ e sejam X= $(A_{l1}, ..., A_{lp})$ e Y= $(A_{m1}, ..., A_{mq})$. Uma *dependência funcional* (FD) em E é uma XEID F da forma (abreviadamente R_i: X→Y).

$$Vx_1 \ldots Vx_n (R_i(\bar{u}) \ \Lambda \ R_i(\bar{v}) \Rightarrow \bigwedge_{jj=1}^{q} U_{mj} = V_{mj})$$

onde $\bar{u}$ e $\bar{v}$ são sequências de variáveis distintas conjunto $(x_1, \ldots, x_n)$ tais que $|\bar{u}| = |\bar{v}| = |U_i|$ e $\bar{u}$ e $\bar{u}[X] = \bar{v}[X]$.

Sejam X, Y e Z sequências de atributos distintos de U_i tais que Z contém todos os atributos de U_i que não estão em X ou Y. Uma *dependência multivalorada* (MVD) em E é uma XEID F da forma onde $\bar{u}$, $\bar{v}$ e $\bar{t}$ são sequências de

$$\forall x_1 \ldots \forall x_n (R_i(\bar{u}) \ \Lambda \ R_i(\bar{v}) \Rightarrow R_i(\bar{t}))$$

variáveis distintas do conjunto $(x_1, \ldots, x_n)$ tais que $|\bar{u}| = |\bar{t}| = |U_i|$, $\bar{u}[X] = \bar{v}[X] = \bar{t}[X] = [Y] = \bar{t}[Y]$ e $\bar{u}[Z] = E[Z]$. Abreviamos F como $R_1 : X \twoheadrightarrow Y|X$.

Sejam X e W sequências de atributos distintos de U_i e U_j, respectivamente, tais que |X|= |W|.

Uma dependência de inclusão (IND) em E é uma XEID F da forma onde $\bar{u}$ e $\bar{v}$ são seqüências de variáveis

$$Vx_1 \ldots Vx_p (R_i(\bar{u}) \Rightarrow \exists y_1 \ldots \exists y_q R_j(\bar{v}))$$

distintas de $(x_1, \ldots, x_p)$ e $(x_1, \ldots, x_p, \ldots, y_q)$, respectivamente, tais que $\bar{u}|X| = \bar{v}|X|$
Abreviamos F como $R_i|X| \subseteq R_j[W]$.

Esta lista cobre as dependências que mencionaremos nas próximas seções, mas poderia ser estendida para incluir outras dependências (para uma discussão mais detalhada sobre o poder expressivo das XEIDS ver |Fa3|).

Finalizamos esta seção mostrando o uso de XEIDs através de um exemplo.

Consideremos o esquema do banco de dados abaixo [Da]:

E =(FORNECEDOR [NF, CIDADE, SITUAÇÃO], FP [NF, NP, QUANTIDADE]).

Definimos as seguintes FDs em E (bem como suas abreviações):

(a) VfVcVsVc'Vs' (FORNECEDOR (f, c, s) Λ
Λ FORNECEDOR (f, c', s') ⇒ c=c' Λ
– FORNECEDOR: NF → CIDADE, SITUAÇÃO

(b) VfVcVsVf'Vs' FORNECEDOR (f, c, s)
Λ FORNECEDOR (f', c, s') ⇒
⇒ s=s'
– FORNECEDOR: CIDADE → SITUAÇÃO

(c) VfVpVqVp' (FP (f, p, q) Λ FP (f, p, q') ⇒ q=q')
– FP: NF, NP → QUANTIDADE

Definimos também a seguinte IND em E (e sua abreviação) que deveria ter sido introduzida no exemplo descrito em [Da]:

(d) VfVpVq (FP(f, p, q) ⇒ ∃ c ∃s FORNECEDOR (f, c, s))
– FP[NF] ⊆ FORNECEDOR[NF] .□

Estes exemplos serão utilizados na seção 3.2 para apresentar nossa linguagem de definição de restrições.

3. UMA LINGUAGEM DE DEFINIÇÃO DE RESTRIÇÕES

Descrevemos nesta seção uma linguagem de definição de restrições que estende a DDL do QUB [Zℓ1, Zℓ2] para incluir as XEID's. Na seção 3.1 descrevemos brevemente importantes características do QBE que são relevantes neste trabalho. Na seção 3.2 apresentamos nossa linguagem de definição de restrições, ilustrando, com exemplos, suas características básicas.

3.1 QUERY-BY-EXAMPLE

Query-by-Example (QBE) é uma linguagem de consulta a banco de dados relacionais da família do Cálculo Relacional [Co2]. Para formular uma consulta em QBE, o usuário cria, primeiramente, um *esqueleto* de tabela com o nome da relação, e os atributos que serão usados na sua consulta. Em seguida o usuário preenche os esqueletos de tabelas com "exemplos" de respostas possíveis para sua consulta. Por exemplo, considerando o esquema do banco de dados apresentado no fim da seção 2, a consulta "liste todos os números das peças fornecidas por qualquer fornecedor e cuja situação seja igual a 10" seria formulada do seguinte modo:

FORNECEDOR	NF	CIDADE	SITUAÇÃO
	f		10

SP	NF	NP	QUANTIDADE
	f	P.	

onde as letras sublinhadas funcionam como variáveis e P. indica a resposta desejada.

Quando uma consulta envolve condições mais complexas, ela pode ser formulada com o auxílio de uma caixa de CONDIÇÃO (CONDITION Box). Por exemplo, a consulta "liste todas as cidades cujo código de SITUAÇÃO seja menor que 10 e maior que 5" seria formulada do seguinte modo:

FP	CIDADE	SITUAÇÃO	CONDITION
	P.	s	s=(> 5 and < 10)

ou, alternativamente,

FP	CIDADE	SITUAÇÃO	CONDITION
	P.	s	s > 5 s < 10

Os dois exemplos são suficientes para dar uma idéia da sintaxe do QBE. Discutiremos agora duas características da linguagem. A primeira é que o QBE tem uma sintaxe bidimensional única, o que permite ao usuário grande flexibilidade para formular consultas, principalmente porque as tabelas podem ser preenchidas na sequência que o usuário achar mais natural. A segunda é que variáveis denotam entradas em tabelas, diferentemente da linguagem SEQUEL [CB] ou QUEL [HSW] ou até mesmo do Cálculo Relacional [Co2] onde variáveis denotam linhas inteiras de tabelas. Deste modo o QBE é considerado uma linguagem orientada para domínios [Pi]. Nossa linguagem de definição de restrições é baseada nestas duas características do QBE.

3.2 PROJETO DE BANCO DE DADOS ATRAVÉS DE EXEMPLOS

Primeiramente mostraremos nossa linguagem de definição de restrições por meio de um exemplo simples. Em seguida, definiremos mais precisamente a estrutura básica da linguagem e, finalmente, voltaremos a apresentar alguns exemplos. Nossa linguagem foi construída em cima dos comandos e convenções básicas do QBE.

Consideremos, de novo, o esquema do Exemplo 2.1:
B=(FORNECEDOR[NF, CIDADE, SITUAÇÃO], FP[NF, NP, QUANTIDADE]) juntamente com as restrições

(a) VfVcVsVc'Vs' (FORNECEDOR (f, c, s) Λ FORNECEDOR (f', c, s') ⇒ c=c' Λ s=s')

(b) VfVcVsVf'Vs' (FORNECEDOR (f, c, s) Λ FORNECEDOR (f', c, s') ⇒ s=s')

(c) VfVpVqVq' (FP(f, p, q) Λ FP(f, p, q,) ⇒ q=q')

(d) VfVpVq (FP(f,p,q) → FORNECEDOR (f,c,s))

As restrições (a) e (c), na realidade, especificam, respectivamente, que NF é a chave de FORNECEDOR e que NF, NP é a chave de FP. Assim sendo, ambas podem ser especificadas usando-se a definição de chave do QBE [Zℓ2].

O mesmo não acontece entretanto com os casos (b) e (d). Consideremos (d) primeiro. Proporemos definir (d) de modo bastante semelhante ao modo como uma consulta é especificada em QBE ou, melhor ainda, semelhante ao modo como uma consulta é armazenada em QBE para posterior processamento. O usuário chama uma caixa de COMANDO (COMMAND box) e indica através do comando I. CONSTR. < nome >, da nossa linguagem, que ele deseja especificar uma restrição que será identificada por "nome". O sistema responde devolvendo um esqueleto de tabela em branco; o usuário preenche então este esqueleto com o nome da relação e os atributos relevantes (podendo ser omitido aqueles que não forem usados). Esqueletos adicionais, se necessário, são obtidos como no BBE.

A definição da restrição continua sendo especificada pelo preenchimento da tabela com variáveis exemplos. As linhas que correspondem ao lado direito da XEID são indicadas pela colocação do operador C., o operador CONSEQüÊNCIA, no campo de comando (coluna abaixo do nome da relação em uma tabela). A especificação (final da restrição)estaria na tela da seguinte forma:

FP	NF
	f

FORNECEDOR	NF
C.	*f*

COMMAND
I. CONSTR. A1

A caixa de COMANDO permanece para identificar, na modalidade de entrada, que uma restrição está sendo especificada. O usuário termina a operação de especificação de uma restrição pressionando a tecla ENTRADA.

O exemplo acima ilustra como seria especificada uma XEID cujo conseqüente é uma fórmula relacional. Consideremos agora a restrição (b) (a FD FORNECEDOR: CIDADE → SITUAÇÃO), que é uma XEID cujo lado direito da fórmula é uma igualdade. Para especificar este tipo de XEID nós introduzimos a caixa de CONSEQUÊNCIA, e a restrição (b) seria formulada do seguinte modo:

FORNECEDOR	CIDADE	SITUAÇÃO
	c *c*	*s* *s'*

CONSEQUÊNCIA
s = *s'*

COMMAND
I. CONSTR. A2

Note que uma caixa de CONSEQUÊNCIA foi trazida para a tela.

Estes dois exemplos apresentam, de maneira superficial, a idéia geral do Projeto de Banco de Dados através de Exemplos.

Deve, neste momento, estar bastante evidente que nossa linguagem de definição de restrições segue, tão perto quanto possível, as características do QBE. É importante observar também que existe um mapeamento direto da notação básica de uma XEID e nossa linguagem. No entanto, devido a sua sintaxe bidimensional, nossa linguagem permite ao usuário máxima flexibilidade na formulação de restrições. Além disto, os quantificadores e conectivos ficaram implícitos na linguagem, o que simplifica significamente a sintaxe.

Seja $E = (R_1[A_{11} \ldots A_{1\,m_1}], \ldots, R_n[An_1 \ldots An_{m_n}])$ um esquema de um banco de dados. De um modo geral, uma XEID F de E da forma $Vx_1 \ldots Vx_p(P_1 v \ldots vP_r \Rightarrow Ey_1 \ldots Ey_q (Q_1 v \ldots vQ_s))$ é formulada em nossa linguagem como se segue:

(i) para cada antecedente P_i, se P_i é de forma $R_k(u_1, \ldots, u_{m_k})$, uma tabela, com o formato abaixo, é criada:

R_k	$A_{k\,1}$	...	$A_{k\,m_k}$
	u_1	...	u_{m_k}

Por convenção, se u_j não é usada na XEID, a coluna correspondente, A_{k_j}, pode ser omitida ou a variável exemplo u_j ser deixada em branco.

(ii) para cada conseqüente Q_i , se Q_i é de forma $R_k(u_1, \ldots, u_{m_k})$, uma tabela, com o formato abaixo, é criada:

R_k	$A_{k\,1}$	...	$A_{k\,m_k}$
C.	u_1	...	u_{m_k}

(A mesma convenção de (i) se aplica). Se por outro lado Q_1 é da forma u= v uma linha, da forma abaixo, é inserida em uma caixa de CONSEQUÊNCIA.

CONSEQUÊNCIA
.
.
.
u= v
.
.
.

O operador I. CONSTR. < nome > colocado na caixa de COMANDO informa ao sistema o nome da restrição sendo definida (< nome > segue as mesmas convenções usadas para tabelas em QBE).

Neste ponto concluímos a descrição do caso geral.

Mostramos, em seguida, outros exemplos de XEIDs definidas em nossa linguagem. Seja C= (EMP[NOME, HABIL, PROJ, GER], NP[NOME, PROJ], PG[PRJ,GER]) um banco de dados. A dependência multivalorada (MVD) EMP: NOME →→ HABIL|PROJ, GER seria definida como:

EMP	NOME	HABIL	PROJ
	n		*p*
	n	*h*	
C.	*n*	*h*	*p*

COMMAND
I. CONSTR. A3

Se desejamos garantir que um gerente deve ser também um funcionário, escrevemos:

EMP	NOME	GER
C.	*g*	*g*

COMMAND
I. CONSTR. A4

Como indicado pelos atributos, podemos assumir que NP e PG são redundantes no sentido em que se existe uma tupla NP(n,p) e uma PG(p,g) então existirá um h tal que exista uma tupla EMP(n, h, p, m). Isto é expresso da seguinte maneira:

NP	NOME	PROJETO
	n	*p*

PG	PROJ	GER
	p	*g*

EMP	NOME	PROJ	GER
C.	*n*	*p*	*g*

COMMAND
I. CONSTR. A5

Nota: segundo nossa convenção não incluímos a coluna HABIL de EMP.

Isto conclui a descrição do núcleo de nossa linguagem de definição de restrições. A próxima seção discute um dicionário que armazena e atualiza as restrições.

4. Um Dicionário para Administrar as Restrições

Considerando que a descrição de um banco de dados pode envolver um grande número de XEIDs, somente a catalogação destas pode ser uma tarefa trabalhosa e demorada. Por isso, introduzimos um dicionário de restrições que ajuda o DBA nesta tarefa. Descreveremos primeiramente a estrutura do dicionário e em seguida os comandos para catalogar e atualizar as restrições.

4.1 O Dicionário de Restrições

As restrições são armazenadas do mesmo modo que programas o são em QBE. O dicionário de restrições consiste de (a) uma tabela para armazenar as restrições e cuja organização é completamente transparente ao usuário; (b) uma tabela do sistema, RESTRIÇÕES, com atributos ANTECEDENTE, CONSEQUENTE, NOME, IDUSUÁRIO e COMENTÁRIOS; que se assemelha também a tabelas do sistema do QBE. A tabela RESTRIÇÕES contêm os nomes da relações que participam em um antecedente de uma XEID, os nomes das relações que participam como conseqüente de uma XEID ou uma indicação que uma caixa de CONSEQUÊNCIA é usada para especificar o conseqüente, o nome da XEID, uma coluna IDUSUÁRIO que especifica o banco de dados ao qual a XEID se refere e uma coluna para comentários. A tabela RESTRIÇÕES é atualizada automaticamente sempre que uma nova restrição é definida, com exceção do campo COMENTÁRIOS que é atualizado diretamente pelo usuário.

Deste modo, após os exemplos da seção 3.2 terem sido definidos, a tabela RESTRIÇÕES conteria:(pág. 344).

As informações podem ser recuperadas desta tabela da mesma maneira que de qualquer outra. Outras operações sobre o dicionário são permitidas, o que discutiremos na próxima seção.

4.2 Operações sobre o Dicionário de Restrições

Mostraremos nesta seção como exibir, eliminar e modificar restrições definidas anteriormente e portanto já armazenadas no dicionário. Note que a operação de inserção não é necessária, conforme discutido na seção 3.2.

Como já mencionamos, a tabela RESTRIÇÕES pode ser consultada como qualquer outra. Assim, voltando ao exemplo do fim da seção 4.1, a consulta

RESTRIÇÕES	ANTECEDENTE	NOME	IDUSUÁRIO
	EMP	P.	P.

produziria a seguinte tabela como resposta, que mostra o nome das restrições onde a relação EMP entra como antecedente:

RESTRIÇÕES	NOME	IDUSUÁRIO
	A3	C
	A4	C

Para exibir uma restrição, o usuário, na modalidade de entrada, com o *operador de exibição de restrição*, P. CONSTR. < nome > na caixa de COMANDO. Assim, assumindo que o usuário tem IDUSUÁRIO igual a C, para ele exibir a restrição A3, ele entraria

COMMAND
P. CONSTR. A3

que resultaria em

EMP	NOME	HABIL	PROJ
	n		*p*
	n	*s'*	
C.	*n*	*s*	*p*

e um usuário, com o IDUSUÁRIO igual a B, exibiria A2

COMMAND
P. CONSTR. A2

que produziria

FORNECEDOR	CIDADE	SITUAÇÃO
	c	s
	c	s'

CONSEQUÊNCIA
$s=s'$

O operador de exibição de restrição oferece ainda a opção de se pesquisar sequencialmente a tabela RESTRIÇÕES. Juntamente com a exibição de uma restrição, o sistema exibe a mensagem "PRESSIONE A TECLA DE ENTRADA PARA VER MAIS RESTRIÇOES"; se o usuário pressiona a tecla de entrada a restrição seguinte na tabela RESTRIÇÕES, para o mesmo IDUSUÁRIO, é exibida (a ordem das restrições na tabela é imaterial). Para terminar a operação de exibição.

5. Testando a Descrição do Banco de Dados

Esta seção é dedicada ao problema de se verificar se um conjunto de XEIDs representa a semântica pretendida da empresa. Nossa abordagem não é verificar se um conjunto de XEIDs dado é logicamente equivalente a uma outra descrição formal da empresa. Em vez disso, nós apresentamos, rapidamente, ferramentas para verificar se a descrição do banco de dados possui determinadas características da linguagem. Na seção 5.1 introduzimos uma nova característica da linguagem para exprimir inferência lógica. Na seção 5.2 discutimos uma ferramenta para projeto baseada na idéia de verificação com dados de teste. Uma outra possibilidade, que não exploramos aqui, seria utilizar Bancos de Dados de Armstrong [Fa4] para a mesma finalidade.

5.1 O Operador de Inferência

Suponhamos que desejemos testar o comportamento de um conjunto F de XEIDs. Uma possibilidade é verificar se F implica uma nova XEID F que acreditamos ser verdadeira em qualquer estado do banco de dados que satisfaça F. Assumimos que F é o conjunto de restrições armazenadas no dicionário para o IDUSUÁRIO que está fazendo esta operação. Assim, é suficiente somente definir F. Isto é feito exatamente como na seção 3.2 com a exceção de que o operador de inclusão de restrição (I. CONSTR. < nome >) não é usado na caixa de comando mas substituído pelo *operador de conseqüência lógica* L. (não é necessário especificar < nome > foi F não será armazenada.

Seja, por exemplo, $E=(R_1[ABC], R_2[DE])$ um banco de dados onde $F=(R_1[AB] \subset R_2[DE], R_1[AC] \subset R_2[DE], R_2: D \rightarrow E)$. Para verificar se F implica logicamente F $(F|=F)$, onde F é $\forall a \forall b \forall c(R_1(a, b, c) \Rightarrow b=C)$, um usuário G entraria com F da maneira usual, mostrada em 3.2, entrando primeiramente com o comando I. CONSTR < nome > m cada etapa

(a)

R_1	A	B
	a	b

R_2	D	E
C.	a	b

COMMAND
I. CONSTR. B1

(b)

R_1	A	C
	a	c

R_2	D	E
	a	c

COMMAND
I. CONSTR. B2

(c)

R_2	D	E
	a	b
	a	b'

CONSEQUÊNCIA
$b = b'$

COMMAND
I. CONSTR. B3

e então entra com F em um esqueleto em branco, após entrar com o operador L.

R_1	B	C
	b	c

CONSEQUÊNCIA
b = c

COMMAND
L.

e o sistema responderá "VERDADEIRO" ou "FALSO" na parte inferior da tela, ou seja, na área de comunicação com o usuário.

Deixamos ao leitor a verificação de que realmente F implica logicamente em F, embora isto fique claro pela nossa escolha das variáveis para o exemplo.

Deve ser ressaltado neste ponto que a capacidade de inferência apresenta sérias limitações. Primeiramente o problema da decidibilidade das Dependências Implicacionais Estendidas (Extended Implicational Dependencies – EID), que são um caso particular das XEIDs, é indecifrável [CLM], ou seja, testar se uma EID é conseqüência lógica de um conjunto de fórmulas F é insolúvel. Assim temos que limitar o uso do operador de

inferência lógica não permitindo sua generalização para qualquer XEID. Na verdade, não sabemos se o problema da decidibilidade tem solução até mesmo para classes mais simples de dependências tais como Dependências Multivaloradas Embutidas (Embedded Multivalued Dependencies – EMVD) [SW] ou FDs juntamente com INDs [CFP].

Sabemos, contudo, que existe um procedimento de decisão polinomial para MVDs [Sa]. Assim, o operador de inferência lógica pode ser eficientemente suportado para o caso que *F* e F são MVDs ou FDs.

Concluimos esta seção observando que a idéia de validar um conjunto F de XEID's usando inferência lógica pode ser generalizado através do conceito de Bancos de Dados de Armstrong [Fa3], [Fa4] para F, isto é, um estado do banco de dados que satisfaz todas as consequências lógicas de F e nenhuma outra. Este conceito foi usado em [SM] para propor uma ferramenta prática para projeto de banco de dados, com espírito bastante semelhante do Projeto-através-de-Exemplos.

5.2 Experiências com Dados de Teste

Uma segunda possibilidade de se verificar um conjunto *F* de XEIDs é a de se usar dados de teste. O usuário define um estado *I* de um banco de dados preenchendo os esqueletos de tabelas com dados de teste e em seguida pergunta ao sistema se *I* satisfaz *F*. O sistema responderia "SIM" ou "NÃO" e neste último caso indicaria quais XEIDs em *F* não são satisfeitas por *I*.

Sugerimos aprimorar a idéia acima do seguinte modo. Em vez de somente indicar quais as XEIDs em *F* que não são satisfeitas por *I* o sistema indicaria também porque elas são falsas. O exemplo seguinte ajuda a entender esta sugestão. Considere o esquema do banco de dados da seção 5.1. E=(R_1[ABC], R_2[DE], com as restrições F=(R_1[AB] ⊂ R_2[DE], R_1[AC] ⊂ R_2[DE], R_2: D→E). um usuário entraria então com dados de teste, preenchendo esqueletos, em branco com o nome da tabela e os nomes dos atributos da forma usual. Em seguida, preencheria as colunas com os dados de teste; um *operador de teste* T. colocado no campo de comando indicaria que a tabela contém dados de teste e ao mesmo tempo após pressionar a tecla ENTRADA solicita ao sistema que verifique os dados de teste contra *F*. Exemplificando

R_1	A	B	C
T.	*a*	*b*	*c*

O sistema responderia que o estado *I* do banco de dados que acabou de ser definido viola alguma restrição, digamos R_1[AB] ⊂ R_2[DE]. Interativamente com o usuário o sistema iniciaria a transformação de *I* em um estado consistente, em estágios sucessivos:

(a)

R_2	D	E
	a	*b*

– para satisfazer R_1[AB] ⊂ R_2[DE]

(b)

R_2	D	E
	a	*b*
	a	*c*

– para satisfazer R_1[AC] ⊂ R_2[DE]

(c)

R_2	D	E
	a	*b*
	a	*b*

– para satisfazer R_2 : D → E

O passo anterior merece um comentário adicional. Ao concluir a etapa (b) o sistema informaria ao usuário que a restrição R_2 : D → E não foi satisfeita no estado *I* que acabou de ser criado através de uma caixa de CONSEQUÊNCIA

CONSEQUÊNCIA
b = c

o usuário então prosseguiria com a etapa (c) já descrita.

A abordagem desta seção, torna-se bastante atraente no seguinte cenário. Suponhamos que a aplicação sendo modelada já exista implementada como um sistema convencional ou mesmo manual. Assim sendo, o projetista do banco de dados tem à sua disposição dados reais. Ele pode então começar uma série de testes construindo uma amostra do banco de dados a partir dos dados disponíveis.

Um destes testes seria, necessariamente, verificar se o estado-amostra é um estado consistente do banco de dados. Acreditamos que este tipo de verificação ajuda a localizar onde as especificações do banco de dados não concordam com a semântica dos dados reais. (Obviamente o estado-exemplo pode ser inconsistente simplesmente por ignorar certos relacionamentos dos dados).

Naturalmente, o enfoque descrito nesta seção também é válido quando dados reais não estão disponíveis. Neste caso, o projetista do banco de dados tem a tarefa adicional de gerar um estado exemplo do banco de dados que corresponda à uma simulação do mundo real sendo representado.

Isto conclui a apresentação geral das ferramentas de validação que, juntamente com a linguagem de definição de restrições, constituem o sistema de Projeto de Banco de Dados através de Exemplos.

6. CONCLUSÕES

Descrevemos neste trabalho uma linguagem de definição de restrições que expressa, de maneira uniforme, a maioria das dependências de dados encontradas na literatura. A linguagem tem uma sintaxe direta, no estilo do QBE, e não requer conhecimento ou treinamento em notação lógica. Descrevemos também um dicionário para armazenar as restrições, mais uma vez tentando manter, tanto quanto possível, o mais próximo da sintaxe padrão do QBE.

Em adição à linguagem de definição de restrições, o Projeto-através-de-Exemplos contêm ferramentas para validar o projeto de um banco de dados. Apresentamos, de uma maneira geral, duas delas, usando o conceito de

inferência lógica e dados de teste, e sugerimos uma terceira, o uso de bancos de dados de Armstrong.

Resumindo, o Projeto de Banco de Dados através de Exemplos consiste de uma linguagem para documentar a semântica do banco de dados e ferramentas para verificar se esta semântica corresponde realmente à pretendida da empresa. Acreditamos que tal sistema pode ser útil nos estágios iniciais do projeto de um banco de dados onde esteja sendo enfatizada a importância da correção semântica do modelo da empresa.

Este trabalho documenta o estágio inicial do Projeto-através-de-Exemplos. Esforço adicional é o necessário para consolidar a linguagem de restrições e o interface com o dicionário, e que poderá ser útil como especificação e documentação para a construção de um protótipo. As ferramentas de validação baseadas em inferência lógica (ver seção 5.1) dependem dos algoritmos existentes hoje, se nos restringirmos à dependências familiares, como FDs e MVDs. Se, contudo, pretendermos incluir também INDs, por exemplo, um esforço considerável é ainda necessário. Por outro lado uma ferramenta de validação baseada em dados de teste pode ser implementada para a classe completa de XEIDs, dependendo do grau de sofisticação desejado.

BIBLIOGRAFIA

(ABU) A.V. Alho, C. Beeri and J.D. Ullman, "The Theory of Joins in Relational Databases", ACM-TODS, Vol. 4, nº 3, Sep. 1979.

(Ar) W.W. Armstrong, "Dependency Structures of Data Relationships", Proc. IFIP 74, North Holland, 1974.

(BFH) C. Beeri, R. Fagin and J.H. Howard, "A Complete Axiomatization for Functional and Multivalued Dependencies in Database Rekations", Proc. ACM-SIGMOD Int. Conf. Management of Data, Toronto, Canada, 1977.

(BM) J.L. Bell and M. Machover, "A Course in Mathematical Logic", North-Holland, 1977.

(CB) D.D. Chamberlin and R.F. Boyce, "SEQUEL: A Structured English Query Language", Proc. 1974 ACM-SIGMOD Workshop on Data Description, Access and Control, 1974.

(CFP) M.A. R. Fagin and C. Papadimitriou, "Inclusion Dependencies and their Interactions with Functional Dependencies", Conf. Principles of Databases Systems, Los Angeles, Calif. Mar. 1982.

(CLM) A.K. Chandra, H.R. Lewis and Johan A. Makowshy, "Embedded Implicatoinal Dependencies and their Inference Problem, Report RC8757, IBM Research Lab, Yorktown Heights, N.Y., Mar. 1981.

(CO1) E.F. Codd, "A Relational Model of Data for Large Shared Data Bases", Comm. ACM, Vol. 13, Jun. 1970.

(Co2) E.F. Codd, "Relational Completeness of Data Base Sublanguages", Data Base Systems, Courant Computer Science Simposia Series, Vol. 6, Englewood Cliffs, N.J., Prentice-Hall, 1972.

(Da) C.J. Date, "An Introduction to Database Systems", Addison-Wesley, 3rd edition, 1981.

(En) H.B. Enderton, "A Mathematical Introduction to Logic", Academic Press, 1972.

(Fal) R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases", ACM-TODS, nº 2, Vol. 3, Sep. 1977.

(Fa2) R. Fagin, "A Normal Form for Relational Databases that is based on Domains and Keys. ACM-TODS. Vol. 6, nº 3, Sep. 1981.

(fa3) R. FAgin, "Horn Clauses and Database Dependencies", Proc. ACM-SIGACT Symp. Theory of Computing, 1980.

(Fa4) R. Fagin, "Armstrong Databases", IBM Report RJ 3440, San Jose, Calif., May 1982.

(GJ) J. Grant and B.E. Jacobs, "On Generalized Dependencies", *to appear.*

(HSW) G.D. Held, M.R. Stonebraker and E. Wong, "INGRES – A Relational Data Base Systems", Proc. NCC44, 1975.

(MMS) D. Maier, A. Mendelzon and Y. Sagiv, "Testing Implications of Data Dependencies, ACM-TODS, Vol. 4, nº 4, Dec. 1979.

(Pi) A. Pirote, "High Level Data Base Query Languages", Advances in Data Theory, Vol. 1, ed. H. Gallaire, J. Minker and J.M. Nicolas, Plenum Press, N.Y., 1978.

(Re) P. Reisner, "Human Factors Studies of Databases Query Languages: A Survey and Assessment", Report RJ3070, IBM Research Lab., San Jose, Calif. Mar. 1981.

(Sa) Y. Sagiv, "An Algorithm for Inferring Multivalued Dependencies that works also for a Subclass of Propositional Logic", VIVCDC-R-79-954, Dept. Computer Sciense, Univ. Illinois, Urbana, Jan. 1979.

(SM) A.M. Silva and M.A. Melkanoff, "A Method for Helping Discover the Dependencies of a Relation", Advances in Data Base Theory Vol. 1, ed. H. Gallaire, J. Minker and J.M. Nicolas, Plenun Press, N.Y., 1978.

(SU) F. Sandri and J.D. Ullman, "A Complete Axiomatization for a Large Class of Dependencies in Relation Databases", 1980 ACM Symp. 1980 ACM Symp. Theory of Computing, 1980.

(SW) Y. Sagiv and S, Walecka, "Subset Dependencies as an Alternative to Embedded Multivalued Dependencies", Tech. Rep. UIUCDCS-R-79-980, Dept. Comp. Science, University of Illinois, 1979.

(YP) M. Yannakakis and C. Papadimitriou, "Algebraic Dependencies", Proc. 21st. IEEE Symp. Found. Computer Science, 1980.

(Za) C. Zaniolo, "Analysis and Design of Relational Schemata for Database Systems", Ph. D. Dissertation, Tech. Rep. UCLA-ENG-7669, U. California, Los Angeles, Calif., Jul. 1978.

(Z11) M.M. Zloof, "Query-by-Example", Proc. National Computer Conference, AFIPS Press, Vol. 44, 1975

(Z12) M.M. Zloof, "Security and Integrity whithin the Query-by-Example Data Base Management Language", Report RC6982, IBM Research Lab., Yorktown Heights, N.Y., Feb. 1978.

TABELA RESTRIÇÕES:

RESTRIÇÕES	ANTECEDENTE	CONSEQUENTE	NOME	IDUSUÁRIO	COMENTÁRIOS
	FP	FORNECEDOR	A1	B	SP.NP é sub de FORNECEDOR. NS
	FORNECEDOR	C.	A2	B	CIDADE → SITUAÇÃO
	EMP	EMP	A3	C	NOME →→ HABIL
	EMP	EMP	A4	C	"gerente é funcionário
	NP	EMP	A5	C	–
	PG	EMP	A5	C	–