UNIVERSIDADE DE CAMPINAS

INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E CIÊNCIA DA COMPUTAÇÃO

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

"SISTEMA DE OPERAÇÕES EM ÁLGEBRA RELACIONAL NÃO-NORMALIZADO"

HILDA CARVALHO DE OLIVEIRA

ORIENTADORA: PROF.ª DR.ª CLAUDIA M. BAUZER MEDEIROS

SISTEMA DE OPERAÇÕES EM ÁLGEBRA RELACIONAL NÃO-NORMALIZADO

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pela Sra. **Hilda Carvalho de Oliveira** e aprovada pela Comissão Julgadora.

Campinas, 04 de setembro de 1991.

Prof.ª Dr.ª Claudia Maria Bauzer Medeiros
Orientadora CLC

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

Dedico este trabalho a meu filho
Yuri

Agradecimento especial a meus pais
Severo e Nair

AGRADECIMENTOS

À Profª Drª Claudia Maria Bauzer Medeiros pela orientação, compreensão e incentivo para que este trabalho fosse realizado com êxito.

Ao Prof. Dr. Geovane Cayres Magalhães e ao Prof. Dr. Waldemar Setzer pela atenção dedicada a este trabalho e pelas sugestões dadas.

Aos colegas e funcionários do Departamento de Estatística, Matemática Aplicada e Computacional, IGCE, Unesp, Rio Claro-SP, em especial aos colegas da área de Computação, que apoiaram e deram condições para o desenvolvimento desta pesquisa.

Ao aluno estagiário do curso de bacharelado em Ciência da Computação da Unesp de Rio Claro Silvio Tadeu pelo contribuição na implementação de estruturas e algoritmos aqui propostos.

Ao Prof. Benedito René Fischer e à Profª Deisy P. M. Fischer pelo incentivo e apoio.

A todos meus amigos, que incentivaram e desejaram a concretização desta Tese.

RESUMO

Esta tese propõe um conjunto de estruturas e algoritmos para a implementação de operadores da álgebra não-normalizada. Os algoritmos foram validados por uma implementação parcial, utilizando um conjunto de estruturas dinâmicas, permitindo realizar as operações algébricas e de atualização nas relações aninhadas.

A álgebra sugerida resulta de um estudo comparativo de diversos trabalhos publicados sobre conjuntos de operadores para relações NF^2. Os operadores selecionados foram os de aninhamento, desaninhamento, seleção, projeção, união, intersecção, diferença, produto cartesiano e junção.

ÍNDICE

1. APRESENTAÇÃO DO MODELO RELACIONAL NÃO-NORMALIZADO

1.1 INTRODUÇÃO

Atualmente, vários sistemas de gerenciamento de banco de dados (SGBDs) relacionais encontram-se comercialmente disponíveis. Sistemas como o INGRES, ORACLE, entre outros, providenciam alto grau de independência dos dados, podendo combinar os dados armazenados de modo bem flexível, além de oferecerem linguagens de consultas relativamente fáceis de se aprender e usar.

Um conceito fundamental do modelo relacional são as formas normais, sendo que as relações devem estar pelo menos na primeira forma normal ou 1NF (*1st Normal Form*). Segundo a 1NF, todos os atributos de uma relação devem ser atômicos.

Apesar da ampla área de utilização do modelo relacional, os seus registros estruturados pela 1NF não são muito adequados a uma série de aplicações. Entre elas, podemos citar: gerenciamento de textos, processamento de imagens, porcessamento de mapas, manipulação de dados estatísticos, aplicações de engenharia e sistemas CAD/CAM. Esses tipos de aplicações freqüentemente requerem estruturas hierárquicas aninhadas, além de uma variedade de tipos de tuplas (relações) diferentes para representar todas as informações necessárias. Por razões semânticas e de desempenho, tais objetos não podem ser armazenados como simples relações planas (em 1NF); a estrutura hierárquica natural não deve ser quebrada.

A generalização do modelo relacional proporciona uma maneira elegante de integrar relações planas e estruturas hierárquicas num único modelo de dados, conservando o poder de expressão das linguagens de consultas relacionais de alto nível. A idéia chave é permitir que os atributos de uma relação sejam compostos (*i.e.*, conjuntos de

atributos) e/ou multivalorados (um só campo de uma única tupla pode ter um conjunto de valores, equivalendo a uma sub-relação). Como isso se opõe à restrição 1NF, relações desse tipo são chamadas relações não-normalizadas ou relações em não-1NF. A liberdade de se compor atributos compostos e multivalorados à vontade justifica que essas relações também sejam conhecidas como relações aninhadas.

Desse modo, pode-se dizer que o modelo relacional pode ser clasificado, de maneira não-ortodoxa, em duas classes: Modelo Relacional Normalizado (MRN) e Modelo Relacional Não-Normalizado (MRNN).

O MRNN também é conhecido por outros nomes: modelo relacional com atributos de relações multivalorados, modelo relacional estruturado hierarquicamente e banco de dados em não-1NF, abreviado por NFNF ou NF^2 (*Non-First-Normal-Form*), o qual parece ter-se imposto sobre os demais.

A "principal falha", ou talvez a "única", do MRN, segundo os defensores do MRNN, é a restrição das relações à primeira forma normal de Codd.

Segundo Setzer [Set86], costuma-se evitar justificativas para o uso da 1NF na literatura do MRN. As poucas que são encontradas, inclusive as de Codd, não são significativas. Roth, Korth e Silberchatz [RKS88] mencionam que o próprio Codd, apenas um ano depois do seu artigo introdutor da 1NF, declarou que há certos casos em que é conveniente transformar uma relação normalizada em não-normalizada, através de uma operação que chama de *fatoração*.

Há diferentes maneiras de se representar uma relação aninhada, de acordo com o ponto de vista teórico (cálculo e álgebra relacional). Uma delas é a usada por Fischer e Thomas [FiT83], a qual foi adotada nesta tese, e é exemplificada a seguir.

Considere um esquema de banco de dados (BD) em NF^2 correspondendo às informações sobre os empregados de uma certa empresa, onde cada empregado tem um conjunto de dependentes. Cada dependente é identificado por um nome e uma data de nascimento (dia/mês/ano). Todo empregado também possui um conjunto de habilidades (formação profissional). À cada tipo de habilidade corresponde um

conjunto de informações. Tais informações consistem nos anos em que o empregado recebeu uma titulação relativa à habilidade especificada, bem como na cidade em que recebeu cada titulação.

O esquema é formado pelas quatro regras abaixo:

EMPREGADOS = (ENOME, DEPENDENTES, HABILIDADES). (1)

Com exceção do atributo ENOME, que é simples (não é composto) e atômico (monovalorado), os outros dois são compostos e multivalorados:

DEPENDENTES = (NOME, DATA_NASCIMENTO); (2)
HABILIDADES = (TIPO, DATAS_TITULAÇÃO). (3)

Os atributos componentes NOME, DATA_NASCIMENTO e TIPO são simples e atômicos, enquanto que DATAS_TITULAÇÃO é multivalorado e composto segundo a seguinte regra:

DATAS_TITULAÇÃO = (ANO, CIDADE). (4)

A tabela na página seguinte exemplifica uma relação sobre o esquema acima, onde ENOME é o atributo chave (supõe-se que não haja dois empregados com nomes iguais).

Cada valor de ENOME identifica uma tupla da relação. Desse modo, a relação EMPREGADOS na página seguinte contém duas tuplas.

A cada atributo composto está associado um conjunto de valores independentes, ou seja, uma relação. DEPENDENTES e HABILIDADES são sub-relações de EMPREGADOS, assim como DATAS_TITULAÇÃO é uma sub-relação de HABILIDADES. Desse modo, segundo a notação de [FiT83], cada conjunto de valores é representado por um atributo composto. Se esse atributo for composto de mais de um atributo, a cada valor de um atributo simples corresponde um valor dos demais atributos componentes, o qual pode ser atômico ou multivalorado, dependendo do referido atributo ser simples ou composto, respectivamente.

Veja que na primeira tupla de EMPREGADOS, Carlos é um advogado que obteve titulações nessa carreira em duas épocas distintas: em 1984, em Campinas, e em 1985 em S.Paulo. Dizemos, então, que os valores

(advogado ((1984 Campinas) (1985 S.Paulo)))

formam uma subtupla de HABILIDADES, bem como (1984 Campinas) constitui uma subtupla de DATAS-TITULAÇÃO.

EMPREGADOS:

ENOME	DEPENDENTES		HABILIDADES		
	NOME	DATA_NASCIMENTO	TIPO	DATAS-TITULAÇÃO	
				ANO	CIDADE
Carlos	José	23/07/81	advogado	1984	Campinas
	Dora	05/04/50		1985	S. Paulo
			professor	1984	Campinas
Vilson	João	01/01/20	professor	1962	Curitiba
				1971	S. Paulo
				1975	S. Paulo
			sociólogo	1984	Campinas

Pode-se encontrar uma variação da representação de [FiT83] em [Set86], onde conjuntos de valores nem sempre são representados por atributos compostos. Veja a comparação entre (a) e (b) abaixo, considerando que a marca "*", na notação de Setzer, indica que o atributo é multivalorado:

A	B	C	
	B1	C1	C2
a	1 2 3	ca ba	2 1
b	5 6	ab bb	3 1

(a) Representação de [FiT83]

A	B1*	C*	
		C1	C2
a	1 2 3	ca ba	2 1
b	5 6	ab bb	3 1

(b) Representação de [Set86]

Uma outra maneira de se representar uma relação aninhada é usada por Jaeschke e Schek [JaS82] e Ozsoyoglu e Yuan [OzY87], por exemplo. Sob esse enfoque os atributos multivalorados não precisam mais ser componentes de atributos compostos. A tabela abaixo representa a relação EMPREGADOS sob essa nova classe de relações aninhadas.

Considere que um atributo multivalorado X seja representado por X* e que as composições de atributos são indicadas por parênteses.

ENOME	(NOME DATA_NASCIMENTO)*	(TIPO (ANO CIDADE)*)*
Carlos	José 23/07/81 Dora 05/04/50	advogado 1984 Campinas 1985 S.Paulo professor 1984 Campinas
Vilson	João 01/01/20	professor 1962 Curitiba 1974 S.Paulo 1975 S.Paulo sociólogo 1984 Campinas

O MRNN tem uma melhor representação semântica do mundo real do que o MRN. Além disso, ele simplifica a estrutura relacional reduzindo, em muitos casos, o número de relações. A simplificação do MRNN também se reflete nas consultas relacionais. Em particular, algumas consultas que tipicamente requereriam junções no MRN podem ser expressas por uma simples seleção no modelo relacional em NF^2, eliminando a necessidade de especificação de caminhos pelo usuário.

Se a relação EMPREGADOS fosse restrita à 1NF, poderia ser separada nas duas relações abaixo: DEPENDENTES e HABILIDADES, onde já se pode constatar as vantagens ditas acima.

DEPENDENTES:

ENOME	NOME	DATA_NASCIMENTO
Carlos	José	23/07/81
Carlos	Dora	05/04/50
Vilson	João	01/01/20

HABILIDADES:

ENOME	TIPO	ANO	CIDADE
Carlos	advogado	1984	Campinas
Carlos	advogado	1985	S.Paulo
Carlos	professor	1984	Campinas
Vilson	professor	1962	Curitiba
Vilson	professor	1971	S.Paulo
Vilson	professor	1975	S.Paulo
Vilson	sociólogo	1984	Campinas

Como ressaltado em [Set86], mesmo na época em que tudo girava em torno das idéias de Codd, alguns gerenciadores de banco de dados, como o ADABAS, implementaram o MRNN, pelo menos quanto à estrutura de dados. Infelizmente, o modelo implementado no ADABAS foi muito restrito, sem uma generalidade desejável e sem uma linguagem que explorasse as potencialidades da multivaloração.

Apesar das raras e limitadas implementações do MRNN que surgiram, pode-se dizer que a idéia do MRNN permaneceu adormecida até **Makinouchi** *reconhecer oficialmente em* [Mak77] que a restrição imposta pela 1NF dificulta muitas aplicações. **Makinouchi** mostrou que uma relação na 3.ª forma normal, pela sua própria definição, podia ser não-normalizada (em não-1NF) [Set86].

Desde então, o MRNN veio obtendo um crescente número de adeptos. Os trabalhos de pesquisas iniciais foram exclusivamente teóricos, de modo a se fundamentar matematicamente o MRNN. Trabalhos sobre a parte física do modelo começaram a surgir em meados de 1984,

mas só por volta de 1987 alguns artigos começaram a fazer menções sobre alguns protótipos em andamento.

O restante deste capítulo faz uma abordagem mais generalizada do MRNN. A *seção 1.2* estuda a forte relação existente entre o MRNN e o modelo orientado a objetos. A *seção 1.3* aborda visões não-normalizadas, para uma melhor interpretação dos dados armazenados. A *seção 1.4* enfoca as maneiras de se implementar o MRNN.

No *capítulo 2* são analisadas as formas normais no MRNN. A princípio isso parece ser um contra-senso, mas ver-se-á que a normalização traz ao modelo os mesmos benefícios que traz ao MRN, além de melhorar a representação semântica do MRNN.

No *capítulo 3* será feita uma abordagem comparativa entre algumas álgebras relacionais aninhadas existentes, apresentando a álgebra selecionada (de [FiT83]), juntamente com a definição dos operadores relacionais estendidos.

No *capítulo 4* são propostas as estruturas de armazenamento de informações para dar suporte ao MRNN.

No *capítulo 5* são apresentados algoritmos para a execução das operações relacionais aninhadas baseados na álgebra selecionada (*capítulo 3*) e nas estruturas propostas no *capítulo 4*. No *apêndice A* são apresentados alguns destes algoritmos com maior nível de detalhes.

O *capítulo 6* mostra as estruturas de dados usadas para a implementação das estruturas lógicas do *capítulo 4*, bem como os algoritmos de atualização dos dados sobre tais estruturas. Um complemento deste capítulo encontra-se no *Apêndice B*, onde o algoritmo de armazenamento dos dados é apresentado mais detalhadamente.

No *capítulo 7* faz-se um levantamento global do MRNN, além de se apresentar diretrizes para trabalhos futuros.

1.2. MRNN E MODELO ORIENTADO A OBJETOS

A questão do que vem a ser um *BD Orientado a Objetos* (*O.O.*), assim como a própria definição de *Orientação a Objetos*, é ainda objeto

de discussão (ver [Dit86]).

Takahashi et al. [Tad90], resumidamente, considera as seguintes características como requisitos para um BD ser *O.O.*:

(i) suporte a objetos complexos (no caso do MRN, uma série de tuplas logicamente ligadas juntas [LoP83]);

(ii) encapsulamento de comportamento e de dados, acessíveis externamente a partir de interfaces padronizadas.

(iii) conceito de tipo ou classe como mecanismo de agrupamento.

Ainda não há um modelo de dados *O.O.* aceito globalmente, apenas propostas. Geralmente, o modelo Entidade-Relacionamento se encaixa como base para um modelo *O.O.*. Conceitos como *generalização* e *agregação* (átômico) não têm sua essência alterada. Se o conceito de *agregação molecular* for associado ao modelo Entidade-Relacionamento e, sobre ele, forem aplicadas operações adequadas, pode ser considerado um modelo *O.O. estrutural* [Dit86].

Surge, então, uma questão: será que se deve obter um BD *O.O.* estendendo-se sistemas convencionais já existentes, como os relacionais, ou deve-se construí-lo a partir do "zero"?

Segundo [Dit86], as duas opções apresentam vantagens. Por um lado a aproximação relacional contribui com sua simplicidade e consistência. Além do mais, suas estruturas de dados planas serão ainda necessárias como componentes de *objetos complexos*. Por outro lado, construindo-se um novo sistema, pode-se explorar melhor certas peculiaridades da *Orientação a Objetos*, como, por exemplo, o agrupamento (*clustering*) dos objetos a nível físico.

O mapeamento de um BD *O.O.* para um BD relacional aninhado, embora não seja inteiramente trivial, é possível desde que o BD aninhado obedeça a certas propriedades. O paradigma do relacionamento aninhado modela conjuntos que são fundamentais para os sistemas *O.O.*. Os problemas encontrados pelos projetistas do modelo *O.O.* são muito similares aos problemas que são enfrentados na representação do modelo relacional aninhado.

Dentre os sistemas considerados *O.O. estruturalmente*,

encontram-se aproximações baseadas em NF^2 [Dit86]. Como exemplo, pode-se citar o sistema AIM (*Advanced Information Management*) de Dadam et al. [Dad86], que abordaremos melhor mais adiante, na *seção 1.1*. É bom esclarecer que a *Orientação a Objetos estrutural* se refere à *facilidade* do modelo de dados para representar entidades com estruturas complexas (*objetos complexos*), juntamente com seus operadores (genéricos).

As transações sobre *objetos complexos* normalmente são mais longas e envolvem volumes maiores de dados do que aquelas encontradas nos BDs tradicionais. Ainda, a manipulação de restrições sobre os objetos são muito mais complexas do que para os registros, tendo em vista a dinamicidade do modelo.

Projetos como o AIM de **Dadam et al.**, que podem ser considerados *O.O.*, deixam claro que sua base é relacional - ainda que relacional não-normalizada.

O projeto AIM ou AIM-P (*AIM-Prototype*) é um trabalho desenvolvido pelo *Heidelberg Scientific Center* da *IBM*. O objetivo do AIM é o suporte eficiente a vários tipos de aplicações, tais como escritórios, manufaturamentos, CAD/CAM, entre outras. Para o projeto, **Dadam et al.** [Dad88] consideraram dois pré-requisitos ao SGBD: suporte a *Orientação a Dados* (relações) e uma visão *Orientada a Objetos*.

O modelo relacional não-normalizado (MRNN) só foi realmente selecionado como modelo de dados após muitas pesquisas, através das quais se certificaram que a base teórica desse modelo é tão forte como a do MRN. Além disso, o MRNN possui a capacidade de suportar tanto uma visão *Orientada para Dados* quanto uma visão *O.O.*.

Durante as pesquisas do sistema AIM, os pesquisadores concluíram que o modelo NF^2 "puro" não era suficiente para desenvolver certas aplicações. Na área de engenharia, por exemplo, as aplicações precisavam de uma representação mais adequada para vetores e matrizes. Para isso, incluiu-se no MRNN o conceito de listas (conjuntos ordenados), impondo-se uma ordenação às linhas e às colunas.

No desenvolvimento do AIM, o conceito de relações aninhadas (ordenadas ou não) foi predominante na definição de objetos. O modelo NF^2 estendido é capaz de integrar tabelas planas (MRN) e hierárquicas (MRNN), contendo *objetos complexos* de maneira homogênea.

Na *seção 1.4*, onde se trata das implementações de sistemas, voltaremos a considerar sistemas em NF^2 *O.O.*, inclusive o próprio AIM. Obviamente, tanto um modelo NF^2 como *O.O.* só são viáveis se forem suportados eficientemente a baixo nível da arquitetura do sistema.

1.3. VISÕES SOBRE BANCOS DE DADOS EM NF^2

Devido à especificação de visões num BD ser um dos principais mecanismos para a seleção de dados, abordar-se-á sucintamente o assunto nessa seção.

No MRN, uma visão é definida como uma consulta sobre as relações do BD ou até mesmo sobre outras visões pré-definidas. Geralmente, a relação resultante dessa consulta (*relação de visão*) está apenas na primeira forma normal, mesmo em BDs completamente normalizados. Ainda, não há nenhuma evidência de que visões não-normalizadas sejam incovenientes para uma representação relacional normalizada [Wie86]. Segundo **Wiederhold** [Wie86], essa *aceitação* de visões não-normalizadas já é, parcialmente, uma validação de que há estruturas mais adequadas do que tabelas normalizadas para se representar objetos do mundo real (conseqüentemente, *objetos complexos* na linha de *Orientação a Objetos - seção 1.2*).

A tendência ao MRNN torna-se mais intensa se considerarmos que a construção da *relação de visão*, nesse modelo, requer menos tuplas e um custo menor na associação de informações, visto que no MRN essas estão geralmente distribuídas em várias relações.

Como o MRNN é relativamente recente, é de se esperar que estudos em *visões* sobre o modelo sejam ainda escassos. No entanto, alguns trabalhos mostram direções nesse sentido.

Em [Wie86], **Wiederhold** reune os conceitos de *visão* em BDs relacionais e de *orientação a objetos*, de modo a unir também suas qualidades.

Durante seus estudos, **Wiederhold** reforça que o MRN apresenta falhas no tratamento de *objetos complexos* (ver *seção 1.2*), enfatizando que pretende tirar proveito das pesquisas sobre BDs em NF^2 em

andamento. O fato do autor trabalhar com o MRN e não com o MRNN, nesse estágio inicial, está associado à complexidade das estruturas de armazenamento desse último modelo.

Em [Per87], Pernul propõe um modelo de dados baseado no MRNN, cujo princípio básico é considerar visões num sistema multi-usuário. Para o autor, um esquema inicial de BD pode se decomposto *sem perdas* de acordo com as visões dos usuários. Para tanto, são empregados conceitos de *fragmentação horizontal*, *vertical* e *fragmentação horizontal derivada*.

A estrutura de decomposição é armazenada num dicionário de dados, o qual é modelado pelo uso de estruturas de dados relacionais em NF^2. Esse dicionário contém dois subesquemas: o de Aplicação, para as informações da estrutura de decomposição, e o subesquema de Dados, para as informações sobre as visões e sobre os acessos diretos para os usuários.

Com esse modelo, Pernul objetiva maior eficiência do BD em termos de recuperações e armazenamento dos dados; é possível processar consultas dos usuários criando processos concorrentes para trabalhar sobre as sub-relações (fragmentos) do esquema inicial. Além disso, a segurança propiciada pelo uso de visões é mantida. Pernul propõe expandir seu modelo em várias direções, como CAD e ambientes multiprocessados.

1.4. PROJETOS FÍSICOS DO MRNN

Apesar da variedade de técnicas sofisticadas de projeto físico de BD, na prática as facilidades disponíveis são muito limitadas. Uma das razões é que, geralmente, os modelos internos diferem da visão lógica, impedindo a transparência do sistema ao usuário. Desse modo, o principal objetivo ao se projetar fisicamente um BD é encontrar uma representação interna ótima do esquema.

Em termos de desempenho, o projeto físico ainda luta contra um feixe de parâmetros inter-relacionados oferecidos pelo Sistema de Gerenciamento do BD (SGBD) e pelo Sistema Gerenciador de Arquivos

(SGA). O número de acessos ao disco, E/S (entrada/saída), é geralmente usado como o mais importante indicador de desempenho de um SGBD.

Um dos principais problemas encontrados em várias técnicas aplicáveis a SGBDs relacionais é a seleção do caminho de acesso a determinados dados, isto é, qual o conjunto de caminhos de acesso que deve ser gerado para se obter um desempenho ótimo das consultas e atualizações.

Algumas soluções não triviais vêm sendo estudadas e já são oferecidas por alguns sistemas. É o caso da *"clusterização"* (*clustering*) ou agrupamento dos dados, onde a regra geral é que os dados que são requeridos juntos sejam armazenados juntos, numa mesma página ou numa mesma vizinhança. Como exemplo temos os *"clusters"* no ORACLE, permitindo armazenar tuplas de relações diferentes numa mesma página, considerando valores idênticos em atributos comuns. O principal ganho com essa técnica é o suporte eficiente à junção.

Outros exemplos, mencionados em [SPS87], são a técnica de *caminhos de acesso generalizado* de **Härder** [Här78] e a de *índices de junção* de **Lehmann e Carey** [LeC86], com o objetivo de propiciar, também, maior suporte às junções.

No MRNN alguns problemas tornam-se mais intensos, como é o caso da *"clusterização"* dos dados, que deverá, agora, não considerar apenas tuplas de atributos atômicos, mas também tuplas com várias subtuplas para uma dada relação. Além disso, os *caminhos de acesso* devem considerar elementos dos atributos aninhados e multivalorados, precisando de uma técnica sofisticada de indexação. A implementação do MRNN também deve considerar o problema da *otimização de consultas*, sendo que, agora, as operações de junção são mais complexas (possivelmente entre níveis diferentes de diferentes tabelas).

Uma solução possível seria *mapear* o MRNN para uma implementação de BD relacional já existente. Por um lado o usuário passaria a trabalhar com uma hierarquia de valores (composições dos atributos) que aproximaria mais seu sistema do mundo real. Por outro lado, as vantagens que poderiam ser tiradas da implementação direta de relações não-normalizadas seriam perdidas. Cria-se, na verdade, um *MRNN falsificado*, acarretando *overhead* desnecessário na tradução de

consultas sobre uma visão não-normalizada para uma normalizada.

Deshpande e Gucht [DeG88] apontam alguns motivos para que não se utilize outras implementações de BDs relacionais existentes para o MRNN:

(1) os atributos, geralmente, não são atômicos, o que dificulta o mapeamento para o MRN;

(2) otimizações de consultas que são usadas sobre o MRNN não podem ser usadas quando a estrutura de base é relacional normalizada;

(3) seleções são feitas freqüentemente sobre componentes profundamente aninhados dentro das tuplas, acarretando *overhead* no processamento;

Em [DeG88], os autores lembram, ainda, que não se pode pensar no modelo hierárquico nem no de redes como estrutura de base, pois não foram desenvolvidos com linguagens de alto nível não-procedurais em mente.

Devido a motivos do gênero acima, excluímos desse texto maiores discussões sobre esses tipos de mapeamentos e concentramos nossa atenção na implementação direta do MRNN.

1.4.1. Modelos Físicos do MRNN

Alguns SGBDs hierárquicos podem ser considerados como precursores dos SGBDs em NF^2, pelo menos quanto às estruturas de dados. Convém lembrar que a hierarquia das composições dos atributos permite que se faça tal consideração.

O SGBD ADABAS (*Adaptable Data Base System*), por exemplo, lineariza os segmentos de registros hierárquicos em um arquivo e em outro coloca as descrições das relações (Dicionário/Diretório) [Oll71]. O ADABAS, em seus registros, assinala os campos multivalorados, embora não represente os compostos [Set86].

No SGBD OASIS, citado por [Wie83], uma instância de uma tupla e seus descendentes são colocados num registro singular compacto

de tamanho variável.

No SGBD IMS [McG77], uma relação e todas suas relações internas aparecem num único arquivo. Cada entrada de nível alto num arquivo contém os atributos atômicos de uma tupla na relação externa.

Recentemente, alguns novos sistemas suportando relações aninhadas têm sido projetados e implementados de modo a suprir as necessidades de áreas de aplicações não-convencionais.

Em geral, de modo a sanar problemas comuns ao MRNN, as aproximações tendem a agrupar os dados fisicamente tanto quanto possível, a fim de minimizar o número de E/S e de evitar que o acesso às informações parciais torne-se muito custoso. Com a dinâmica do BD, entretanto, esse agrupamento é facilmente comprometido.

Outra necessidade eminente é o tratamento de novas técnicas de caminhos de acesso e de indexação para informações gerais e específicas. A separação das informações estruturais e dos dados também é de consenso geral.

É claro que todas as implementações de relações em NF^2 visam estabelecer uma conexão eficiente entre uma tupla aninhada e suas subtuplas. Convém ressaltar que o uso de "*surrogates*", para valores de atributos em tuplas das relações mais internas, é o suficiente para tirar as ambigüidades das estruturas de tuplas de relações aninhadas. No entanto, há duas técnicas adicionais mencionadas por [Ha089]:

(1) índices de junção binário: para estabelecer conexão entre dois arquivos, representando, respectivamente, uma relação externa R_i e uma interna R_j. Para cada par de arquivos (R_i, R_j), há um índice binário, ou seja, uma relação com dois atributos *surrogates*. O número de entradas no índice binário é igual ao número de tuplas na relação R_j. Embora essa técnica tenha se mostrado muito útil em otimização de junções, os autores ressaltam que a eficência diminui para alguns operadores NF^2 (*NEST*, *UNNEST*, *PROJECT* e *SELECT*).

(2) índices de junção hierárquico: consiste em usar os *surrogates* das relações externas/internas da relação em NF^2 como um todo, obedecendo a ordem de composição e hierarquia. Segundo [Ha089], essa técnica é muito boa para se descrever a relação. Mesmo que se use índices de junção binários, a recuperação das subtuplas podem

necessitar da manutenção de índices de junção binários adicionais.

De modo geral, segundo Setzer [Set86], há duas formas básicas de se implementar um modelo interno de BDs não-normalizado:

(1) colocar, no lugar de cada coluna multivalorada, um apontador para os valores representados em arquivos à parte.

(2) representar valores multivalorados dentro dos registros que contêm os dados das linhas, por inspiração do SGBD ADABAS.

Como exemplo da opção (1) acima, tem-se o SGBD POSTGRES, que possui atributos que são procedimentos. Para suportar relações aninhadas, definiu-se um novo tipo de dados chamado POSTQUEL. Um campo do tipo POSTQUEL contém uma seqüência de comandos para recuperar dados de outras relações. Todas as relações são armazenadas como pilhas ("*heaps*") dentro de uma coleção opcional de índices secundários [Jhi88] [Ha088]. Esse SGBD relacional estendido, por outro lado, une-se também à linha de Orientação a Objetos (ver *seção 1.2*).

O modo de implementação do item (2) acima abrange a maioria das implementações, inclusive a adotada por Setzer . O autor utiliza um dicionário contendo as descrições das relações e dos atributos, não precisando de nenhum indicador especial para campos compostos ou multivalorados a nível de registros, ao contrário do seu SGBD inspirador: ADABAS. Seu modelo permite encaixar atributos multivalorados e compostos em qualquer número de níveis. Os registros, de tamanho variável, são linearizados utilizando-se encadeamentos entre os valores dos campos, segundo a valoração e composição do atributo. Sua aproximação suprime os dados *vazios* e compacta os dados eliminando zeros à esquerda (campos numéricos) e brancos à direita (campos alfanuméricos).

Sob outra ótica, diferente da de [Set86], Hafez e Ozsoyoglu [Ha088], classificam, de modo geral, os modelos de armazenamento para relações aninhadas em quatro categorias básicas, antes só usadas para *objetos complexos* sem que se levasse em conta a álgebra relacional:

(1) Modelo de Armazenamento Decomposto: utiliza armazenamento invertido. Cada atributo atômico de uma relação juntamente com um *surrogate* para identificar um registro formam uma relação binária; cada uma dessa relações é armazenada num arquivo separado. Esse modelo proporciona eficiência nas operações de projeção e seleção (para poucos atributos), além de agilizar as junções, se o modelo for acompanhado de índices de junção.

(2) Modelo de Armazenamento Normalizado: decompõe uma relação aninhada de tal modo que os atributos atômicos de cada tupla de uma relação formam um registro de um arquivo. As relações mais internas em um dado nível são relacionadas entre si por *surrogates*. Podem ser usados *índices de junção* para recuperação de uma relação interna. Esse modelo só é eficiente para operações sobre nós no mesmo nível e, por outro lado, é muito ruim para o caso de haver atributos compostos com muitos subatributos.

(3) Modelo de Armazenamento Plano ou Direto: uma relação aninhada é armazenada diretamente em um só arquivo. Cada registro representa uma tupla da relação externa. Os registros de um arquivo são agrupados sobre os atributos atômicos da relação externa (primeiro nível de aninhamento). O acesso aos outros valores internos das tuplas são feitos por busca seqüencial ou utilizando estruturas de dados adicionais, embora isso aumente os custos de acesso e manutenção. O mapeamento de uma tupla conceitual para um registro é perfeito. Além disso, as junções, usadas nos outros modelos, tornam-se desnecessárias. Esse método proporciona um eficiente acesso às tuplas, mas por outro lado, a recuperação das informações é ineficiente quando as tuplas são muito grandes.

(4) Modelo de Armazenamento Parcialmente Decomposto: é um "casamento" dos modelos (1) e (2) acima. É um bom método quando se tem seleções e projeções muito freqüentes. Seu desempenho depende do particionamento vertical, sendo muito ruim para junções.

Hafez e Ozsoyoglu [Ha088] optam por uma generalização dos Modelos de Armazenamento Normalizado ((2)) e Direto ((3)), apresentando, assim, um novo modelo: o P-NSM (Modelo de Armazenamento

Normalizado Parcial). No P-NSM, as relações em NF^2 são particionadas verticalmente, de forma que as relações internas freqüentemente acessadas juntas sejam armazenadas no mesmo arquivo. Cada arquivo contém os atributos atômicos de uma relação interna e de alguns de seus descendentes.

A justificativa para a introdução de um novo modelo é baseada no tempo (número de E/S) de resposta às consultas. Segundo [Ha088], de um modo geral, o modelo de armazenamento de uma relação aninhada pode ser classificado pelo suporte a três tipos de consultas:

(a) consultas que manipulam tuplas da relação;
(b) consultas que manipulam subtuplas;
(c) consultas que manipulam componentes individuais específicos das tuplas/subtuplas da relação e seus atributos atômicos nos diferentes níveis de aninhamento.

Se num BD o tipo de consulta (a) é dominante, então o modelo de armazenamento direto é a melhor escolha. Se o tipo (b) predomina, então o modelo de armazenamento normalizado é o mais adequado. Se o tipo dominante de consulta é o (c), optar por um dos quatro modelos classificados anteriormente fica muito difícil, visto que esse tipo é um caso genérico para as consultas relacionais aninhadas. O P-NSM providencia bom suporte a sistemas que se enquadram nesse último caso, devido à consideração de informações de *workload* do sistema, obtendo um custo mínimo de consultas.

Como já foi visto no início desta seção, algumas propostas para o MRNN consistem na linearização dos registros com indicadores especiais. Essa idéia, no entanto, também surgiu na área de *Orientação a Objetos* (*O.O.*) por Lorie e Plouffe [LoP83]. Segundo os autores, o armazenamento de um *objeto complexo* se dá pelo armazenamento das tuplas como parte de tabelas planas com atributos adicionais não visíveis para os usuários. Tais atributos contêm os ponteiros usados para o encadeamento entre as tuplas do objeto [Dad86].

Deshpande e Gucht [DeP88] e Dadam et al. [Dad86] [Dad88] também aderiram a essa linha de pesquisa estendendo o MRNN , para maior eficiência (ver também *seção 1.2*). Ambos trabalhos enquadram-se na categoria de Modelos de Armazenamento Direto de [Ha089], visto

anteriormente, fazendo uso de estruturas adicionais. Deshpande e Gucht utilizam esquemas de *hashing elaborados*, generalizando os *índices de junção* de [Val87]. Dadam et al., por outro lado, utilizam *vetores de ponteiros*. Como ilustrativo, vejamos rapidamente alguns pontos básicos dessas implementações.

Para Dadam et al. [Dad86] [Dad88], cada tupla da relação externa (*objeto complexo*) é armazenada em um registro. Para cada tupla, há uma entrada num diretório central chamado mini-diretório (MD), que é usado para manipular as ações sobre cada tupla. Cada *objeto complexo* possui um identificador de tupla (TID), referente ao endereço do objeto relativamente ao segmento do BD. Além do MD central, cada objeto é armazenado sob forma de uma árvore, também um MD, que contém todas as suas informações estruturais, correspondentes à sua estrutura hierárquica. Os nós dessa árvore podem ser subtuplas de dados (valores atômicos) ou mesmo subtuplas de MD referentes ao "subobjetos", decorrentes da hierarquia. As subtuplas de dados armazenam todos os valores atômicos dos atributos de um respectivo nível, desde que sejam componentes de um atributo composto comum. Cada subtupla é equivalente a uma página.

Desse modo, pode-se navegar sobre os objetos, sem necessariamente se verificar todos os dados.

Os *subobjetos* são identificados por mini-TIDs, ou seja, endereços relativos à raiz do objeto complexo ao qual pertencem. Esse endereçamento por mini-TIDs apresenta vantagens muito interessantes:

(1) por serem menores que os TIDs, economizam espaço de armazenamento no MD e aceleram o processamento de *objetos complexos* (tuplas externas);

(2) para se mover um *objeto complexo* para outro lugar do BD, basta atualizar o seu endereço.

Segundo Dadam et al. [Dad86], poder-se-ia construir a árvore de cada objeto de três maneiras: usando subtuplas MD somente para *objetos complexos*, somente para subtabelas ou para ambos. No sistema AIM-P, os autores utilizam a segunda alternativa, considerando critérios como espaço de armazenamento, tempo de acesso, etc..

A primeira versão do AIM-P tornou-se operacional em 1986 [Dad88]. Desde então, o sistema tem sido instalado em vários lugares

para pesquisa e avaliação. Em 1988, a implementação ainda não possuia uma eficiente otimização de consulta, nem suporte a visões, mas já suportava, satisfatoriamente, versões temporais, assim como tipos de dados e funções definidos pelo usuário, ambientes estação de trabalho-servidor e suporte a transações básicas num ambiente mono-usuário [Dad88].

Deshpande e Gucht [DeG88] propoem uma implementação para o MRNN, denominada ANDA (*Acronymn for a Nested Database Architecture*). Os autores consideram uma estrutura de árvores singular (VALTREE), contendo todos os atributos atômicos do BD. Através de um valor atômico dado, pode-se localizar todas as tuplas externas e internas tendo aquele valor. Para que isso seja possível, foi proposto uma identificação hierárquica de tuplas para melhorar o tempo de acesso. Cada relação do BD possui um identificador de estrutura diferente, como por exemplo: um identificador p para uma relação de nome *PROJETOS*. Cada valor atômico possui um ou mais identificadores, caso pertença a várias subtuplas de uma relação e/ou a várias relações. A forma geral desse identificador é dada por:

$$p \begin{matrix} (a/\ldots/z) \ldots (a/\ldots/z) \leftarrow (I) \\ (1/\ldots/9) \ldots (1/\ldots/9) \leftarrow (II) \end{matrix}$$

(III)

onde (I) se refere à coluna da tupla, (II) ao número seqüencial da tupla e (III) é um par indicando o nível de aninhamento usado quando a coluna é composta; se for multivalorada, para seu número de tupla, não haverá indicador de coluna correspondente.

Para dar suporte ao projeto, há uma estrutura de lista de registros (RECLIST), que mapeia um TID hierárquico para seu endereço físico real, utilizando um esquema de *hashing* linear. [DeG88] observam que quanto mais uma tupla for quebrada em fragmentos menores (*granularidade*), mais fácil é a obtenção direta de um valor, além de se evitar o *overflow* nas inserções. No entanto, isso requer muitos acessos a disco para reconstruir uma tupla.

Através das estruturas adicionais acima, os autores visam melhorar o desempenho tanto das operações direcionadas por valor (aninhamento, seleção e junção), pelo uso da VALTREE, como daquelas

orientadas por estruturas (projeção e desaninhamento), usando-se a RECLIST.

Nem sempre há interesse em se implementar o MRNN desde o núcleo do BD até as visões lógicas do usuário. Nessa linha, Scholl et al. [SPS87] apresentam um projeto pioneiro, consistindo de uma família de sistemas de BD com base num núcleo comum em NF^2, DASDBS (*Darmstadt Database System*), que constitui o subsistema de armazenamento do SGBD. O sistema suporta vários *front-ends*, com estruturas distintas, como por exemplo, BD geológico, de escritórios e relações planas em 4NF. Desse modo, várias técnicas aplicadas a projetos físicos de BD podem ser representadas numa única, utilizando um subconjunto do MRNN para nível de registros (modelo interno).

Na interface do núcleo, as estruturas de dados são relações aninhadas e as tuplas das relações são mapeadas para *registros complexos* (RCs). Esses RCs são endereçados hierarquicamente por uma combinação de *surrogates* e TIDs, permitindo eficiente acesso direto tanto às tuplas quanto às subtuplas. Cada RC é agrupado num conjunto minimal de páginas, sendo que todos os RCs de uma mesma relação são armazenados em páginas adjacentes ("*clustering* hierárquico").

Para um *front-end* relacional em 4NF, os esquemas lógicos (MRN) são mapeados para o modelo interno (MRNN) utilizando-se um subconjunto da álgebra relacional aninhada, estendida de mais dois operadores. Um deles é o seletor de endereços (ψ), que recupera por acesso direto o conjunto de RCs endereçados por um certo conjunto; é usado para construir e executar os caminhos de acesso a partir do núcleo. O outro operador, outer-join, introduz valores nulos em tuplas com valores pendentes. Foi, ainda, introduzida uma operação de redução especial para eliminar os valores nulos, quando desnecessários.

As transformações algébricas, exceto a junção, utilizam somente seleções e projeções, passando a seguir por um processo de otimização, que já reflete o projeto físico do BD. Deve-se ressaltar que as otimizações de consultas utilizadas normalmente são feitas a um nível muito longe do das páginas, impedindo um aproveitamento eficiente da "clusterização" das informações [SPS88].

O sistema proposto suporta todas as operações importantes de

junção, devido ao modo pelo qual são gerados os caminhos de acesso. Estruturas adicionais, como *índices de junção*, proposto em [Här78] e outras menos usuais como a "denormalização" (*denormalization*) podem ser descritas formalmente por RCs do DASDBS.

A denormalização foi proposta por Schkolnick e Sorenson [ScS80] [ScS81], citados em [SPS87], como um meio de materializar junções, ou seja, armazenar internamente os resultados das junções mais freqüentes. Assim, tais operações tornam-se menos onerosas. [SPS87] optaram por armazenar as junções juntamente com as relações originais, ao invés de simplesmente armazenar o resultado da junção no lugar das relações lógicas, como em [ScS80] e [ScS81]. Para isso foram introduzidos valores nulos nas tuplas pendentes. Além disso, como o resultado de uma junção é uma relação interna e o esquema lógico não é alterado, o usuário e os programas aplicativos não são afetados.

Se todas junções na consulta são materializadas, uma consulta otimizada algebricamente pode ser executada diretamente pelo núcleo. Isso, entretanto, nem sempre é o melhor caso. Normalmente, algumas junções não são armazenadas, sendo calculadas pela camada superior do núcleo por repetidas chamadas, seguindo uma estratégia determinada cuidadosamente. Nesse caso, os endereços de referência das relações participantes da junção é que são materializados. Adicionalmente, tem-se que gerar e selecionar caminhos de acesso (planos de acesso) ou índices de junção.

No DASDBS, um módulo, chamado MPP (*Multi Pass Query Processor*), encontra a estratégia de caminho de acesso, enquanto que outro módulo, SPP (*Single Pass Processor*), executa a estratégia.

Segundo [SPS87], outros *front-ends*, incluindo os que suportam noções de *subobjetos distribuídos* e *objetos complexos*, encontrarão, no DASDBS, um sistema de armazenamento adequado. Entretanto, tais modelos não poderão ser mapeados para o núcleo em NF^2 simplesmente por substituições algébricas, pois esses modelos incluem fatores como estruturas recursivas e herança de atributos, que introduzem complexidade adicional ao problema do projeto físico.

1.4.2. Análise de Desempenho

Infelizmente, ainda não foi feita uma análise de desempenho satisfatória para demonstrar a boa *performance* do MRNN.

Alguns trabalhos chegam a trazer análises de certas características aplicadas ao MRNN, como eliminação de junções e transações mistas envolvendo denormalizações [SPS87], mas nada específico a este modelo.

Há muito poucos protótipos do MRNN implementados, mas menor ainda é a quantidade de trabalhos que trazem suas respectivas análises de desempenho. Entre os sistemas implementados até 1988, pelo menos, encontram-se parte do ANDA de Deshpande e Gucht [DeG87], parte do AIM-P de Dadam et al. [Dad86] [Dad88] e o núcleo DASDBS de Scholl et al. [SPS87], juntamente com seu otimizador algébrico. Pelo menos a nível de nossos conhecimentos, o único trabalho a trazer algo sobre o assunto é [SPS87]. O artigo, entretanto, não oferece uma análise de desempenho real, porque o protótipo do sistema ainda era recente quando o artigo foi escrito. Os autores oferecem um demonstrativo de experimentos práticos, que podem ser tomados como fortes indicativos do bom desempenho do MRNN e que serão resumidos a seguir.

Sobre o *SQL/DS*, os autores simularam o efeito de uma materialização de junção relacional aninhada. Isso foi feito através do *carregamento* adequado de uma seqüência de tuplas das relações envolvidas e da criação de índices. Adicionalmente, foi preciso considerar "manualmente" criações apropriadas de caminhos de acesso e otimizações. Foram gerados três BD *SQL/DS* para um esquema relacional convencional com três *layouts* físicos, distinguidos pelas estratégias de "clusterização". Desses três BDs, dois merecem mais nossa atenção: um BD em 4NF e um BD em NF^2, obtido através da simulação de suas estruturas. Em termos de E/S e de chamadas ao SGBD, o desempenho do BD em NF^2 mostrou-se superior ao 4NF; no pior caso, foi praticamente igual.

2. NORMALIZAÇÃO NO MODELO RELACIONAL ANINHADO

2.1. INTRODUÇÃO

O conceito de *normalização* foi introduzido por **Codd** juntamente com o modelo relacional **[Cod70]**, onde o autor estabeleceu o que posteriormente chamou de *1ª forma normal* ou *1NF* (*1st normal form*). Em 1972, **Codd** apontou "anomalias" (propriedades indesejáveis) de inserção, atualização e eliminação de dados, que podem ocorrer nas relações que estão simplesmente em 1NF **[Cod72]**. Para tirar tais *anomalias*, introduziu, então, mais duas formas normais baseadas nos conceitos de dependências funcionais, que receberam o nome de *2ª forma normal* (*2NF*) e *3ª forma normal* (*3NF*). Essa última foi aperfeiçoada para se eliminar "certas misturas de informação" **[Set86]**, gerando uma nova forma normal: a *forma normal de Boyce-Codd* ou *BCNF* **[Dat84]**. Em **[Fag77]** foi introduzida a *4ª forma normal* ou *4NF* baseada no conceito de dependências multivaloradas (MVDs), também introduzido pelo autor. Tais MVDs são generalizações das FDs, objetivando a extensão da semântica do BD. Em 1979, **Fagin** apresentou um caso muito particular de formas normais: a *5ª forma normal* ou *5NF*. A introdução dessa última forma normal se deu para resolver problemas de *anomalias* de atualização em relações que, se decompostas em duas (processo usado nas outras formas normais), sofreriam perda de informação. Segundo a *5NF*, tais relações podem ser decompostas em três relações, sem perda de informações.

Todas essas formas normais acima têm como principal objetivo a eliminação de redundâncias e *anomalias*, que estão associadas às atualizações de informações armazenadas redundantemente **[Wie86]**.

Com esse mesmo objetivo, o conceito de *normalização* foi introduzido no MRNN. Adicionalmente, uma forma normal para relações

aninhadas visa não somente agrupar atributos relacionados em conjuntos, mas, principalmente, obter uma estrutura de aninhamento que seja uma boa representação do conjunto de conexões semânticas existentes entre os atributos e o mundo real [RoK8].

Algumas propostas de normalização no MRNN que vêm sendo apresentadas podem ser classificadas basicamente de duas maneiras: aproximações que utilizam novas formas normais, específicas para o MRNN, e aproximações que estendem as conhecidas formas normais do MRN para o MRNN. O primeiro tipo de aproximação será visto na *seção 2.3* e o segundo tipo, na *seção 2.4*.

Toda a teoria de normalização do MRNN está fundamentada em conceitos e definições teóricas, que serão apresentadas na próxima seção para melhor compreensão do texto que segue.

2.2. DEFINIÇÕES E CONCEITOS AUXILIARES

Para representar os valores componentes A_1, A_2, ..., A_n de uma tupla t de uma relação usar-se-á a notação:

$$t[A_1 A_2 \ldots A_n].$$

Definição 1: Dada uma relação r com esquema R em *1NF*, considere X e Y são conjuntos de atributos de R. Dizemos que $X \rightarrow Y$ é uma dependência funcional ou FD (*Functional Dependency*), *i.e.*, Y é funcionalmente dependente de X *se, e somente se*, para quaisquer duas tuplas t e u de r se $t[X] = u[X]$ então $t[Y] = u[Y]$.

Definição 2: *Seja* r uma relação aninhada de esquema R, cujo conjunto de atributos é denotado por E_R. Sejam $X, Y \subseteq E_R$. A dependência funcional estendida $X \rightarrow Y$ é válida em r *se, e somente se*,

$$(\forall\, t, u \in r)\ \ t[X] = u[X] \implies t[Y] = u[Y]$$

Se X ou Y são atributos multivalorados, então a igualdade acima se refere à igualdade de conjuntos.

Definição 3 [RoK87]: Seja r uma relação qualquer sobre um esquema R, com X, Y e Z subconjuntos de R. Seja $Z = R - XY$. A *relação r satisfaz a* dependência multivalorada ou MVD (*Multivalued Dependency*) $X \twoheadrightarrow Y$ se para todo par de tuplas $t1$ e $t2$ de r, com $t1[X] = t2[X]$, então existe uma tupla $t3$ em r tal que $t3[X] = t1[X]$, $t3[Y] = t1[Y]$ e $t3[Z] = t2[Z]$.

Definição 4 [RoK87]: Seja r uma relação qualquer sobre um esquema R, com X, Y e Z subconjuntos de R. Seja $X \subseteq Z$ e $Y \subseteq Z$. A relação r satisfaz a dependência multivalorada embutida ou EMVD (*Embedded MVD*) $X \twoheadrightarrow Y|Z-XY$ quando a MVD $X \twoheadrightarrow Y$ é mantida na projeção de r sobre Z.

Uma MVD pode se manter numa projeção de uma relação, embora não se mantenha na própria relação. Essas EMVDs podem aparecer quando se decompõe um esquema de relação em esquemas menores.

Definição 5 [RoK87]: Seja r uma relação qualquer sobre um esquema R e seja $\mathcal{R} = (R_1, R_2, \ldots, R_n)$ um conjunto de esquemas que são projeções do esquema R. A relação r satisfaz a dependência de junção $*(R_1, R_2, \ldots, R_n)$ se a decomposição de r sobre $R_1, R_2, \ldots, R_n$ é sem perdas, isto é,

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \ldots \bowtie \pi_{R_n}(r).$$

Definição 6 [RKS88]: Seja uma relação aninhada r com um esquema R, cujos atributos são denotados por E_R. Sejam $A_1, A_2, \ldots, A_n$ os atributos simples de R e $X_1, X_2, \ldots, X_l$ os atributos compostos. Dizemos que R está na Forma Particionada ou em PNF (*Partioned Normal Form*) *se, e somente se,* os ítens *(i* e *(ii* são satisfeitos:

(i *se* $X_1, X_2, \ldots, X_l = \emptyset$ *então* R está em PNF;

(ii $A_1 A_2 \ldots A_n \rightarrow E_R$;

(iii $(\forall\ t \in r)$ e $(\forall\ X_i,\ 1 \leq i \leq l)$, r_{ti} está em PNF, onde r_{ti} é uma relação de esquema X_i e de valor $t[X_i]$.e valor $t[X_i]$.

Definição 7 [RoK87]: Uma árvore de esquema T é uma árvore cujos vértices são rotulados por conjuntos disjuntos de atributos não-aninhados, onde as arestas (ligações) da árvore representam MVDs entre os atributos nos seus vértices.

Exemplo [RoK87]:

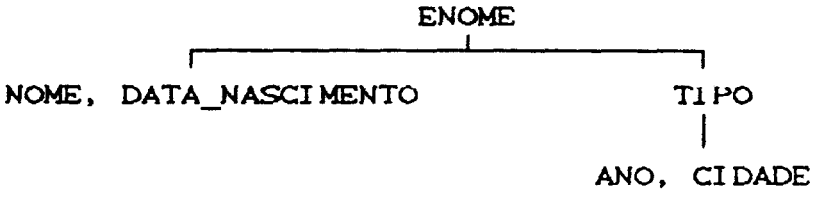
Considerando a relação EMPREGADOS da *seção 1.1*, de esquema:

EMPREGADOS = (ENOME, DEPENDENTES, HABILIDADES)
DEPENDENTES = (NOME, DATA_NASCIMENTO)
HABILIDADES = (TIPO, DATAS_TITULAÇÃO)
DATAS_TITULAÇÃO = (ANO, CIDADE).

Suponha válidas as seguintes dependências:

ENOME $\twoheadrightarrow$ NOME, DATA_NASCIMENTO
ENOME $\twoheadrightarrow$ TIPO, ANO, CIDADE
ENOME, TIPO $\twoheadrightarrow$ ANO, CIDADE.

A árvore de esquema implicada pelas MVDs acima é dada por:



Para as próximas definições, considere U um esquema universal, T uma árvore de esquema sobre U em relação a um conjunto de MVDs e $e=(u,q)$ uma aresta de T. Considere, ainda, as seguintes notações:

$F(q)$: o pai de q, que é u;
$A(q)$: a união dos rótulos de todos os antecessores de q, incluindo u;
$D(q)$: a união dos rótulos de todos os descendentes de q, incluindo q;
$S(t)$: a união dos rótulos de todos os nós de T;
$MVD(T)$: o conjunto de MVDs representadas pelas arestas de T;

M: o conjunto de MVDs válidas sobre U;

$LHS(M)$: o conjunto dos lados esquerdos das MVDs do conjunto M.

Definição 8 [RoK87]: Sejam $u_1, u_2, \ldots, u_n$ todos os nós folhas de T. Então o conjunto de caminhos de T é dado por:

$$P(T) = (A(u_1), \ldots, A(u_n)).$$

Definição 9 [RoK87]: Seja $C \subset U$. Uma base de dependência para X, denotada por $DEP_M(X)$, é uma partição de $U - X$ em conjuntos de atributos $Y_1, Y_2, \ldots, Y_n$ tal que:

se $Z \subseteq U - X$ *então* $X \twoheadrightarrow Z \iff Z$ *é a união de alguns* Y_i*'s.*

Definição 10 [RoK87]: Para um conjunto M de MVDs, M^+ denota o fecho de M, *i.e.*, o conjunto de todas MVDs implicadas por M, utilizando-se as conhecidas regras de inferência para MVDs.

Ainda, dados dois conjuntos de MVDs M e N, dizemos que M é uma cobertura de N se $M^+ = N^+$.

Definição 11 [RoK87]: Dado um conjunto M de MVDs sobre U, uma MVD $X \twoheadrightarrow W$ em M^+ é dita ser

(i) trivial se $XW = U$, $W = \emptyset$ ou $W \subseteq X$;

(ii) redutível à esquerda se $\exists\ X' \subset X$ *tal que* $X' \twoheadrightarrow W$ está em M^+;

(iii) redutível à direita se $\exists\ W' \subset W$ *tal que* $X \twoheadrightarrow W'$ está em M^+;

(iv) transferível se $\exists\ X' \subset X$ *tal que* $X' \twoheadrightarrow W(X-X')$ está em M^+.

Definição 12 [RoK87]: Um conjunto M de MVDs é dito ser uma cobertura minima se:

(i) toda MVD em M é reduzida (não satisfaz os ítens (i)-(iv) da *definição 11*);

(iii) nenhum subconjunto próprio de M é uma cobertura de M, *i.e.*, $M \subseteq M^-$.

Definição 13 [RoK87]: Denotemos por M^- o conjunto de todas MVDs reduzidas (não satisfazem os itens (i)-(iii) da *definição 11*) implicadas por M. Suponha dados um conjunto N de MVDs e $M \subseteq M'$ uma cobertura minima de N. Então os elementos de $LHS(M^-)$ são chamados chaves de N. Os elementos em $LHS(M)$ são chamados chaves essenciais e os elementos em $LHS(M^-) - LHS(M)$ são chamados chaves não essenciais.

Definição 14 [OzY87]: Seja D um conjunto de MVDs e FDs e $R = (R_1, \ldots, R_n)$ um conjunto de esquemas R_i. Então R está na 4ª forma normal ou 4NF (*4th Normal Form*) se:

(i) D implica $*(R_1, \ldots, R_n)$, onde "*" denota uma dependência de junção (*definição 5*);

(ii) $\forall R_i \in R$, se toda vez que uma MVD não trivial $X \twoheadrightarrow Y$ é válida em R_i, então a FD $X \rightarrow A$ é válida para cada A em R_i.

Definição 15 [OzY87]: Seja D um conjunto de MVDs e FDs e $R = (R_1, \ldots, R_n)$ um conjunto de esquemas R_i. Então R está na 4ª forma normal fraca ou W4NF (*Weak 4NF*) [OzY87] se:

(i) D implica $*(R_1, \ldots, R_n)$, onde "*" denota uma dependência de junção (*definição 5*);

(ii) $\forall R_i \in R$, se $D \vdash (X \twoheadrightarrow Y)$ e $XY \subset R_i$, então $D \vdash (X \rightarrow Y)$.

Definição 16: Uma relação R está na 5ª forma normal ou 5NF (*5th Normal form*) *se, e somente se*, cada dependência de junção de R etiver subtendida pelas chaves candidatas de R.

Definição 17 [OzY87]: Sejam $X, V \subseteq U$. Dizemos que X separa V se há dois dependentes W_1 e W_2 de X tal que $W_1 \cap V \neq \emptyset$ e $W_2 \cap V \neq \emptyset$.

Um conjunto M de MVDs separa V se há $Y \in LHS(M)$ tal que Y

separa V.

Definição 18 [OzY87]: Um conjunto M de MVDs é livre-de-conflito se:

(i) M não separa elementos em $LHS(M)$ e

(ii) $DEP_M(X) \cap DEP_M(Y) \subseteq DEP_M(X \cap Y)$, $\forall X, Y \in LHS(M)$.

Definição 19 [OzY87]: Suponha que haja uma chave X de M tal que exista $Z \in DEP(X)$ e $D(q) = Z \cap S(T)$. Então q é dito ser uma redundância parcial em T em relação à X se $X \subset A(u)$ (note que a redundância nesse caso se deve ao fato de que a MVD $A(u) \twoheadrightarrow D(q)$ embutida em U pode ser derivada por aumentação da MVD $X \twoheadrightarrow D(q)$).

No contexto de $S(T)$, a MVD $X \twoheadrightarrow D(q)$ é uma dependência parcial em $S(T)$.

Definição 20 [OzY87]: Se existem nós irmãos $q_1, q_2, \ldots, q_n$ de q em T tal que $W = \bigcup_{i=1}^{n} D(q_i)$, $X \subset A(u) \cup W$ e M não implica $XW \twoheadrightarrow D(q)$ no contexto de $S(T)$, então q é dito ser uma redundância transitiva em relação à X em T.

Nesse caso, $X \twoheadrightarrow D(q)$ é uma dependência transitiva em $S(T)$.

Definição 21 [RoK87]: Seja $V \subseteq U$. O conjunto de chaves fundamentais sobre V, denotado por $FK(V)$, é definido como segue:

$$FK(V) = \{\, V \cap X \mid X \in LHS(M) \wedge V \cap X \neq \emptyset \wedge \nexists Y \in LHS(M) \text{ tal que } X \cap V \supset Y \cap V \neq \emptyset \,\}.$$

(Intuitivamente, o conjunto das chaves fundamentais V consiste de todas as chaves que têm intersecções não vazias e minimais com V).

Definição 22: Uma árvore de esquema normal pode ser definida informalmente como uma árvore de esquema que não tem redundâncias parciais nem transitivas. Ainda, cada nó é uma chave fundamental (ver *definição 20*) sobre os seus descendentes. Isso faz com que as chaves não fiquem divididas entre os nós nas árvores. Formalmente, tem-se a

seguinte definição:

Um esquema de árvore T é dito ser um esquema de árvore normal em relação ao conjunto M de MVDs se:

(i) $M \vdash MVD(T)$;

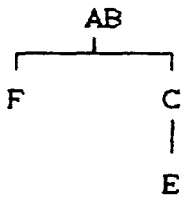
(ii) não há nenhuma dependência parcial nem transitiva em T;

(iii) a raiz de T é uma chave (*definição 12*) e para todos os demais nós u em T, se $FK(D(u)) \neq \emptyset$ então $u \in FK(D(u))$.

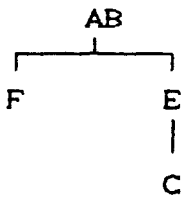
Exemplo [RoK87]:

Sejam $U = (A, B, C, E, F)$ e $M = \langle AB \twoheadrightarrow CE, BC \twoheadrightarrow F \rangle$. Considere as seguintes árvores de esquema:

T_1:



T_2:



T_1 é uma árvore de esquema normal, enquanto que T_2 não o é. Embora ambas não tenham nós redundantes, T_2 não satisfaz a condição (iii) da definição; para o nó E de T_2, $D(E) = CE$ e $FK(CE) = (C)$, mas $E \notin FK(CE)$.

Definição 23 [OzY87]: Seja M um conjunto de MVDs sobre U e $F = (T_1, \ldots, T_n)$ um conjunto de árvores de esquemas normais. F é uma floresta de esquemas normais ou NSF (*Normal Scheme Forest*) em relação a M se:

(i) $S(F) = \bigcup_{i=1}^{n} S(T_i) = U$ e

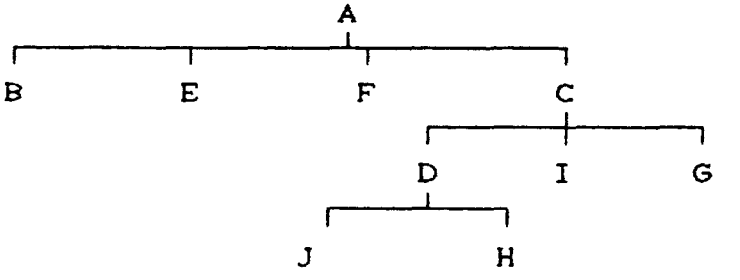
(ii) $M \vdash *(S(T_1), \ldots, S(T_n))$, onde "*" denota uma dependência de junção.

Exemplo 02YB/1:

Sejam U = ⟨A, B, C, D, E, F, G, H, I, J⟩ e
M = ⟨A ↠ B, B ↠ E|F, AC ↠ DJH, AJ ↠ H, AD ↠ J|G⟩.

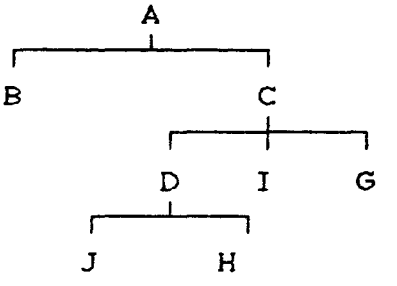
M é uma cobertura mínima. O conjunto de chaves essenciais de M é $LHS(M)$ = ⟨ A, B, AC, AJ, AD⟩ e a única chave não essencial é ACJ. Escolhendo a ordem A, B, AC, AD, AJ, ACJ para as chaves, obtém-se a árvore de esquema T abaixo:

T:

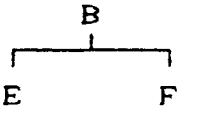


Os nós E e F são redundantes em relação à chave B, gerando, então, as árvores de esquema T_1 e T_2 abaixo:

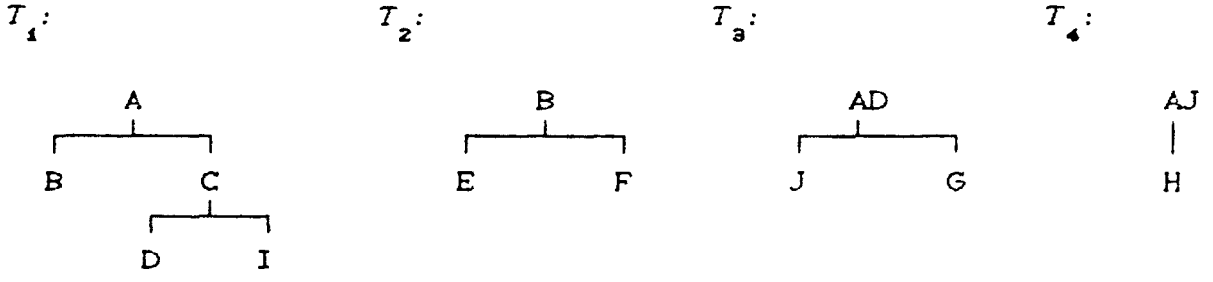
T_1:



T_2:



Retirando todos os nós redundantes em relação às demais chaves, tem-se a *floresta de esquemas normais* F = ⟨T_1, T_2, T_3, T_4⟩, onde:



2.3. NOVAS FORMAS NORMAIS PARA O MRNN

Juntamente com o renascimento da idéia de bancos de dados não-normalizados em [Mak77], **Makinouchi** estendeu os conceitos de dependências funcionais (FDs) e multivaloradas (MVDs) para manipularem conjuntos de valores, além dos valores atômicos convencionais. **Mackinouchi** propôs, então, uma nova forma normal que incorporou as dependências estendidas na definição da 4NF (*definição 14*), embora não apresentasse um método para se obter um esquema de BD nessa forma normal [OzY87].

Posteriormente, **Roth, Korth e Silberchatz** [RKS84] [RKS88] apresentam a chamada Forma Normal Particionada ou PNF (*definição 6*). Essa nova forma normal considera somente FDs, convencionais (*definição 1*) e estendidas (*definição 2*), usando igualdade de conjuntos. Em [RKS84], os autores definem um modelo baseado no de [FiT83] (ver *capítulo 3*), e um cálculo relacional equivalente a uma álgebra relacional estendida para as relações em PNF. Os autores mostram que a classe de relações em PNF é fechada sob os operadores estendidos.

Para se fazer sentir a forte ligação entre a PNF e a álgebra estendida de [RKS84], antecipar-se-á rapidamente o *capítulo 3*. Essa álgebra inclui dois operadores definidos em [FiT83] e [JaS82]: *NEST* (operador de multivaloração) e *UNNEST* (operador de monovaloração). Basicamente, *NEST* agrupa os valores de alguns atributos sob um atributo comum, formando uma sub-relação aninhada. *UNNEST*, por outro lado, desagrupa os valores de um atributo composto, resultando numa sub-relação plana formada pelos seus atributos componentes com seus

respectivos valores.

Após uma operação *NEST* sempre é possível retornar à relação original por uma operação *UNNEST*; conseqüentemente, toda seqüência de *NEST's* pode ser desfeita por uma seqüência de *UNNEST's*. O contrário, no entanto, não é válido, dificultando certas operações algébricas.

Quando uma relação está em PNF, sempre há uma seqüência de *NEST's* para se anular uma dada seqüência de *UNNEST's*. Essa propriedade é fundamental para a álgebra estendida de **Roth, Korth e Silberchatz**, como será visto no *capítulo 3*.

Segundo **Ozsoyoglu e Yuan** [OzY87], uma relação aninhada *R* está em PNF se, e somente se, o esquema de *R* é uma árvore de esquema (*definição 7*), com relação ao conjunto de MVDs dadas (devem considerar as FDs como um caso particular de MVDs). Nem todas as árvores de esquema, entretanto, são boas estruturas para relações aninhadas; podem, ainda, conter certas anomalias como redundâncias parciais (*definição 19*) e transitivas (*definição 20*).

Para se evitar tais anomalias, **Ozsoyoglu e Yuan** introduzem uma representação parametrizada da árvore de esquema chamada <u>árvore de esquema normal</u> (*definição 22*). Seus vértices são conjuntos de atributos e suas arestas representam MVDs implicadas por um dado conjunto de MVDs.

Os autores definem, então, uma nova forma normal: a <u>forma normal aninhada</u> ou <u>NNF</u> (*Nested Normal Form*). Informalmente, uma relação está em NNF se está estruturada na forma de uma árvore de esquema normal.

Ozsoyoglu e Yuan consideram a NNF como uma forma normal mais poderosa que a PNF. Note que, segundo os conceitos de [OzY87], se uma árvore de esquema normal é uma árvore de esquema, então qualquer relação que esteja em NNF está em PNF; o contrário, no entanto, nem sempre é válido.

[OzY87] apresenta um algoritmo para se obter uma decomposição NNF de um conjunto de atributos (esquema de relação universal) em relação ao conjunto de MVDs dado (considera FDs como casos particulares de MVDs). Tal decomposição pode se considerada como uma maneira formal de se projetar BDs hierárquicos utilizando-se as

MVDs entre os atributos.

Uma decomposição NNF tem certas propriedades desejáveis, tais como:

(a) representação explícita de um conjunto *M* de MVDs completas e embutidas (EMVDs - *definição 4*) implicadas por *M*, obtendo, assim, uma boa representação das conexões semânticas dos atributos com o mundo real;

(b) representação confiável e não redundante.

Outras boas propriedades associam-se a essas duas anteriores quando o conjunto M de MVDs é livre-de-conflito (*definição 18*). Segundo [OzY87], se *M* é livre-de-conflito, então a decomposição em NNF é precisamente a única decomposição em 4NF do conjunto de atributos *U* em relação a *M*. Exceto quando *M* é livre-de-conflito, há muitos modos de se decompor em NNF um conjunto de atributos *U* em relação a *M*, com vários conjuntos de caminhos em 4NF. Esses diferentes resultados dependem das chaves fundamentais (*definição 21*) selecionadas para se decompor *U* e da ordem escolhida para o uso de todas as chaves usadas no teste de dependências parciais e transitivas.

Como já foi dito, para um conjunto de atributos *U* e um conjunto *M* de MVDs, uma árvore de esquema normal é uma boa representação para as instâncias de *U* satisfazendo *M*. Entretanto, nem sempre é possível se obter uma árvore de esquema normal, mesmo quando *M* é livre-de-conflito. Nesse caso, pode-se obter um conjunto de árvores de esquema normal, o que constitui uma floresta de esquemas normais (*definição 23*).

De acordo com [OzY87], há uma outra definição para a 4NF, chamada de 4NF fraca ou W4NF (*Weak 4NF*) (*definição 15*), que considera também as EMVDs associadas às MVDs dadas. Obviamente, a 4NF implica a W4NF, mas o contrário não é válido.

Desse modo, obtém-se a seguinte generalização: se um conjunto *M* de MVDs é livre-de-conflito, então o conjunto de caminhos da raiz até as folhas numa floresta de esquemas normais é exatamente a única decomposição em W4NF (ou em 4NF, se não se considerar as EMVDs) do conjunto de atributos *U* em relação a *M*.

Em face das propriedades acima, as formas normais de [OzY87]

e de [Mak77] têm algo em comum: ambas tiram proveito das boas propriedades da 4NF do MRN, conforme discutido em [Fag77]. Isso, no entanto, não é bem um ponto de analogia entre elas, visto que ambas estão fundamentadas em princípios diferentes, o que até torna incomparáveis os conceitos de redundâncias [OzY87] ([Mak77] estende a 4NF sob FDs estendidas e [OzY87] baseia-se em dependências reais - não estendidas).

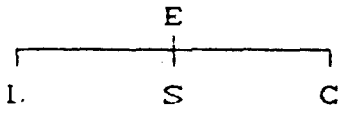
Embora os estudos de **Ozsoyoglu e Yuan** [OzY87] mostrem resultados muito animadores em relação a NNF, os próprios autores crêem que a NNF pode ser melhorada. O fato de considerar MVDs como extensões de FDs ($X \rightarrow Y \Rightarrow X \twoheadrightarrow Y$) faz com que nem todas as conexões semânticas entre os atributos na forma de FDs e MVDs sejam utilizadas para a NNF.

Seguindo a linha de raciocínio acima, **Roth e Korth** [RoK87] dão continuidade aos estudos sobre a NNF levando em conta as diferenças semânticas entre as FDs e as MVDs sobre um conjunto de atributos, além das EMVDs. Há, portanto, muitas regras de inferências fortes para EMVDs isoladamente, assim como juntas com as MVDs e FDs.

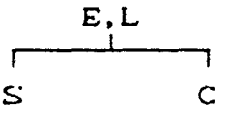
Antes mesmo do artigo [RoK87] ser publicado, **Ozsoyoglu e Yuan** já haviam combinado FDs e MVDs num chamado "conjunto envelope" (*envelope set*) de dependências. Segundo [RoK87], **Ozsoyoglu e Yuan** propunham, então, usar tal conjunto como entrada para um algoritmo de decomposição NNF *ligeiramente* modificado em relação ao anterior. Só que usando tal adaptação do algoritmo, é comum aparecerem conjuntos singulares (*singletons*) quando se usa FDs na decomposição.

Os exemplos abaixo ilustram o problema da distinção semântica entre FDs e MVDs [RoK87].

"Seja $U = (E, L, S, C)$ e $D = (E \twoheadrightarrow S, E \rightarrow L)$, onde E é um empregado ID, S são seus filhos e L é seu endereço. Usando o método de **Ozsoyoglu e Yuan**, $E \rightarrow L$ implicaria em $E \twoheadrightarrow L$, gerando uma relação em NNF, cuja árvore de esquema está mostrada em (a) abaixo. Por outro lado, se $E \rightarrow L$, cada conjunto de L criado por este esquema será um conjunto singular (*singleton*), o que faz [RoK87] usar a árvore de esquema (b)."



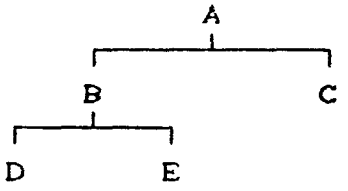
(a)



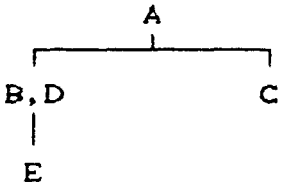
(b)

Pelo exemplo acima, pode-se notar que numa decomposição NNF as FDs podem acarretar conjuntos singulares se a MVD representada por uma aresta numa árvore de esquema também é uma FD. Em geral, o aninhamento dos valores é necessário quando $u \longrightarrow q$ numa aresta (u, q).

"Considere a árvore de esquema (c) . Se B $\longrightarrow$ D é válida, então cada valor de B tem um valor singular D associado a ele. Portanto, os valores de D não precisam ser aninhados, gerando uma estrutura menor, como mostrado em (d). Essa última estrutura é consistente com a de (c), visto que as MVDs de (c) implicam nas MVDs de (d)".



(c)



(d)

Roth e Korth [RoK87] propoem uma nova aproximação para a decomposição NNF, segundo os passos abaixo.

Sejam *U* um conjunto de atributos a ser usado no projeto, *D* um conjunto de FDs e MVDs e *F* um conjunto de EMVDs dadas.

Primeiramente, usando-se as regras de inferência conhecidas, geram-se todas as dependências FDs, MVDs e EMVDs implicadas pelas EMVDs de entrada ou pelas EMVDs juntamente com todas as FDs e MVDs conhecidas. As novas EMVDs são adicionadas à F. Gera-se, então, um conjunto *M'* constituído das MVDs implicadas pelas MVDs de *D* e fecho dos lados esquerdos. Fecha-se, também, os lados esquerdos das EMVDs em *F* (já alterado) usando as FDs de D.

O segundo passo consiste em usar M' como entrada de um algoritmo de decomposição 4NF, pelos mesmos motivos de [OzY87], citados anteriormente. A escolha de Roth e Korth foi decidida entre duas aproximações, as quais consideram FDs distintamente de MVDs: uma de Beeri e Kifer [BeK84] e Katsuno [Kat84] e a outra de Yuan e Ozsoyoglu [YuO86].

Na primeira aproximação, gera-se um conjunto M' de MVDs a partir de um conjunto D de FDs e MVDs dado. Para isso, obtém-se uma versão completa das MVDs em D e então substitui-se os lados esquerdos X de cada MVD na versão completa pelo fecho de X em relação a D.

Na segunda aproximação, dado um conjunto D de FDs e MVDs, cria-se um conjunto $E(D)$ de MVDs, chamado "conjunto envelope", que representa as dependências adequadas ao processo. Segundo [RoK87], nesse método, componentes de MVDs completas são eliminadas se eles são também componentes de FDs.

Roth e Korth adotam o primeiro método, justificando que a segunda aproximação não faz qualquer tentativa de associar os atributos dependentes funcionalmente com as chaves.

Obtém-se, então, um conjunto de esquemas $R = (R_1, \ldots, R_n)$ em 4NF em relação às MVDs M'. Entretanto, cada esquema de R pode ainda ter uma ou mais FDs implicadas pelas MVDs embutidas nele. Caso isso ocorra, para cada esquema R_i é executada uma síntese 3NF temporária, obtendo-se os esquemas $S = (S_1, \ldots, S_k)$. Através de informações de F, cada S_j, quando conveniente, é decomposto e adicionado a R.

O próximo passo da aproximação de [RoK87] consiste na decomposição NNF. Os autores acreditam que utilizando uma decomposição inicial em 4NF, o projetista de BD tenha mais controle sobre os objetos e relacionamentos a serem enfatizados no projeto final. Para isso utilizam uma versão simplificada do algoritmo de Ozsoyoglu e Yuan. O procedimento é mais simples porque o conjunto de atributos dado forma um esquema em 4NF, havendo precisamente só um dependente na base de dependências (*definição 9*) de qualquer conjunto de atributos que seja um subconjunto de um esquema em 4NF.

Finalmente, as árvores de esquema geradas são combinadas por um algoritmo, mantendo-se a NNF e a decomposição em 4NF.

O exemplo abaixo ilustra as etapas da aproximação de Roth e Korth [RoK87]:

Considere o conjunto de atributos U = (*classe, dia, hora, monitor, sala, estudante, média e exame*). As dependências válidas sobre U são:

$$D = (\ classe \twoheadrightarrow dia,\ estudante \rightarrow média,$$
$$monitor \rightarrow sala,\ classe,\ estudante \twoheadrightarrow exame,$$
$$classe,\ monitor \twoheadrightarrow hora\ ).$$

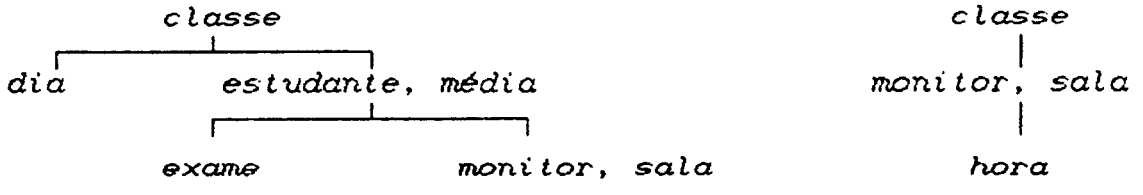
Obtém-se, então, o conjunto M' implicado pelas MVDs completas e pelo fecho de $LHS(M)$:

$$M' = (classe \twoheadrightarrow Dia \mid hora,\ monitor,\ sala,\ estudante,\ média,\ exame;$$
$$classe,\ estudante,\ média \twoheadrightarrow exame \mid dia \mid hora,\ monitor,\ sala;$$
$$classe,\ monitor,\ sala \twoheadrightarrow hora \mid dia \mid exame,\ estudante,\ média\ ).$$

A única decomposição em 4NF consiste em quatro esquemas :

classe, dia
classe, estudante, média, exame
classe, monitor, sala, hora
classe, monitor, sala, estudante, média.

O algoritmo de decomposição, então, resulta no seguinte conjunto de esquemas em NNF:

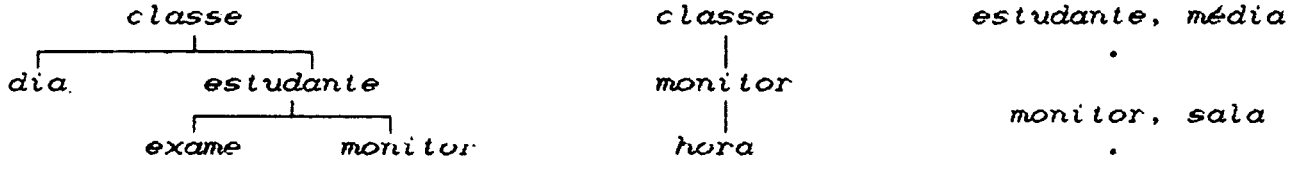


Veja que há algumas redundâncias óbvias em relação à M', envolvendo os nós: "*monitor, sala*" e "*estudante, média*". Como é válida a dependência *estudante* → *média*, cada vez que um valor de estudante é repetido para diferentes valores de *classe*, a *média* também

é repetida. A situação é pior no outro caso. Se *monitor* → *sala* é válida, cada vez que um valor de *monitor* é repetido para diferentes valores de *classe* e *estudantes* em uma relação, e para diferentes valores de *classe* em outras relações, o mesmo valor de *sala* também é repetido.

Isso se deu porque os grupos de atributos aninhados introduzem redundâncias se um subconjunto do grupo determina funcionalmente alguma outra parte do grupo. Esse problema já não ocorre se o grupo é a raiz da árvore de esquema, porque eles são os atributos atômicos da relação e seus valores ocorrem uma única vez.

Desse modo, a decomposição em NNF final é dada pelas quatro árvores de esquema abaixo:



2.4. ESTENDENDO FORMAS NORMAIS DO MRN PARA O MRNN

Ao invés de criarem novas formas normais para o MRNN, alguns pesquisadores simplesmente adaptaram as formas normais do MRN, com exceção da 1NF.

Um desses estudiosos, Setzer [Set86], justifica sua direção de pesquisa dentro da teoria do MRNN, alegando que a decomposição em 1NF de uma relação não é um pré-requisito para as demais formas normais introduzidas para o MRN. Segundo o autor de [Set86], as justificativas para a 1NF quando não são evitadas, podem ser facilmente contestadas.

Assim, usando o conceito de FDs estendidas (*definição 2*), Setzer considera as MVDs *naturalmente* válidas sobre conjuntos compostos ou multivalorados, como FDs que também manipulam conjuntos de valores. Essas FDs estendidas decorrem do conceito de tupla

de uma relação aninhada.

Desse modo, preservam-se as definições da 2NF, 3NF e da BCNF, mesmo com colunas compostas e/ou multivaloradas. Como a 4NF do MRN *era* usada para retirar as redundâncias geradas pela representação multivalorada das multivalorações e isso já é eliminado nesse novo contexto, ela não é mais necessária. Em todo caso, supondo que alguém tenha representado uma relação monovaloradamente e esta já esteja em BCNF, pode-se, então, aplicar-lhe a 4NF. O resultado não deve ser uma decomposição, mas a multivaloração das colunas onde isso é possivel.

Para efeitos da 4NF, segundo o conceito acima, Setzer [Set86] introduz um operador tv não comutativo semelhante ao *NEST*, que produz a multivaloração de uma coluna X do esquema R: $tv_X(R)$. O operador $tv_X(R)$ produz uma nova relação R' com colunas de mesmo nome que as de R, tal que os valores das colunas complementares a X se preservam e todas as tuplas nas quais esses valores coincidam são representadas por uma só tupla com os valores de X agrupados na forma de um conjunto.

Como exemplo, considere a relação LIVROS abaixo. Essa relação está em BCNF, pois não há nenhuma FD.

LIVROS:

NÚMERO_DE_ CHAMADA	AUTOR	ASSUNTO
1	AU1	AS1
1	AU1	AS2
1	AU2	AS1
1	AU2	AS2
2	AU1	AS3
2	AU1	AS4
2	AU1	AS1
2	AU3	AS3
2	AU3	AS4
2	AU3	AS1

Devido às duas MVDs NÚMERO_DE_CHAMADA $\twoheadrightarrow$ AUTOR|ASSUNTO, aplica-se a 4NF convencional (*definição 14*) sobre LIVROS, resultando na seguinte decomposição:

LIVROS1:

NÚMERO_DE_CHAMADA	AUTOR
1	AU1
1	AU2
2	AU1
2	AU3

LIVROS2:

NÚMERO_DE_CHAMADA	ASSUNTO
1	AS1
1	AS2
2	AS3
2	AS4
2	AS1

Por outro lado, executando-se as operações $tv_{\mathbf{AUTOR}}$(LIVROS) e $tv_{\mathbf{ASSUNTO}}$(LIVROS) nessa ordem, obtém-se a relação abaixo:

LIVROS3:

NÚMERO_DE_CHAMADA	AUTOR*	ASSUNTO*
1	AU1 AU2	AS1 AS2
2	AU1 AU3	AS3 AS4 AS1

Oliveira [Oli88] também segue a mesma linha de Setzer [Set86], com algumas alterações. Oliveira também estende as três formas normais do MRN, juntamente com a BCNF, usando um conceito de FD estendida similar ao da *definição 2*, mas não chega a fazer menção explícita de uma adaptação da 4NF; menciona apenas que essa torna-se desnecessária, por motivos análogos ao de [Set86]. Entretanto, Oliveira estende também a 5NF (*definição 16*) pelos mesmos motivos que ela foi introduzida no MRN, ou seja, devido às anomalias de

atualização, eliminação e inserção de tuplas em relação a MVDs.

Como exemplo, considere a seguinte relação:

PROJETOS:

#FORNECEDOR	#PEÇA	#PROJETO*
F1	P1	PRO1 PRO2
F1	P2	PRO1
F2	P1	PRO1

PROJETOS está em BCNF e em 4NF, mas de modo a satisfazer a restrição de dependência de junção, se, por exemplo, a tupla (F2, P2,PRO2) for inserida, então as tuplas (F2, P2, PRO1) e (F2, P1, PRO2) também devem ser inseridas. Por outro lado, se a tupla (F1, P1, PRO1) for eliminada, (F1, P2, PRO1) também deve ser excluída da relação.

Desse modo, PROJETOS pode ser decomposta nas relações abaixo sem perda de informações.

#FORNECEDOR	#PEÇA*
F1	P1 P2
F2	P1

#PEÇA	#PROJETO*
P1	PRO1 PRO2
P2	PRO1

#PROJETO	#FORNECEDOR*
PRO2	F1
PRO1	F1 F2

3. O SUBSISTEMA DE MANIPULAÇÃO ALGÉBRICA

3.1. INTRODUÇÃO

Este capítulo descreve problemas envolvendo operações em álgebra de relações aninhadas, comparando diversas propostas. É dada especial atenção à álgebra de relações de Fischer e Thomas [FiT83], cujas operações são utilizadas no protótipo desenvolvido.

3.2. HISTÓRICO

Desde 1977, quando **Makinouch [Mak77]** levantou a necessidade de relações aninhadas, muitas pesquisas vieram sendo desenvolvidas nesse sentido. Os estudos se concentravam em duas áreas: criação de relações aninhadas a partir de relações convencionais e propostas de operações para relações aninhadas.

Jaeschke e Schek [JaS82] introduziram os operadores *NEST*, ν, que "aninhava" as operações planas e *UNNEST*, μ, que fazia a operação inversa. As relações aninhadas, entretanto, limitavam-se a um único nível de aninhamento e eram formadas apenas por atributos singulares.

Como se pretendia conservar ainda as boas características do modelo relacional convencional, **Jaeschke e Schek [JaS82]** apresentaram, ainda, possibilidades de se estender os operadores da álgebra relacional para operar com as relações em NF^2. Estenderam, assim, os operadores de seleção, projeção, união, diferença e produto cartesiano com as comparações de conjuntos.

A limitação de níveis de aninhamento (apenas um) e a omissão de atributos compostos, no entanto, acarretavam a perda das

propriedades desejáveis de hierarquia do MRNN. Assim, em 1983, começaram a surgir trabalhos que generalizavam esses estudos.

Em [Tho83] e [FiT83], Fischer e Thomas expandem os trabalhos de [JaS82], estendendo os operadores *NEST* e *UNNEST* para relações em NF^2 com vários níveis de aninhamento e com atributos compostos e multivalorados; preservam, assim, a *visão* de hierarquia do MRNN. Esses autores estendem, ainda, os demais operadores estudados em [JaS82], adequando-os a esse modelo generalizado.

Em [FiT83], os autores mostram, ainda, que as limitações impostas pelos estudos de [JaS82] geram propriedades nem sempre válidas referentes aos operadores estendidos.

Em [Tho83] foram introduzidas duas operações variantes de *NEST* e *UNNEST*: a de *aninhamento total*, indicada por NEST sem parâmetros, que, dada uma relação, aninha todos os atributos possíveis, e o *desaninhamento total*, indicado por UNNEST*, que transforma uma relação aninhada numa relação plana.

Em [Rot86], Roth define um conjunto de qualidades desejáveis, a partir das quais propõe extensões dos operadores apresentados em [Tho83]. Definições semelhantes de muitos desses operadores são apresentadas em termos do modelo VERSO em [AbB84] e [Bid87]. As propriedades formais enunciadas em [AbB84] se referem a relações universais. Para Abiteboul e Bidoit, as relações aninhadas são chamadas de *formatos*. Apresentam um operador de multivaloração denominado *reestruturação* e uma união especial, denominada *fusão*.

Seguindo, ainda, a metodologia de [Rot86], Roth e Kirkpatrick [RoK88] apresentam duas propriedades, desejando que sejam garantidas para os novos operadores. A primeira dessas propriedades, confiabilidade, requer que um operador relacional estendido quando aplicado a relações planas convencionais chegue ao mesmo resultado a que chegaria o operador não estendido. A segunda, precisão, é relativa ao desaninhamento de relações aninhadas: *se* um operador estendido for aplicado a relações aninhadas e sobre esse resultado aplicar-se um *UNNEST**, *então* deve-se chegar ao mesmo resultado aplicando-se o operador equivalente não estendido sobre as mesmas relações completamente desaninhadas.

A segunda das propriedades acima requer que o resultado da

aplicação de um operador para relações aninhadas seja totalmente desaninhado e *normalizado*. Essa necessidade de *normalização* deve-se ao fato de que ela sempre proporciona uma seqüência de operações *NEST* dada uma seqüência válida de *UNNEST's*. Essa propriedade não é verdadeira para todos os casos, mas torna-se sempre válida se considerada uma normalização de esquemas como a *"Partioned Normal Form"* (*PNF*) e a *"Nested Normal Form"* (*NNF*), já vistas anteriormente (*seção 2.3*).

Em [Rot86] são apresentadas demonstrações formais das propriedades desejáveis dos operadores para relações em *PNF*. Já em [RoK88] a análise é feita em torno da *NNF*, embora apresente apenas definições de dois operadores estendidos: intersecção ($\cap^*$) e junção ($\ast^*$). Roth, Korth e Silberchatz [RKS88] também definem uma álgebra aninhada para relações em PNF, provando a sua equivalência com o cálculo relacional em não-1NF, também propostos pelos autores.

Convém ressaltar que a necessidade de *normalização* surge principalmente por causa da operação de junção estendida, visto que ela pode envolver diferentes e complexas estruturas, obtendo-se um resultado não muito claro [RoK88].

Além das álgebras relacionais estendidas mencionadas, há outras propostas, como, por exemplo, a de *álgebras recursivas* proposta inicialmente por Schek e Scholl [ScS86] e posteriormente estendida por Deshpande e Larson [DeL87]. Esse tipo de álgebra leva em conta expressões algébricas para substituirem atributos nos predicados de seleções e em listas de projeções. Essas álgebras são excelentes candidatas para especificação de consultas e para planos de execução de consultas internamente ao sistema de BD. Roth e Kirkpatrick [RoK88] juntamente com Batory [RKB87], propoem fazer uso de álgebras recursivas não-normalizadas para otimização e implementação de esquemas.

Alguns pesquisadores têm apresentado trabalhos sobre linguagens de consultas para BDs relacionais não-normalizados. Abiteboul e Bidoit [AbB84], por exemplo, apresentam uma linguagem usando quantificadores existenciais. A maioria dos estudos, porém, recai sobre uma extensão da linguagem relacional SQL,

tornando-a ainda mais poderosa, sem aumentar a complexidade das operações originais.

Em [ScP82] Schek e Pistor associam à SQL operadores apropriados para colunas multivaloradas.

Pistor e Traunmuller [PiT86] descrevem uma interface de linguagem de consulta para um modelo NF^2 estendido, o qual permite que os elementos de um conjunto de valores estejam ordenados, recebendo o nome de *lista*. O objetivo dessa interface é que as consultas possam ser tratadas de uma maneira *ad-hoc*. Deixam em aberto, para futuros trabalhos, uma melhor exploração do potencial da linguagem SQL.

Roth, Korth e Batory [RKB87] também investigam um tipo de linguagem de manipulação de dados do tipo SQL.

Ozsoyoglu e Ozsoyoglu descrevem uma linguagem de alto nível orientada para tela, para manipulação de um sistema de BD estatístico, chamada *Summary-Table-by-Example*, ou, simplesmente, *STBE*, também baseada na SQL. A linguagem *STBE* é baseada num cálculo relacional estendido que permite funções agregados e relações com atributos multivalorados. Os autores comparam *STBE* com SQL, mostrando que as inovações acrescentadas à SQL fazem da *STBE* uma linguagem mais poderosa.

3.3. DEFINIÇÃO DA ÁLGEBRA USADA NO SUBSISTEMA

Dentre as álgebras mencionadas na seção anterior, a de **Fischer e Thomas** [FiT83] foi a selecionada para servir de base ao subsistema proposto nesta tese. Essa escolha é devido ao fato de que essa álgebra possui várias características desejáveis.

Primeiramente, em [FiT83] os autores dão um grande passo à frente em relação a [JaS82], generalizando os operadores relacionais estendidos. Além disso, segundo [RoK88], foi em [FiT83] que foram introduzidas as mais poderosas versões estendidas de *NEST* e *UNNEST*. Isso é um fator muito importante, visto que esses operadores fundamentam as demais operações e permitem várias aplicações, como por exemplo:

(a) podem ser usados para descrever formalmente o mapeamento conceitual-físico num BD;

(b) podem ser usados por um administrador de BD para definir visões não-normalizadas a partir de esquemas conceituais normalizados.

Fischer e Thomas não se preocupam com a *normalização* dos esquemas aninhados, como é o caso de [Rot86] e [RoK88]. Evidentemente, o uso de formas normais para esquemas em NF^2 tem suas vantagens, mas a complexidade introduzida também é maior. Para efeitos da tese, entretanto, a simplicidade da álgebra de Fischer e Thomas é mais conveniente, ressaltando-se que essa álgebra tornou-se o ponto de partida para várias extensões nessa área. Os próprios autores em [FiT83] comentam que os resultados ali discutidos podem ser usados para descrever formas normais ainda não definidas em relação ao MRNN. Concluem que o objetivo é se chegar a estruturas que possam ser estudadas por uma série de operações *NEST e UNNEST*, sem restrições. Isto é justamente o que ocorre quando se supõe a *normalização* sobre as relações não-aninhadas, visto que, sem ela, nem sempre é possível encontrar uma seqüência de *NEST's* anulando uma dada seqüência de *UNNEST's*.

Nesta seção são apresentados os pontos fundamentais da álgebra proposta por [FiT83], visto que os algoritmos propostos na tese estão nela baseados. À medida em que serão apresentados os conceitos dessa álgebra, tentar-se-á compará-la às de [JaS82] e de [RoK88], de modo a se transmitir as diferenças marcantes entre elas.

3.3.1. Notação

Esta subseção introduz a notação usada por [FiT83], adotada na tese. Tal notação tem sido usada em outros artigos, como em [Tho83] e [Rot86], e é baseada nos conceitos de *lógica de BD*.

Entende-se como esquema do banco de dados S, uma coleção de regras da forma

$$R_j = (R_{j_1} , R_{j_2} , \ldots , R_{j_n}),$$

onde os elementos R_{j_i}, $1 \leq i \leq n$, são atributos da relação definida pela regra R_j e são denotados por E_{R_j}.

R_j é um atributo de ordem alta se ele aparece no lado esquerdo de alguma regra, caso contrário ele é um atributo de ordem zero. Se um atributo de ordem alta só aparecer em uma única regra, ele é chamado de atributo externo.

Em um esquema de BD não pode haver duas regras com o mesmo conjunto de nomes do lado direito.

Seja R_j um nome externo num esquema de BD S. As regras em S que são acessíveis a partir de R_j formam um subesquema de S. Esse subesquema é chamado esquema de relação se:

(1) $R_j = (R_{j_1} , R_{j_2} , \ldots , R_{j_n})$ está no esquema;

(2) Quando um nome de ordem alta R_j está no lado direito de alguma regra no esquema, a seguinte regra também pertence ao esquema: $R_j = (R_{j_1} , R_{j_2} , \ldots , R_{j_n})$.

(3) Nenhum nome deve aparecer no lado direito de duas regras diferentes no esquema.

Para todo BD ou esquema de relação S, pode-se definir um único grafo direcionado, T_S, que é a estrutura de árvore de S. Note que isso é oposto à árvore de esquema de Oszoyuglu e Yuan [OsY87]. O conjunto de nós de T_S é formado exatamente por todos os atributos de R_j. Além disso, TS contém um ramo direcionado (R_i , R_j) de R_i para R_j se, e somente se, $R_j \in E_{R_i}$.

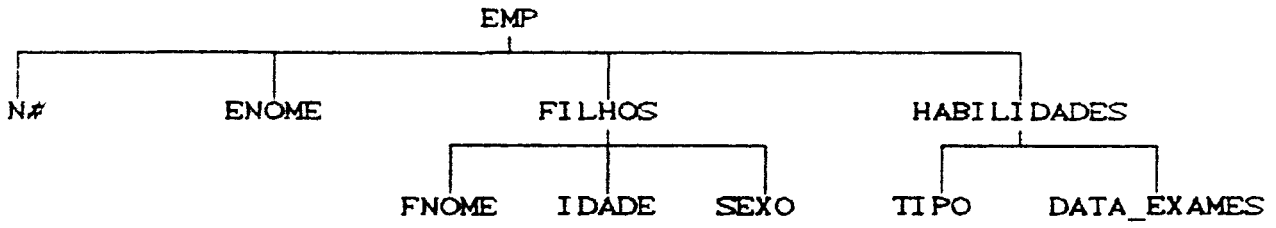
Considere, por exemplo, um BD de empregados (EMP), onde cada um deles é determinado por um número de identificação (NÚ), contendo seu nome (ENOME), informações sobre seus filhos (FILHOS) e sobre suas

habilidades (HABILIDADES). Um possível esquema de BD poderia consistir das regras:

EMP = (N#, ENOME, FILHOS, HABILIDADES),
FILHOS = (FNOME, IDADE, SEXO),
HABILIDADES = (TIPO, DATA_EXAME).

Nesse caso, EMP, FILHOS e HABILIDADES são os atributos de ordem alta, sendo que EMP é um *atributo externo*. Note que se identificássemos ambos ENOME e FNOME simplesmente por NOME, EMP não poderia ser considerado um esquema de relação, pois infringiria a restrição (3) vista anteriormente.

A estrutura de árvore, T_{EMP} , corresponde a um grafo direcionado de altura 2 (dois):



Percebe-se, assim, que o termo "relação aninhada" descreve a situação em que um atributo de ordem alta aparece no lado direito de alguma regra, e, portanto, "aninha" uma relação dentro de outra relação.

Seja R um atributo num esquema de BD S. Uma <u>ocorrência de</u> R, dada por r, é um par ordenado da forma (R, V_R), onde V_R é um valor para o nome R. Quando R é de *ordem zero*, V_R é apenas um conjunto do domínio de R. Quando R é um nome de *ordem alta*, V_R deve ser expandido em termos dos nomes do lado direito da regra R.

Uma <u>estrutura de relação</u>, ou, simplesmente, <u>relação</u> $\overline{R} = (R, r)$ denota um esquema de relação R e uma ocorrência r.

Para representar os componentes A_1, A_2, ..., A_n de uma tupla t de uma relação, será usada a notação:

$$t[A_1 A_2 \ldots A_n]$$

Convém lembrar que as tuplas no MRNN podem conter estruturas de relações embutidas como componentes, além dos tradicionais valores atômicos. Usando-se os conceitos acima também se pode definir esquemas para relações planas, bastando para isso que todos os elementos sejam de ordem zero. Por essa razão são, às vezes, denominadas "supertuplas" [FiT83] [DeG88].

3.3.2. Definição dos Operadores Relacionais Estendidos

A seguir são descritos operadores relacionais estendidos, segundo a álgebra de [FiT83], de forma comparativa. Para cada operador, além de uma definição informal, são dados sua definição formal e um exemplo, acompanhados de alguns comentários ilustrativos.

3.3.2.1. Aninhamento: NEST

NEST é um operador que considera uma estrutura de relação $\bar{R} = (R, r)$ e agrega os valores iguais de um subconjunto de atributos em R. Excluindo-se o atributo a ser aninhado por *NEST*, esse operador requer a projeção dos demais atributos. Essa projeção forma as partições (ou combinações distintas) sobre as quais o atributo aninhado será agrupado [FiT83].

A definição de **Fischer e Thomas** leva em conta cada elemento de um conjunto associado a uma dada partição, a fim de torná-los um conjunto de atributos, ao invés de meramente um atributo singular, como é feito em [JaS82]. **Jaescke e Skeck** particionam as tuplas sobre a base de atributos não-aninhados e colecionam os conjuntos de atributos aninhados em cada uma dessas partições. Novamente, a chave para a extensão feita por **Fischer e Thomas** está no fato de que cada elemento do atributo aninhado pode ser composto de outros atributos.

Definição:

Seja um BD S com um atributo externo R, definido pela regra

$$R = (A_1, A_2, \ldots, A_n)$$

Sejam

$$(B_1, B_2, \ldots, B_m) \subset E_R, \quad 1 \leq m < n \quad e$$

$$(C_1, C_2, \ldots, C_k) = E_R - (B_1, B_2, \ldots, B_m), \quad 1 \leq k \leq n-m.$$

Suponha que a regra $B = (B_1, B_2, \ldots, B_m)$ não esteja em R, ou seja, que B não apareça no lado esquerdo de nenhuma regra de R, nem $(B_1, B_2, \ldots, B_m)$ do lado direito de nenhuma delas.

Então, a operação *NEST* em relação a B é denotada

$$NEST_{B=(B_1, B_2, \ldots, B_m)}(\bar{R}) = (R', r') = \bar{R}', \qquad \text{onde:}$$

(1) $R' = (C_1, C_2, \ldots, C_k, (B_1, B_2, \ldots, B_m)) =$
$= (C_1, C_2, \ldots, C_k, B)$,
ficando a regra $B = (B_1, B_2, \ldots, B_m)$ anexada ao conjunto de regras de R, se ainda não o tiver sido;

(2) $r' = \{ t \mid \exists\, u \in r$ *tal que*
$t[C_1, C_2, \ldots, C_k] = u[C_1, C_2, \ldots, C_k] \wedge$
$\wedge\ t[B] = \{ w[B_1, B_2, \ldots, B_m]$ *onde* $w \in r \wedge$
$\wedge\ w[C_1, C_2, \ldots, C_k] = t[C_1, C_2, \ldots, C_k]) \}$.

Como exemplo, considere a relação $\bar{R}$ abaixo e a operação *NEST* a seguir:

$\bar{R}$:

A	B	C
0	0	1
0	1	0
0	0	0

$NEST_{B'=B}(\bar{R})$:

A	B	C
	B'	
0	0	1
0	1 0	0

Em palavras, é feita projeção dos atributos em $E_R - B$ e depois aninhamento dos valores de B para cada atributo do resultado da projeção.

Vale observar que a propriedade de <u>comutatividade</u> <u>não é válida</u> para dois *NEST's*, visto que podem resultar em estruturas claramente diferentes, ou seja, $NEST_A(NEST_B(R)) \neq NEST_B(NEST_A(R))$.

Há outras maneiras de se obter operações semelhantes à *NEST*, como, por exemplo, a usada por Oliveira em [Oli88], fazendo uso de um *operador de multivaloração*. A operação consiste no *aninhamento* dos valores de atributos específicos sem que haja um *aninhamento* dos atributos envolvidos. [Set86] define um operador de aninhamento *tv* que produz a multivaloração de uma só coluna X já existente, cuja definição, segundo o próprio autor, é mais simples que a de [JaS82].

3.3.2.2. Desaninhamento: UNNEST

UNNEST é o inverso de *NEST*. O desaninhamento permite que se torne *plana* uma estrutura aninhada formada por um operador *NEST*.

Na sua execução, cada elemento de um atributo aninhado é repetidamente combinado com sua partição associada aos atributos restantes.

<u>Definição</u>:

Seja $R = (A_1, A_2, \ldots, A_n)$ uma regra de um esquema de S, onde R é um atributo externo, o que significa que há uma relação de esquema R. Suponha que B seja um atributo de ordem alta em E_R de modo que

$$B = (B_1, B_2, \ldots, B_m), \quad 1 \leq m < n.$$

Seja, também,

$$(C_1, C_2, \ldots, C_k) = E_R - B, \quad 1 \leq k \leq n - m.$$

Então,

$$UNNEST_B(\bar{R}) = (R', r') = \bar{R}', \qquad \text{onde:}$$

(1) $R' = (C_1, C_2, \ldots, C_R, B_1, B_2, \ldots, B_m)$. A regra $B = (B_1, B_2, \ldots, B_m)$ é removida do conjunto de regras em S se esta não aparecer em nenhum outro esquema de relação;

(2) $r' = \{t \mid \exists\, u \in r$ tal que $t[C_1, C_2, \ldots, C_R] = u[C_1, C_2, \ldots, C_R] \wedge t[B_1, B_2, \ldots, B_m] \in u[B]\}$.

Considere a relação $\bar{R}$ a seguir e veja o resultado obtido com uma aplicação de *UNNEST* sobre um de seus atributos:

$\bar{R}$:

A	B'	C'
	B	C
0	0	0
	1	
0	0	1

$UNNEST_B(\bar{R})$:

A	B	C'
		C
0	0	0
0	1	0
0	0	1

Ao contrário de *NEST*, o operador *UNNEST* é <u>comutativo</u>, ou seja, $UNNEST_A(UNNEST_B(\bar{R})) = UNNEST_B(UNNEST_A(\bar{R}))$.

Além disto, uma operação *NEST* sempre pode ser desfeita por um subseqüente *UNNEST*, sem perda de informação, i.e.,

$$UNNEST_B(NEST_B(\bar{R})) = \bar{R}.$$

O contrário, no entanto, não é válido, porque há estruturas que não reassumem a forma aninhada somente com *UNNEST* seguido de *NEST*, ou seja, $NEST(UNNEST(\bar{R})) \neq \bar{R}$.

Desta forma, se sobre uma estrutura plana for aplicada uma série de *NEST's* e, posteriormente, uma série de *UNNEST's*, sempre é possível se recuperar a estrutura original em 1NF. O contrário, infelizmente, não é verdade: a álgebra de Fischer e Thomas não permite recuperar sempre uma estrutura aninhada, uma vez que ela seja aplainada. Para que isso também seja válido é que [Rot86] e [RoK88] necessitam da *normalização* dos esquemas aninhados. Esse resultado lhes

é muito útil no caso da operação de junção.

A álgebra de Fischer e Thomas [FiT83] inclui, ainda, uma extensão de *UNNEST* : é o *UNNEST**, que representa uma série de *UNNEST*'s aplicados sobre os atributos de ordem alta (enquanto existirem) de uma estrutura aninhada, de modo a transformá-la numa estrutura em 1NF.

3.3.2.3. Propriedades dos Operadores Binários Estendidos

Assim como na álgebra relacional convencional as operações podem ser compostas de acordo com a consulta solicitada. No caso de se trabalhar também com relações em NF^2, os operadores *NEST* e *UNNEST* também poderão interagir com os demais operadores. Essa interação pode, no entanto, correr riscos de ser mal definida, causando resultados imprecisos, tendo em vista a propriedade de comutatividade e dependência de ordem. Além disso, precisa-se saber com exatidão até que ponto os operadores básicos podem ser executados sobre estruturas em NF^2 com efeitos que sejam transparentes a um usuário com uma visão 1NF do BD.

Jaeschke e Schek [JaS82] definem uma álgebra em que os operadores relacionais possuem determinadas propriedades quando interagem com *NEST* e *UNNEST*.

São quatro as propriedades estudadas, as quais estão relacionadas entre si. Considere e como sendo um dos operadores binários: união, intersecção, diferença, produto cartesiano e junção. Assim,

(P1) $NEST_B (\bar{R}_1 \; e \; \bar{R}_2) = NEST_B(\bar{R}_1) \; e \; NEST_B(\bar{R}_2)$

(P2) $UNNEST_B (\bar{R}_1 \; e \; \bar{R}_2) = UNNEST (\bar{R}_1) \; e \; UNNEST (\bar{R}_2)$

(P3) $\bar{R}_1 \; e \; \bar{R}_2 = UNNEST_B (NEST_B(\bar{R}) \; e \; NEST_B(\bar{R}_2))$

(P4) $UNNEST^* (\bar{R}_1 \; e \; \bar{R}_2) = UNNEST^* (\bar{R}_1) \; e \; UNNEST^* (\bar{R}_2)$.

Na álgebra aqui adotada, de [FiT83], nem sempre essas

propriedades são válidas, porque a definição das operações binárias não é a mesma que em [JaS82].

Uma diferença em termos de operações entre [JaS82] e [FiT83] está na maneira mais simples com que esses últimos fazem comparações de tuplas. Para Fischer e Thomas, uma comparação de tuplas envolve apenas uma comparação simples de um atributo singular aninhado de cada relação: para cada nível singular somente um aninhamento, como veremos a seguir.

3.3.2.4. União

Para que seja possível a união entre duas relações $\bar{R}_1$ e $\bar{R}_2$, suas estruturas de esquema devem ser iguais e o seu resultado é dado por:

$$\bar{R}_1 \cup \bar{R}_2 = (R_1, r') = \bar{R}',$$

onde $r' = \{ t \mid t \in r_1 \vee t \in r_2 \}$.

Exemplo:

$\bar{R}_1$:

A	B'
	B
0	1 2
1	1 6

$\bar{R}_2$:

A	B'
	B
3	1 2
1	1

$\bar{R}_1 \cup \bar{R}_2$:

A	B'
	B
0	1 2
1	1 6
3	1 2
1	1

O operador de *União* satisfaz as propriedades (P2), (P3) e (P4) *(seção 3.3.2.3)*, como demonstrado em [FiT83], mas não satisfaz (P1).

3.3.2.5. Intersecção

Para se executar a intersecção estendida de duas estruturas de relação $\bar{R}_1$ e $\bar{R}_2$, ambos esquemas de relação devem ser iguais. O resultado obtido é definido como:

$$\bar{R}_1 \cap \bar{R}_2 = (R_1, r') = \bar{R}',$$

onde $r' = \{ t \mid t \in r_1 \wedge t \in r_2 \}$.

Para este operador, infelizmente, nenhuma das quatro propriedades de [JaS82] pode ser garantida. O problema é que para Fischer e Thomas [FiT83] a comparação de tuplas é muito simplificada, envolvendo as tuplas como um todo.

Para exemplificar a diferença, considere um exemplo que trate de empregados de uma empresa identificados por um certo número e informações sobre suas habilidades. Suponhamos que os dados estejam armazenados como abaixo:

$\overline{EMP1}$:

NUM_EMP	HABILIDADES
	NOME
#1	professor advogado
#2	advogado jornalista

$\overline{EMP2}$:

NUM_EMP	HABILIDADES
	NOME
#2	advogado
#1	advogado historiador

Segundo a álgebra de [FiT83], o resultado de $EMP1 \cap EMP2$ é vazio, enquanto que as informações comuns entre as duas relações são as seguintes:

NUM_EMP	HABILIDADES
#1	advogado
#2	advogado

Já no exemplo abaixo, o resultado da intersecção é vazio em ambas as álgebras:

$\overline{EMP3}$:

NUM_EMP	HABILIDADES	
	NOME	ANO_EXAME
#1	professor	1963
	advogado	1972
#2	advogado	1965
	jornalista	1980

$\overline{EMP4}$:

NUM_EMP	HABILIDADES	
	NOME	ANO_EXAME
#2	advogado	1970
#1	advogado	1963
	historiador	1980

O resultado atingido via [FiT83] considera as comparações do conjunto de valores de HABILIDADES como um todo em relação ao valor da chave NUM_EMP.

3.3.2.6. Diferença

A diferença estendida entre duas relações $\bar{R}_1$ e $\bar{R}_2$ com esquemas iguais ($R_1 = R_2 = R$) resulta em:

$$\bar{R}_1 - \bar{R}_2 = (R, r') = \bar{R}',$$

onde $r' = \{ t \mid t \in r_1 \wedge t \notin r_2 \}$.

Nenhuma das propriedades P1, P2, P3 ou P4 (*seção 3.3.2.3*) é válida para a diferença estendida. Vejamos, por exemplo, uma contradição para P1.

$\bar{R}_1$:

A	B	C
0	4	5
0	6	5
1	2	3

$\bar{R}_2$:

A	B	C
0	6	7
1	3	3
1	2	3

$\bar{R}_1 - \bar{R}_2$:

A	B	C
0	4	5
0	6	5

$NEST_{B'=(B)}(\bar{R}_1 - \bar{R}_2)$:

A	B'	C
	B	
0	4 6	5

$NEST_{B'=(B)}(\bar{R}_1)$:

A	B'	C
	B	
0	4 6	5
1	2	3

$NEST_{B'=(B)}(\bar{R}_2)$:

A	B'	C
	B	
0	6	7
1	3 2	3

$$NEST_{B'=(B)}(\bar{R}_1) - NEST_{B'=(B)}(\bar{R}_2):$$

A	B'	C
	B	
0	4 6	5
1	2	3

3.3.2.7. Produto Cartesiano

Sejam $\bar{R}_1$ e $\bar{R}_2$ duas estruturas de relação com esquemas R_1 e R_2 em um esquema de BD S. Suponha que S contenha as regras:

$$R_1 = (R_{11}, R_{12}, \ldots, R_{1n}) \quad e$$

$$R_2 = (R_{21}, R_{22}, \ldots, R_{2m}).$$

Então, o produto cartesiano de $\bar{R}_1$ e $\bar{R}_2$ é definido por:

$$\bar{R}_1 \times \bar{R}_2 = (R, r) = \bar{R}, \qquad \text{onde:}$$

(1) $R = (R^1_{11}, R^1_{12}, \ldots, R^1_{1n}, R^2_{21}, R^2_{22}, \ldots, R^2_{2m})$

e, para toda regra $R_k = (R_{k1}, R_{k2}, \ldots, R_{kl})$ em um esquema de relação R_1, o esquema de relação R contém uma regra $R^1_k = (R^1_{k1}, R^1_{k2}, \ldots, R^1_{kl})$, assim como para toda regra $R_k = (R_{k1}, R_{k2}, \ldots, R_{kl})$ em R_j, R contém uma regra $R^2_k = (R^2_{k1}, R^2_{k2}, \ldots, R^2_{kl})$;

(2) $r = (t \mid \exists (u \in r_1 \wedge q \in r_2)$ tal que

$$t[R^1_{11} R^1_{12} \ldots R^1_{1n}] = u[R^1_{11} R^1_{12} \ldots R^1_{1n}] \wedge$$

$$\wedge\ t[R^2_{21} R^2_{22} \ldots R^2_{2m}] = q[R^2_{21} R^2_{22} \ldots R^2_{2m}].$$

Veja o sequinte exemplo:

$\bar{R}_1$:

A	B		C
	B1	B2	
0	1 2	0 1	3
2	1	2	0

$\bar{R}_2$:

A	D
	D1
0	6 4
1	5

$\bar{R}_1$ ⋈ $\bar{R}_2$:

A^1	B^1		C^1	A^2	D^2
	$B1^1$	$B2^1$			$D1^2$
0	1 2	0 1	3	0	6 4
0	1 2	0 1	3	1	5
2	1	2	0	0	6 4
2	1	2	0	1	5

Das quatro propriedades apresentadas na *seção 3.3.2.3*, somente (P4) se aplica ao produto cartesiano, como demonstrado em [FiT83]. Vale, entretanto, uma variação de (P2), também devidamente demonstrada por Fischer e Thomas. Essa variação é dada como segue:

Sejam $\bar{R}_1$ *e* $\bar{R}_2$ *estruturas de relações, onde* $A \subset E_{R_1}$ *e* $B \subset E_{R_2}$ *são nomes de ordem alta associados com as regras:*

$$A = (A_1, A_2, \ldots, A_k) \quad e \quad B = (B_1, B_2, \ldots, B_l).$$

Então,

$$UNNEST_B\,(UNNEST_A\,(\bar{R}_1 \text{ ⋈ } \bar{R}_2)) = UNNEST_A(\bar{R}_1) \text{ ⋈ } UNNEST_B(\bar{R}_2).$$

3.3.2.8. Junção

O operador de junção para relações em NF^2 é muito interessante e polêmico. Embora uma relação aninhada já contenha junções embutidas, estender essa operação para lidar consistentemente com o aninhamento não é nada trivial. Em [Rot86], o autor ressalta que "operações de junção são difíceis de serem definidas no mundo aninhado devido à possibilidade de se trabalhar com diferentes níveis de aninhamento para os atributos". É o caso, por exemplo, envolvendo as estruturas de esquemas R_1 e R_2 abaixo:

$$R_1 = (A, X) \ , \ X = (B, C) \qquad e \qquad R_2 = (B, D).$$

Executar a junção entre $\bar{R}_1$ e $\bar{R}_2$ resulta num produto cartesiano, pois não há atributos comuns do mesmo nível. Por outro lado, se $\bar{R}_1$ for desaninhada, B será um atributo comum e a junção recairá sobre ele.

Esse problema pode ser solucionado limitando-se as estruturas de relações envolvidas, no que se refere aos atributos comuns. Desse modo, há várias maneiras plausíveis de se estender a junção natural do MRN para relações não-normalizadas.

Jaeschke e Schek [JaS82] definem dois métodos para a junção natural de relações aninhadas. O <u>primeiro método</u> pode ser considerado simplesmente uma *extensão da junção natural* ($\bar{\bowtie}$). As comparações de valores, no entanto, envolvem igualdade de conjuntos quando os atributos são conjuntos valorados. O <u>segundo método</u> é chamado de *junção de intersecção* ($\bar{\bar{\bowtie}}$). Nesse método, toma-se a intersecção de conjuntos aninhados comuns quando se combinam as tuplas para a junção. "*Se* as intersecções de todos os conjuntos aninhados comuns não são vazias e a junção das tuplas se dá sobre atributos atômicos, *então* essas tuplas fazem parte do resultado" [RoK88].

Fischer e Thomas [FiT83] estendem a <u>junção de intersecção</u> de [JaS82] para estruturas com atributos compostos e aninhamentos com mais de um nível. A definição é dada como segue:

Definição:

"Sejam R_1 e R_2 dois esquemas num BD S. Sejam $X = E_{R_1} \cap E_{R_2}$, $(A_1, A_2, \ldots A_n) = E_{R_1} - X$ e $(B_1, B_2, \ldots, B_m) = E_{R_2} - X$.

Suponha, também, que $X_1 \subset X$ seja formado pelos atributos de ordem zero de X e $X_2 \subset X$, pelos atributos de ordem alta.

Então,

$$\bar{R}_1 \bar{\times} \bar{R}_2 = (R, r) = \bar{R} ,$$ onde:

(1) $R = (A_1, A_2, \ldots, A_n, X, B_1, B_2, \ldots, B_m)$;

(2) $r = \{t \mid \exists(u \in r_1) \wedge \exists(q \in r_2)$ tal que

$$t[A_1 A_2 \ldots A_n] = u[A_1 A_2 \ldots A_n] \wedge$$
$$\wedge\ t[B_1 B_2 \ldots B_m] = q[B_1 B_2 \ldots B_n] \wedge$$
$$\wedge\ t[X_1] = u[X_1] = q[X_1] \wedge$$
$$\wedge\ t[X_2] = (u[X_2] \cap q[X_2]) \neq \emptyset\}."$$

Nenhuma das quatro propriedades da *seção 3.3.2.3* é válida para a junção de intersecção estendida. **Fischer e Thomas [FiT83]** comentam que o fato de [JaS82] considerarem **(P3)** válida sobre múltiplos atributos se deve ao fato de que estes, possivelmente, não consideram a aplicação de *NEST* sobre objetos de ordem alta.

Como exemplo da junção de intersecção de **Fischer e Thomas**, vejamos o caso abaixo, o qual servirá também como contra-exemplo de **(P1)**.

$\bar{R}_1$:

A	B	C
		C'
0	1	1 2

$\bar{R}_2$:

D	B	C
		C'
1	1	2 3

$\bar{R}_1 \ \bar{\times} \ \bar{R}_2$:

A	B	C	D
		C'	
0	1	2	1

$NEST_{E=(B,C)}(\bar{R}_1 \ \bar{\times} \ \bar{R}_2)$:

A	E		D
	B	C	
		C'	
0	1	2	1

Os atributos X da intersecção são BC, sendo B de ordem zero (i.e., $X_1 = B$) e C de ordem alta ($X_2 = C$).

$NEST_{E=(B,C)}(\bar{R}_1)$:

A	E	
	B	C
		C'
0	1	1 2

$NEST_{E=(B,C)}(\bar{R}_2)$:

D	E	
	B	C
		C'
0	1	2 3

$NEST_{E=(BC)}(\bar{R}_1) \ \bar{\times} \ NEST_{E=(BC)}(\bar{R}_2)$:

A	E		D
	B	C	
		C'	
∅			

Observe que o resultado final é vazio, pois a intersecção dos atributos de ordem alta é vazia. Como $X = (E)$, onde $E = (B,C)$ e $C = (C')$, $\neg \exists u \in (NEST_{E=(B,C)}(\bar{R}_1))$ e $\neg \exists q \in (NEST_{E=(B,C)}(\bar{R}_2))$ tal que $u[X] \cap q[X] \neq \emptyset$. Isso pode ficar mais claro se observarmos que C é composto em ambos esquemas, portanto seu valor é um conjunto de subtuplas, cujos valores pertencem a C'. Desse modo, para que o

conjunto de valores de C em $\bar{R}_1$ pertença ao resultado, cada subtupla sua (de valores 1 e 2) deveria ter uma subtupla equivalente em C de $\bar{R}_2$, o que não ocorre.

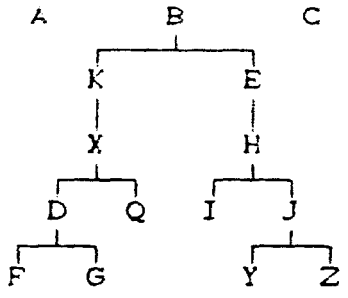
A álgebra apresentada de [FiT83] indica que a junção de intersecção toma elementos comuns como um *todo*, isto é, para um atributo ser comum a dois esquemas de relação R_1 e R_2, ele deve pertencer a $E_{R_1} \cap E_{R_2}$ e, ainda, todas as regras descendentes desse atributo devem ser iguais. Note que a última junção realizada acima tomou $X = (E)$ porque os componentes de E se têm a mesma composição nos dois esquemas de relação envolvidos.

Roth e Kirkpatrick [Rot86] [Kir88] evitam a limitação acima através da *normalização* segundo a *PNF*.

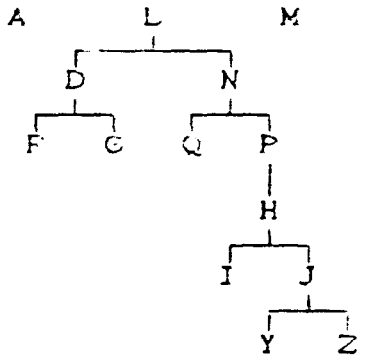
Em [RoK88], os autores ressaltam que as extensões de junção apresentadas em [Tho83] e [FiT83] estão longe de propiciar efeitos sobre relações aninhadas arbitrárias. Infelizmente, as propriedades desejáveis segundo [RoK88] *(seção 3.2)* não são satisfeitas, especialmente a de *precisão*. De modo a suprir essa deficiência, [Rot86] e [Kir88] apresentam um método chamado *junção de extensão* (x^*).

O método apresentado em [Rot86] pode ser aplicado sobre dois esquemas quaisquer, desde que tenham pelo menos um atributo de ordem alta comum, juntamente com suas regras descendentes. Isso se aplica mesmo nos casos em que tais atributos aparecem em níveis diferentes nos dois esquemas. Esse método consiste numa *remodelagem* das estruturas envolvidas na junção, representadas sob a forma de grafos. Essa *remodelagem* [Kir88] é feita através de aplicações sucessivas de *UNNEST's* sobre cada estrutura até que os atributos de ordem alta comuns *subam* até o topo do grafo, tornando os esquemas passíveis à x^*. Deve-se ressaltar que o grafo usado não inclui o atributo de ordem alta externo (nome da relação) como em [FiT83]. Uma vez executada a operação x^*, são aplicados sucessivos *NEST's* às estruturas para que retornem as suas formas iniciais. A normalização neste caso é importante porque possibilita que uma seqüência de *NEST's* sempre anule uma de *UNNEST's*.

Sejam as estruturas abaixo:

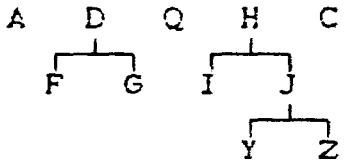


(a)

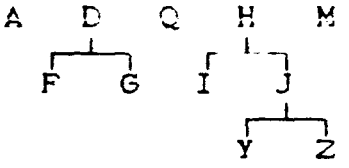


(b)

Note que há duas subárvores comuns às árvores de (a) e (b), em diferentes níveis. *Remodelando* as duas estruturas acima chega-se aos esquemas (a') e (b'), respectivamente, permitindo, então, que seja aplicado o operador $\alpha^{\bullet}$.



(a')



(b')

Segundo [RoK88], essa reestruturação das relações envolvidas tem como objetivo manter uma estrutura específica, mas ainda constitui um método aparentemente limitado. Assim, os autores trabalham com as relações em *NNF*. Com isso elas podem ser *reajuntadas* exatamente na ordem inversa em que foram formadas, voltando a um estado comparável ao que tinham antes da decomposição da estrutura inicial. Desse modo, dado um conjunto de relações em *NNF*, junções arbitrárias podem ser executadas entre quaisquer duas relações sem a necessidade prévia de reestruturá-las. Por outro lado, ao se juntar as relações que foram separadas durante o processo da *normalização*, dependências parciais são reintroduzidas no resultado, embora este ainda se mantenha livre das dependências transitivas.

Em [Oli88], Oliveira, baseado na álgebra de [JaS82], apresenta duas extensões da junção relacional convencional: uma usada quando os atributos envolvidos na junção são monovalorados (atômicos) denominada *junção natural para* NF^2 ($*$) e outra quando os atributos comuns são multivalorados, *junção de intersecção* ($\bar{*}$). No primeiro método, é possível se obter um esquema resultante com dois subatributos de mesmo nome, o que não é permitido na álgebra de [FiT83]. Em cada um dos métodos, o autor considera dois casos:

(1) quando o atributo comum é um subatributo nas duas relações, assim como na relação resultante. Parece que esse caso independe dos níveis do subatributo nas relações, visto que serão partes de atributos distintos.

(2) quando o atributo comum é um subatributo em apenas um esquema de relação. Nesse caso, o esquema resultante o conterá como subatributo.

3.3.2.9. Seleção

Como na álgebra relacional convencional, um operador de seleção em NF^2 permite a recuperação das tuplas de uma relação que satisfazem a um certo predicado. Os predicados no MRNN incluem operadores de comparação de conjuntos, além dos operadores lógicos usuais.

Definição:

Suponhamos um esquema de relação R e um predicado p.

A execução de uma operação de *seleção estendida* sobre $\bar{R}$ resulta em:

$$\sigma_p(\bar{R}) = (R, r') = \bar{R}',$$

onde $r' = \{ t \mid t \in r_1 \wedge t \text{ satisfaça } p \}$.

Para a seleção, as propriedades de [JaS82] são adaptadas

como segue:

$$(P1)'\ NEST_{B=(Y)}\ (\ \sigma_p(\bar{R})\ ) = \sigma_p\ (\ NEST_{B=(Y)}\ (\bar{R})\ )$$

$$(P2)'\ UNNEST_B\ (\ \sigma_p(\bar{R})\ ) = \sigma_p\ (\ UNNEST_B\ \bar{R})\ )$$

$$(P3)'\ UNNEST_B\ (\ \sigma_p\ (\ NEST_B(\bar{R})\ ) = \sigma_p(\bar{R})$$

$$(P4)'\ UNNEST^*\ (\ \sigma_p(\bar{R})\ ) = \sigma_p\ (\ UNNEST^*(\bar{R})\ )$$

Das propriedades acima, (P1)', (P2)' e (P3)' só são válidas quando os predicados p das seleções não envolverem nomes B ou Y. (P4)', por sua vez, é válida quando o predicado p envolve somente atributos de ordem zero.

Veja no exemplo de seleção abaixo, um contra-exemplo para (P1)' envolvendo atributos aninhados nos predicados [Fit83]:

$\bar{R}$:

A	B
0	1
1	0
1	1

$\sigma_{B=1}(\bar{R})$:

A	B
0	1
1	1

$NEST_{B'=B}\ (\sigma_{B=1}(\bar{R}))$:

→

A	B'
	B
0	1
1	1

$NEST_{B'=(B)}(\bar{R})$:

A	B'
	B
0	1
1	0 1

$\sigma_{1\subset 1}(NEST_{B'=(B)}(\bar{R}))$:

→

A	B'
	B
0	1
1	0 1

3.3.2.10. Projeção

Sejam $\bar{R}$ uma estrutura de relação e

$$X = (X_1, X_2, \ldots, X_n) \subset E_{\bar{R}}.$$

A projeção estendida de $\bar{R}$ sobre X é dada, então, como:

$$\pi(\bar{R}) = (R', r') = \bar{R}', \qquad \text{onde:}$$

(1) $R' = (X_1, X_2, \ldots, X_n)$;
(2) $r' = \{ t' \mid \exists\, u \in r$ tal que $t = u[X] \wedge t' = t$, se t ainda não pertencer a $r' \}$.

Veja, agora, o exemplo abaixo:

$\bar{R}$:

A	E		D
	B	C	
		C'	
0	1	2	0
		3	
	4	1	
		5	
1	2	3	1

$\pi_{AE}(\bar{R})$:

A	E	
	B	C
		C'
0	1	2
		3
	4	1
		5
1	2	3

Para projeção, as quatro propriedades são definidas como segue:

(P1)" $NEST_{B=(Y)}(\pi_{XY}(\bar{R})) = \pi_{XB}(NEST_{B=(Y)}(\bar{R}))$

(P2)" $UNNEST_B(\pi_{XB}(\bar{R})) = \pi_{XY}(UNNEST_B(\bar{R}))$, *onde* $B = (Y)$

(P3)" $\pi_{XY}(\bar{R}) = UNNEST_B(\pi_{XB}(NEST_{B=(Y)}(\bar{R})))$

(P4)" $UNNEST^*(\pi_Z(\bar{R})) = \pi_Z(UNNEST^*(\bar{R}))$, *onde* $Z = (Z_1, Z_2, \ldots, Z_n)$ *é uma coleção de conjuntos de ordem alta de*

R e Z' consiste nos nomes de todos os nós folhas de todas as subárvores de R de raiz Z_i.

A propriedade (P4)" é sempre válida enquanto que (P2)" e (P3)" só são válidas supondo-se que $X = (X_1, X_2, \ldots, X_n) \subset E_R$ e que $B \in E_R$ é um atributo de ordem alta. (P1)", por outro lado, nem sempre é satisfeita, como se pode ver no contra-exemplo a seguir:

$\bar{R}$:

A	B	C
0	0	2
0	1	2
0	1	3

$NEST_{B'=(B)}(\pi_{AB}(\bar{R}))$:

A	B'
	B
0	0
	1

$\pi_{AB'}(NEST_{B'=(B)}(\bar{R}))$:

A	B'
	B
0	0
	1
0	1

4. Estruturas de Armazenamento de Relações

4.1. INTRODUÇÃO

Neste capítulo são apresentadas as estruturas de armazenamento propostas para o sistema desenvolvido.

A hipótese básica é que uma relação aninhada tenha somente um atributo chave, simples e monovalorado, que sempre ocupará a primeira posição na ordem dos atributos. Esta simplificação visa diminuir a complexidade das estruturas de índice utilizadas e dos algoritmos de comparação de tuplas.

Vale observar que os exemplos seguem a linha implícita de [FiT83], onde a multivaloração está associada à composição dos atributos, ou seja, todo atributo multivalorado é composto. As estruturas de dados, no entanto, são mais gerais, pois permitem modelar casos em que atributos multivalorados não são compostos.

Para cada relação $\bar{R} = (R, r)$ será utilizado um conjunto de duas estruturas:

(i) estrutura de armazenamento do esquema ou ESQ_R: contém as informações necessárias sobre o esquema R, incluindo informação para o caminho de acesso a um atributo, a partir da raiz, seguindo a árvore do esquema. Esta última informação é denominada identificador genérico de um atributo;

(ii) estrutura de armazenamento dos dados ou DADOS_R: contém os valores de r com seus respectivos identificadores, para facilitar o acesso aos dados.

A relação $\overline{PROJ}$ a seguir será usada para exemplificar essas estruturas nas seções seguintes. Sua chave é o atributo DEPTO:

$\overline{PROJ}$:

DEPTO	PROJETOS				EQPTO	
	PNUM	PNOME	PMEMBROS / MNOMES	PVERBA	EQUANT	ETIPO
Rio	B608	FXP	Carlos Lauro	680.000	5 20	PC/AT PC/XT
	B700	Neve	Jairo Lauro	700.000	1	386/SX
Mat	M101	Mat1	Cesar Paulo Decio	104.000	2 1	PC/XT PC/AT

4.2. ESTRUTURA DE ARMAZENAMENTO DE ESQUEMAS

A informação necessária para manutenção do esquema é armazenada em uma *tabela* com seis linhas e (n - 1) colunas, onde n é o número de nós do grafo correspondente ao esquema.

Esta tabela permite que se tenha controle da composição de um esquema quanto a seus componentes e inter-relacionamentos entre eles. Contém, ainda, informações auxiliares para o acesso aos dados de uma relação.

Os dados contidos nessa tabela, à qual nos referiremos por

ESQ_ <nome_do_esquema>,

são classificados por linhas, conforme as normas abaixo:

1ª linha: contém os rótulos dos nós do grafo, com exceção da raiz (nome da relação). A seqüência desses rótulos é obtida

através de um percurso recursivo *semelhante* ao pré-ordem, começando do filho mais à esquerda da raiz.

2ª linha: indica o tipo do atributo quanto à sua composição. Para um *atributo simples*, usamos o valor indicativo *0 (zero)*, exceção feita à *chave*, que recebe um valor negativo. Um *atributo composto*, por sua vez, é indicado por um valor j > 0, onde j é o número de elementos da subárvore da qual ele é raiz.

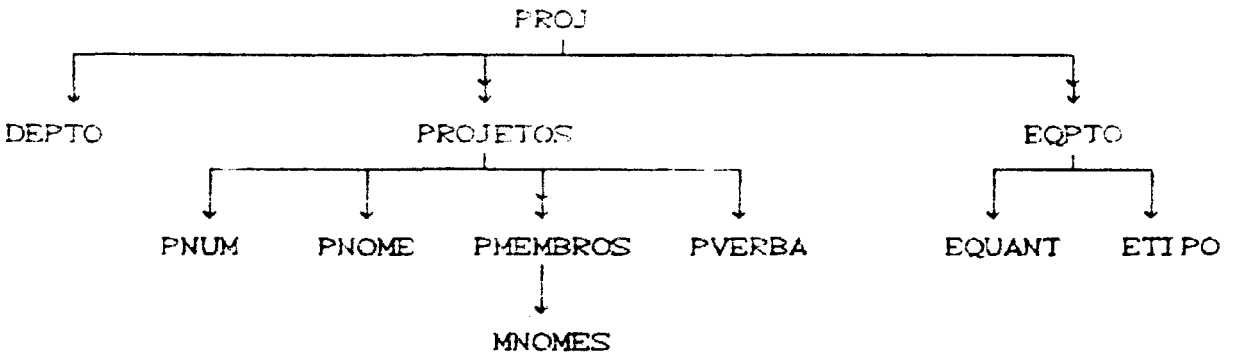
3ª linha: indica a posição do atributo em relação aos seus irmãos, sendo que o mais à esquerda tem posição *1 (um)*, o próximo a sua direita, *2 (dois)*, e, assim, sucessivamente.

4ª linha: indica o tipo do atributo quanto à sua valoração. Usa-se o valor *0 (zero)*, para um *atributo atômico* e *1 (um)* para um *atributo multivalorado*.

5ª linha: indica o nível do atributo em relação ao grafo do esquema, considerando que a raiz inicial está no nível 0 (zero).

6ª linha: contém a forma genérica do identificador de cada atributo. Esta linha é explicada com detalhes na seção seguinte, já que é fundamental para a execução das operações algébricas.

Como exemplo, considere a árvore abaixo, correspondente ao esquema de $\overline{PROJ}$ *(seção 4.1)*, onde "$\longrightarrow$ " indica *monovaloração* e "$\longrightarrow\!\!\!\!\!\rightarrow$ ", *multivaloração*. Note que o valor de n (número de nós), nesse caso, é *11*.



A tabela abaixo mostra a estrutura de armazenamento do esquema de $\overline{PROJ}$, *ESQ_PROJ*, porém sem a 6ª *linha*, visto que a formação dos identificadores genéricos dos atributos será vista com detalhes na próxima seção.

ESQ_PROJ:

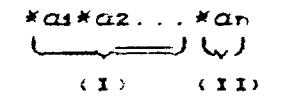
	1	2	3	4	5	6	7	8	9	10
1	DEPTO	PROJETOS	PNUM	PNOME	PMEMBROS	MNOMES	PVERBA	EQPTO	EQUANT	ETIPO
2	-1	5	0	0	1	0	0	2	0	0
3	1	2	1	2	3	1	4	3	1	2
4	0	1	0	0	1	0	0	1	0	0
5	1	1	2	2	2	3	2	1	2	2

4.3. IDENTIFICADORES DE ATRIBUTOS - LINHA 6

A 6ª linha da tabela de esquema contém os identificadores genéricos dos atributos.

O tamanho dos identificadores de uma relação depende do número de níveis do grafo de seu esquema. Se esse possui k níveis, os identificadores terão (k x 2) elementos.

A forma geral do identificador de um atributo *A* é dada por:



onde n é o número de níveis do grafo. Caso o identificador ainda não tenha (k x 2) elementos, acrescentam-se quantos zeros forem necessários a sua direita para completar o seu tamanho. O identificador indica que o caminho na árvore para atingir *A* é $a_1, a_2, \ldots, a_n$, percorrido na ordem de geração de *ESQ_R*.

Como exemplo, tem-se **2*3*1* que corresponde ao identificador de MNOMES. Este identificador genérico indica que o caminho para o nó MNOMES consiste em visitar o 2.° (segundo) atributo de nível 1 (PROJETOS), o 3.° (terceiro) atributo filho de PROJETOS (PMEMBROS) e o 1.° (primeiro) filho de PMEMBROS.

Já o atributo EQUANT possui o identificador **3*100*, indicando que o caminho a ser percorrido visita primeiramente o nó EQPTO (3.° atributo de nível 1) e depois seu 1.° filho de nível 2. Como o grafo de $\overline{PROJ}$ possui três níveis (k=3), o tamanho do identificador deve ser 6 (k x 2), daí a razão dos dois zeros complementares.

A forma genérica também pode ser escrita semelhante à forma de [DeG88]:

```
 a1 a2 ... an      ← coluna da relação
 *  *  ...  *      ← número da tupla
```

Quando os "*" de um identificador genérico do atributo *A* são substituídos por números, obtém-se o identificador de um <u>valor</u> de *A*. Este identificador permite acessar o valor do atributo dentro da estrutura de dados DADOS_R.

A parte (I) do identificador genérico serve para o direcionamento do percurso na estrutura de dados até a região onde deve estar o conjunto de valores do atributo desejado. A parte (II) é que identifica o atributo e o valor procurados.

A construção dos identificadores genéricos segue as normas abaixo, utilizando um método semelhante ao de Deshpande e Gucht

[DeG88]:

(a) Se o atributo é chave ou pertence ao nível 1 do grafo, seu identificador é composto por "*" seguido do valor contido na linha 3 de *ESQ_R* na respectiva coluna (posição em relação aos seus irmãos).

(b) Se o atributo não pertence ao nível 1 do grafo, seu identificador é formado pelos elementos dados em *(i)* seguidos pelos elementos gerados em *(ii)* abaixo:

(i) componentes do identificador do primeiro atributo composto de nível imediatamente inferior e que esteja à sua esquerda na tabela de esquema;

(ii) marca "*" seguida do valor correspondente na *linha 3* de *ESQ_R* (posição em relação aos irmãos).

Segundo as regras acima, os atributos da relação $\overline{PROJ}$ têm os seguintes identificadores genéricos:

DEPTO:	*10000	MNOMES:	*2*3*1
PROJETOS:	*20000	PVERBA:	*2*400
PNUM:	*2*100	EQPTO:	*30000
PNOME:	*2*200	EQUANT:	*3*100
PMEMBROS:	*2*300	ETIPO:	*3*200

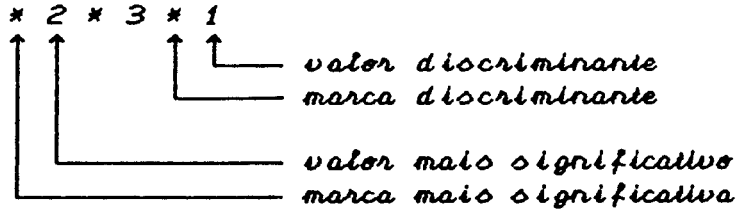
No texto que segue, adotar-se-á algumas convenções:

(i) Chamamos de valor mais significativo ao posicionado mais à esquerda do identificador genérico de um atributo. A marca "*" mais à esquerda desse identificador é dita marca mais significativa.

(ii) Ao valor distinto de zero mais à direita do identificador genérico de um atributo, denominamos valor discriminante. A marca "*" mais à direita desse

identificador é referenciada por marca discriminante, uma vez que será substituída por valores que discriminarão as tuplas/subtuplas da relação.

Desse modo, considerando-se o identificador do atributo MNOMES, como exemplo, tem-se:



A tabela de armazenamento de esquemas da relação $\overline{PROJ}$ da *seção anterior*, pode ser apresentada agora com seu conteúdo completo, ou seja, com a 6ª *linha*, que contém os identificadores genéricos dos atributos:

ESQ_PROJ:

	1	2	3	4	
1	DEPTO	PROJETOS	PNUM	PNOME	...
2	-1	5	0	0	...
3	1	2	1	2	...
4	0	1	0	0	...
5	1	1	2	2	...
6	*10000	*20000	*2*100	*2*200	...

	5	6	7	8	9	10	
...	PMEMBROS	MNOMES	PVERBA	EQPTO	EQUANT	ETIPO	1
...	1	0	0	2	0	0	2
...	3	1	4	3	1	2	3
...	1	0	0	1	0	0	4
...	2	3	2	1	2	2	5
...	*2*300	*2*3*1	*2*400	*30000	*3*100	*3*200	6

4.4. ESTRUTURA DE ARMAZENAMENTO DOS DADOS

Considerando uma estrutura de relação $\bar{R} = (R, r)$, cada valor de r tem um identificador. Primeiramente, vejamos como são formados tais identificadores. Para isso, consideremos alguns valores de $\overline{PROJ}$ (*seção 4.1*).

Note que o valor Bio pertence ao atributo DEPTO. A forma genérica desse atributo é **10000* (ver *tabela de identificadores*). Como Bio é o valor da primeira tupla de DEPTO, seu identificador passa a ser *110000*. Por sua vez, Mat, que é o segundo valor do atributo, tem o identificador *210000*.

Seja o valor Neve, que pertence a PNOME, cuja forma genérica é **2*200*. A parte (I) do identificador, composta pelos dois primeiros caracteres **2*, deve indicar em qual tupla de $\overline{PROJ}$ está o valor. Nesse caso, como ele pertence à primeira tupla, esses caracteres assumem a forma *12*. A parte (II), representada por **2*, refere-se à subtupla de PROJETOS (atributo pai de nível mais alto) à qual Neve pertence, ou seja, à segunda. Logo seu identificador passa a ser *122200*. Sob a notação de [DeG88], ter-se-ia ρ^{22}, supondo-se ρ o representante de $\overline{PROJ}$.

Veja, agora, o caso do valor Jairo do atributo MNOMES, cujo identificador é **2*3*1*. A parte (I) é composta, então, por **2*3*. A parte referente ao atributo PROJETOS deve indicar que para se

encontrar Jairo deve-se começar pela primeira tupla da relação, mas a partir do segundo atributo (o próprio PROJETOS). Assim, os dois primeiros caracteres passam a *12*. Quanto ao atributo PMEMBROS, o caminho é indicado por *23*, visto que Jairo pertence à segunda subtupla de PROJETOS e ao seu terceiro atributo. Finalmente, a parte (II) do identificador, **1*, passa a *11*, por estar na primeira subtupla de PMEMBROS. Assim, o identificador do valor Jairo é *122311* ou, segundo [DeG88], ρ_{121}^{231}.

Para reforçar a definição dos identificadores, vejamos, ainda, o último valor PC/AT de $\overline{PROJ}$. Ele pertence ao atributo ETIPO, cuja forma é **3*200*. O primeiro "*" deve indicar a segunda tupla de $\overline{PROJ}$. A segunda marca deve indicar, por sua vez, à qual subtupla de EQPTO PC/AT pertence, ou seja, a segunda. Desse modo seu identificador é dado por *232200* ou ρ_{22}^{32}, segundo [DeG88].

Cada dado possui um identificador de tupla ou tid (*tuple-identifier*), cuja finalidade é possibilitar a localização real (física) do valor armazenado. Essa posição é dada relativamente à posição inicial da estrutura de dados da relação, vista, nesse caso, como *zero*. Devido a esse endereçamento não ser relativo ao segmento do BD, segundo [Dad86], esse tipo de identificador é chamado de um *mini-identificador* de tupla ou *mini-tid*.

Os dados de uma relação são armazenados em uma tabela com duas colunas, onde a primeira contém os identificadores dos valores que estão na segunda coluna. Essa tabela é referenciada por

DADOS_ <nome_do_esquema>

Na página seguinte apresentamos a tabela de armazenamento dos dados da relação $\overline{PROJ}$ (*seção 4.1*):

DADOS_PROJ:

	1	2
1	110000	Bio
2	121100	B608
3	121200	FXP
4	121311	Carlos
5	121321	Lauro
6	121400	680.000
7	122100	B700
8	122200	Neve
9	122311	Jairo
10	122321	Lauro
11	122400	700.000
12	131100	5
13	131200	PC/AT
14	132100	20
15	132200	PC/XT
16	133100	1
17	133200	386/SX
18	210000	Mat
19	221100	M101
20	221200	Mat1
21	221311	Cesar
22	221321	Paulo
23	221331	Decio
24	221400	104.000
25	231100	2
26	231200	PC/XT
27	232100	1
28	232200	PC/AT

5. ALGORITMOS PARA EXECUÇÃO DAS OPERAÇÕES NAS ESTRUTURAS PROPOSTAS

5.1. INTRODUÇÃO

Neste capítulo são apresentados os algoritmos das operações definidas na *seção 3.3* sobre as estruturas propostas no *capítulo 4*.

A funcionalidade dos algoritmos será demonstrada através de exemplos.

Supõe-se que cada operação dê origem a uma nova relação. Os algoritmos geram, portanto, novas estruturas de esquema e de identificadores para a relação resultante. Outra alternativa seria alterar as estruturas originais, com destruição das relações anteriores.

Por um lado, a primeira técnica acima exige muito espaço de armazenamento, mas, por outro, permite o reaproveitamento posterior dos argumentos para novas operações.

5.2. ALGORITMO DE ANINHAMENTO *(NEST)*

O algoritmo de aninhamento resulta, primeiramente, numa mudança na disposição dos atributos, uma vez que estes são agrupados (aninhados) sob o nome especificado na operação. Cada tupla resultante do aninhamento é formada considerando-se inicialmente os atributos não envolvidos na operação *NEST* que tenham valores iguais. Em seguida, agrupa-se as subtuplas dos atributos especificados na operação.

Para exemplificar o algoritmo de aninhamento será utilizada a relação abaixo, cujo esquema contém somente atributos atômicos:

EMP3:

	EMP#	E_NOME	E_IDADE	CARGO
2	2510	Rafael	10	operário
3	4910	Sandro	15	chefe
4	3613	Geraldo	3	gerente
5	2510	Beth	25	operário
6	4910	Ana	17	chefe

ESQ_EMP3:

	EMP#	E_NOME	E_IDADE	CARGO
2	-1	0	0	0
3	1	2	3	4
4	0	0	0	0
5	1	1	1	1
6	*1	*2	*3	*4

DADOS_EMP3:

1	11	2510
2	12	Rafael
3	13	10
4	14	operário
5	21	4910
6	22	Sandro
7	23	15
8	24	chefe
9	31	3613
10	32	Geraldo
11	33	3
12	34	gerente
13	41	2510
14	42	Beth
15	43	25
16	44	operário
17	51	4910
18	52	Ana
19	53	17
20	54	chefe

Na operação abaixo, os atributos EMP# e CARGO não estão especificados na operação, gerando, assim, o seguinte resultado:

EMP = *NEST* E_INFO = (E_NOME, E_IDADE) EMP3

EMP:

EMP#	E_INFO		CARGO
	E_NOME	E_IDADE	
2510	Rafael Beth	10 25	operário
4910	Sandro Ana	15 17	chefe
3613	Geraldo	3	gerente

O algoritmo supõe que os atributos aninhados pertençam ao mesmo nível. Ainda, se um dos atributos for componente de um atributo composto, os outros atributos a serem aninhados na operação *NEST* devem ser seus irmãos.

Deste modo, a operação abaixo é inválida:

NEST E_NOTA = (E_NOME, CARGO) EMP

O nome do atributo introduzido na operação também não deve ser igual a nenhum outro já existente na relação resultante, nem na anterior. Assim, é inválida a operação:

NEST E_INFO = (CARGO) EMP

ALGORITMO DE ANINHAMENTO

(a) *Gerar uma nova estrutura de armazenamento para o esquema resultante, diferindo da original pela inclusão da coluna referente ao atributo composto introduzido na operação NEST, uma posição antes do primeiro atributo componente do* aninhamento

especificado em *NEST*. As linhas 2-4 dessa nova coluna serão preenchidas, respectivamente, com os valores correspondentes ao número de elementos componentes (identificados na operação), a posição na composição (a mesma original do primeiro atributo componente) e com o valor 1 (referente à multivaloração). A quinta e a sexta linha conterão, por sua vez, valores iguais aos do nível e do identificador genérico de seu primeiro componente na tabela original. As demais colunas a partir dessa deverão ser modificadas somente nas linhas 3, 5 e 6, caso se enquadrem nas respectivas restrições:

linha 3: os **n** atributos componentes do aninhamento têm suas posições reordenadas de 1 até **n**, progressivamente; para os demais atributos do mesmo nível do novo atributo composto, suas posições originais são decrementadas de (**n** - 1).

linha 5: os atributos componentes têm seus níveis acrescidos de 1.

linha 6: os identificadores genéricos dos atributos componentes são acrescidos da marca *, seguida pelo valor da linha 3 da nova tabela. Os identificadores dos atributos seguintes aos componentes do aninhamento podem ter seus valores mais significativos (mais à esquerda) alterados, se os respectivos valores da linha 3 (posição na composição) sofreram alteração.

(b) Considerando os identificadores dos atributos não especificados na operação de aninhamento, fazer iterações sobre as suas marcas adequadamente (conforme a disposição ou nível dos atributos), de modo a encontrar todas as tuplas que tenham conjuntos de valores iguais correspondentes a esses atributos. A cada coleção de tuplas com tais conjuntos iguais, corresponde uma tupla da relação aninhada.

(c) Cada tupla da relação aninhada encontrada em (b) é armazenada na

estrutura de armazenamento de dados da nova relação, evitando-se repetição de valores dos atributos não aninhados. Os identificadores de todos os valores devem ser alterados seguindo a nova tabela.

Assim, no caso da operação feita anteriormente:

EMP = *NEST* E_INFO = (E_NOME, E_IDADE) EMP3,

obtém-se as seguintes estruturas de armazenamento referentes à relação resultante:

ESQ_EMP:

	EMP#	E_INFO	E_NOME	E_IDADE	CARGO
1					
2	-1	2	0	0	0
3	1	2	1	2	3
4	0	1	0	0	0
5	1	1	2	2	1
6	*100	*200	*2*1	*2*2	*300

DADOS_EMP:

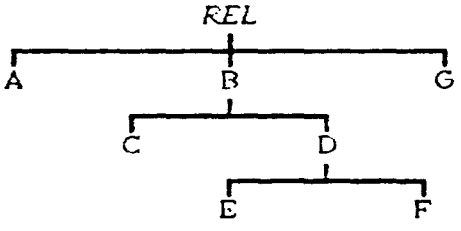
1	1100	2510
2	1211	Rafael
3	1212	10
4	1221	Beth
5	1222	25
6	1300	operário
7	2100	4910
8	2211	Sandro
9	2212	15
10	2221	Ana
11	2222	17
12	2300	chefe
13	3100	3613
14	3211	Geraldo
15	3221	3
16	3300	gerente

5.3. ALGORITMO DE DESANINHAMENTO *(UNNEST)*

Para ilustrar o algoritmo de desaninhamento será usado o seguinte exemplo:

$\overline{REL}$:

A	B			G
	C	D		
		E	F	
2510	10	16 20	7 1	9
4910	15	1 2	1 2	7



ESQ_REL:

1	A	B	C	D	E	F	G
2	-1	4	0	2	0	0	0
3	1	2	1	2	1	2	3
4	0	1	0	1	0	0	0
5	1	1	2	2	3	3	1
6	*10000	*20000	*2*100	*2*200	*2*2*1	*2*2*2	*30000

DADOS_REL:

1	110000	2510
2	121100	10
3	121211	16
4	121212	7
5	121221	20
6	121222	1
7	130000	9
8	210000	4910
9	221100	15
10	221211	1
11	221212	1
12	221221	2
13	221222	2
14	230000	7

O *ALGORITMO DE DESANINHAMENTO* supõe que os componentes do atributo a ser desaninhado não sejam compostos. Desse modo, é inválida a operação abaixo:

UNNEST B REL .

Para validar essa operação, deve-se antes desaninhar o atributo D.

O algoritmo segue [FiT83] e, portanto, não considera o desaninhamento total (*UNNEST**) automaticamente, mas isso pode ser obtido através de sucessivos *UNNESTs*. Assim,

*UNNEST** REL

deve ser transformado em:

REL1 = *UNNEST* D REL e
REL2 = *UNNEST* B REL1.

ALGORITMO DE DESANINHAMENTO

Seja **X** *o atributo a ser desaninhado tendo* **k** *como seu valor discriminante (valor diferente de zero à extrema direita do identificador).*

Sejam X_1, X_2, ..., X_n *os componentes (filhos) de* X, *respectivamente com os valores discriminantes 1, 2, ..., n.*

(a) *Gerar uma nova tabela de esquema para a nova relação excluindo-se* **X**. *Se* X_1, X_2, ..., X_n *ainda forem componentes de algum outro atributo (em nível mais alto), digamos* **P**, *deve-se decrementar de 1 o valor da linha 2 de* **P**. *Os campos dos atributos restantes devem ser adequadamente alterados nas linhas abaixo, segundo as respectivas normas:*

linha 3: a posição do primeiro atributo componente, X_1, *recebe o valor da posição referente ao atributo desaninhado,* **k**; *os demais componentes, bem como os atributos da relação de mesmo nível que o desaninhado e pertecentes a* **P**, *recebem o valor correspondente a uma progressão aritmética (P.A.) de razão 1 com valor inicial* **k** + 1.

linha 5: os níveis dos componentes de X são decrementados de 1.

linha 6: os novos identificadores genéricos de X_1, ..., X_n

são determinados conforme (i) e (ii).

(ii) X_1 recebe o identificador de X.

(iii) $X_2, \ldots, X_n$ recebem identificadores iguais ao de X_1, tendo seus respectivos valores discriminantes acrescidos de 1 em P.A.; em outras palavras, seus dígitos discriminantes receberão os valores k+1, ..., k+n-1, respectivamente.

(b) Determinar os novos identificadores genéricos dos atributos restantes do esquema, segundo a tabela de esquema gerada em (a). Isso deverá ocorrer se n > 1 (atributo desaninhado com mais de um componente) ou se os atributos componentes de X ainda pertencerem a algum outro atributo composto. Nesse caso, os atributos à direita de X_n e de mesmo nível têm seus identificadores alterados do seguinte modo:

(b1) se o atributo for simples, seu valor discriminante recebe o valor subseqüente na P.A. a partir de k+n-1;

(b2) se o atributo for composto, além do seu valor discriminante receber o próximo valor na P.A. a partir de (k + n - 1), essa mudança deve refletir nos atributos filhos.

(c) Os valores que integrarão a nova estrutura de dados serão recuperados da estrutura original iterando-se convenientemente as marcas dos identificadores. As iterações devem considerar as regras abaixo para a formação das novas tuplas:

(c1) se o atributo desaninhado X pertence ao primeiro nível ou não tem atributos irmãos, cada tupla da relação original gera no máximo w ≤ t tuplas na relação resultante, onde t é o número máximo de tuplas de X. Isso se dá porque os atributos distintos de $X_1 \ldots, X_n$ devem ser repetidos para cada subtupla desses atributos.

(c2) *se o atributo desaninhado X não pertence ao primeiro nível e tem outros atributos irmãos (no seu nível), cada tupla da nova relação é formada pela repetição dos valores dos atributos irmãos para cada subtupla do atributo decomposto. O restante da tupla original permanece inalterado.*

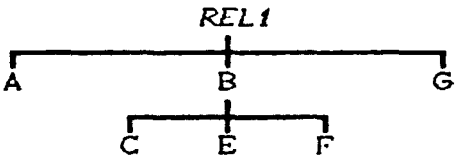
(d) *Para fins de construção da estrutura de dados, as tuplas obtidas em (c) deverão ter seus identificadores alterados convenientemente segundo a linha 6 da tabela de esquema de (a). Isso, na verdade, é feito simultaneamente com a recuperação dos valores ((c)), mantendo-se a ordenação original na nova estrutura.*

Assim, para a operação abaixo, tem-se o seguinte resultado:

REL1 = *UNNEST* D REL

REL1:

A	B			G
	C	E	F	
2510	10	16	7	9
	10	20	1	7
4910	15	1	1	7
	15	2	2	



ESQ_REL1:

	A	B	C	E	F	G
1						
2	-1	2	0	0	0	0
3	1	2	1	2	3	3
4	0	1	0	0	0	0
5	1	1	2	2	2	1
6	*100	*200	*2*1	*2*2	*2*3	*300

DADOS_REL1:

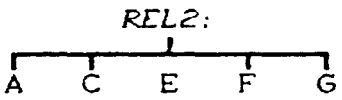
1	1100	2510
2	1211	10
3	1212	16
4	1213	7
5	1221	10
6	1222	20
7	1223	1
8	1300	9
9	2100	4910
10	2211	15
11	2212	1
12	2213	1
13	2221	15
14	2222	2
15	2223	2
16	2300	7

No caso da operação abaixo, obtém-se a relação seguinte:

REL2 = *UNNEST* B REL1

REL2:

A	C	E	F	G
2510	10	16	7	9
2510	10	20	1	9
4910	15	1	1	7
4910	15	2	2	7



As estruturas de armazenamento seriam, então, alteradas para:

ESQ_REL2:

	A	C	E	F	G
1					
2	-1	0	0	0	0
3	1	2	3	4	5
4	0	0	0	0	0
5	1	1	1	1	1
6	*1	*2	*3	*4	*5

DADOS_REL2:

1	11	2510
2	12	10
3	13	16
4	14	7
5	15	8
6	21	2510
7	22	10
8	23	20
9	24	1
10	25	8
11	31	4910
12	32	15
13	33	1
14	34	1
15	35	7
16	41	4910
17	42	15
18	43	2
19	44	2
20	45	7

Para determinados esquemas, a hipótese de chave simples deixa de ser válida após uma operação de *UNNEST*. Esta dissertação ignora estes casos, considerados na *seção 7.3.5*.

5.4. ALGORITMO DE INTERSECÇÃO *(INTER)*

O algoritmo de intersecção de duas relações supõe que ambas tenham a mesma árvore de esquema, com nomes dos atributos idênticos.

A intersecção de duas relações, que resulta numa nova relação contendo as tuplas comuns entre elas, requer comparações complexas de tuplas, devido aos diferentes conjuntos de valores de cada atributo multivalorado.

Um conjunto de valores é uma estrutura em que a posição dos elementos não é relevante. Uma lista de valores, por outro lado, é uma estrutura em que a disposição dos elementos é essencial.

Por convenção, representar-se-á os conjuntos por chaves, {...}, e as listas por colchetes, [...]. Assim, veja como fica a representação da tupla da relação abaixo:

A	B					C
	D	E	F			
			G	H		
				I	J	
0	1	3	5	6	7	1
				8	9	
			7	9	10	
	2	4	5	1	7	

tupla:

```
1 2     3
[ 0 { [ 1 3 { [ 5 { [ 6 7 ] [ 8 9 ] } ] }
                                  3 2
                [ 7 { [ 9 10 ] } ] } ]
4                                 4 1
[ 2 4 { [ 5 { [ 1 7 ] } ] } ] } ]
```

Para o algoritmo apresentado, duas tuplas de duas relações de mesmo esquema são consideradas *iguais* se todas as comparações de

seus valores forem bem sucedidas. Para isso, veja as regras abaixo:

(i) A comparação de dois *valores atômicos* é bem sucedida somente quando estes são iguais;

(ii) A comparação de duas *listas* é bem sucedida quando todos os elementos de uma têm correspondentes iguais em valor e posição na outra;

(iii) A comparação de dois *conjuntos* é bem sucedida quando ambos têm o mesmo número de elementos e quando cada elemento de um dos conjuntos tem seu correspondente em valor no segundo.

Desse modo, considere $\overline{R_1}$ e $\overline{R_2}$ abaixo:

$\overline{R_1}$:

B	C'
	C
1	1 2

$\overline{R_2}$:

B	C'
	C
1	2 3

A intersecção de suas tuplas é dada por:

$$[\ 1\ \langle [\ 1\ ]\ [\ 2\ ]\ \rangle\ ] \cap [\ 1\ \langle [\ 2\ ]\ [\ 3\ ]\ \rangle\ ] = \emptyset.$$

Por outro lado, se o atributo C' for desaninhado em ambas relações, o resultado da intersecção será a tupla [1 2].

A comparação tupla a tupla de duas relações, $\overline{R_1}$ e $\overline{R_2}$, merece muita atenção, pois é essencial não só à intersecção, mas às outras operações. Para isso, é apresentado a seguir o *ALGORITMO COMPARA_TUPLAS*. Dada uma tupla de $\overline{R_1}$ (caracterizada pelo identificador da chave), o algoritmo percorre as tuplas de $\overline{R2}$ até encontrar uma tupla igual, segundo as normas consideradas anteriormente (*comparação bem sucedida*) ou até o final de $\overline{R_2}$ (*comparação mal sucedida*). As tuplas de $\overline{R_2}$ que já foram comparadas com sucesso são excluídas das outras comparações.

No *Apêndice A*, o *ALGORITMO COMPARA_TUPLAS* é reapresentado com maiores detalhes.

ALGORITMO COMPARA_TUPLAS

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas estruturas de relação e* #1 *o identificador genérico do atributo chave de* $\overline{R_1}$. *Seja* t *uma tupla de* $\overline{R_1}$.

Procure em $\overline{R_2}$ *uma tupla* u *tal que os valores do atributo chave sejam iguais em* u *e* t. *Se* u *não for encontrada, a comparação é interrompida.*

Seja A *um atributo simples. Uma comparação é bem sucedida se* t[A] = u[A].

Percorra a árvore de esquema da seguinte forma:

(a) *Compare inicialmente todos os atributos simples de* t *e* u.

(b) *Para cada atributo composto* C *do esquema, execute recursivamente:*

(b1) *Compare inicialmente todos os atributos simples de* C.

(b2) *Compare os atributos compostos de* C, *usando COMPARA_TUPLAS.*

Observação: *para casos em que não há necessidade de comparação de tuplas completas (por exemplo, junção), este algoritmo pode ser simplificado, sendo que cada subtupla passa a ser tratada como se fosse uma tupla inteira.*

ALGORITMO DE INTERSECÇÃO

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas relações passíveis de intersecção.*

(a) *Criar tabela de esquema para o resultado.*

(b) *Enquanto houver tuplas* t *de* $\overline{R_1}$ *a comparar, faça:*

(b1) *Use COMPARA_TUPLAS para comparar* t *com as tuplas de* $\overline{R_2}$ *ainda não selecionadas.*

(b2) *Se encontrar* u $\in \overline{R_2}$ *tal que* u = t, t *pertence à intersecção. Elimine* u *das próximas iterações.*

(c) *Cada tupla comum encontrada em* (b) *é armazenada na nova estrutura de dados com as marcas dos identificadores alteradas convenientemente, refletindo a seqüencialidade e composição dos valores.*

Como exemplo, suponha $\overline{R_1}$ e $\overline{R_2}$ definidas abaixo:

$\overline{R_1}$:

A	B'
	B
1	1 2
0	1

$\overline{R_2}$:

A	B'
	B
3	2
1	2 1

ESQ_R1 ≈ *ESQ_R2*:

	A	B'	B
1			
2	-1	1	0
3	1	2	1
4	0	1	0
5	1	1	2
6	*100	*200	*?*1

DADOS_R1:

1	1100	1
2	1211	1
3	1221	2
4	2100	0
5	2211	1

DADOS_R2:

1	1100	3
2	1211	2
3	2100	1
4	2211	2
5	2221	1

Considerando, então, a operação

INTER $\overline{R_1}$ $\overline{R_2}$,

a primeira instância do algoritmo indica a tupla de $\overline{R_1}$ de chave *1*, que passa a ser o argumento do *ALGORITMO COMPARA_TUPLAS*.

Desse modo, procura-se uma tupla u de igual chave em $\overline{R_2}$, econtrando-se a identificada por *2100*.

Como os esquemas só possuem mais um atributo e este é composto, toma-se o identificador do seu atributo componente simples: **2*1*. Comparam-se, então, os valores de t[*1211*] com u[*2211*] e u[*2221*], sendo encontrado t[*1211*] = u[*2221*]. Segue-se com a comparação de t[*1221*] com u[*2211*], que são iguais. Além dessas comparações serem bem sucedidas, o número de elementos de B em t e u são iguais (de valor dois). Assim, as tuplas t e u são consideradas comuns.

Como a próxima tupla de $\overline{R_1}$ não tem chave de mesmo valor em $\overline{R_2}$, a relação resultante é

A	B'
	B
1	1 2

5.5. ALGORITMO DE DIFERENÇA *(DIF)*

A seguir é apresentado o *ALGORITMO DE DIFERENÇA*, ilustrado por um exemplo.

ALGORITMO DE DIFERENÇA

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas relações de mesmo esquema R.*

Seja D *o resultado da diferença entre as duas relações acima e* d *uma tupla de* D.

(a) *Construir a tabela ESQ_D, cujo conteúdo é idêntico ao de ESQ_*R_1 *e ESQ_*R_2.

(b) *Preencher DADOS_D à medida que se encontrem tuplas que pertençam à diferença, dado que* $d \in D$ *se* $d \in \overline{R_1}$ *e* $d \notin \overline{R_2}$. *A verificação é feita usando-se o ALGORITMO COMPARA_TUPLAS descrito na seção anterior.*

Suponhamos $\overline{R_1}$ e $\overline{R_2}$ dadas como segue:

$\overline{R_1}$:

A	B	C
		C'
0	1	1 2
0	1	1 3
2	3	1 4

$\overline{R_2}$:

A	B	C
		C'
0	1	1 3
0	1	1
2	3	1 5

DADOS_R1:

1	1100	0
2	1200	1
3	1311	1
4	1321	2
5	2100	0
6	2200	1
7	2311	1
8	2321	3
9	3100	2
10	3200	3
11	3311	1
12	3321	4

DADOS_R2:

1	1100	0
2	1200	1
3	1311	1
4	1321	3
5	2100	0
6	2200	1
7	2311	1
8	3100	2
9	3200	3
10	3311	1
11	3321	5

Deste modo, para a operação

$$\overline{R3} = DIF\ \overline{R1}\ \overline{R2},$$

a seqüência de passos do algoritmo dado implicaria nos seguintes resultados:

Primeiramente, seria determinado o identificador da chave de $\overline{R1}$, **1*, com a marca "*" podendo assumir valores de 1 a 3. Para cada um destes valores, executa-se o *ALGORITMO COMPARA_TUPLAS* (*seção 5.3*). Inicialmente, considere t uma tupla de $\overline{R1}$ e u uma tupla de $\overline{R2}$. Primeiramente, comparam-se t[*11*] com u[*1*], que são iguais bem como t[*12*] e u[*12*], mas os conjuntos de valores do atributo C em $\overline{R1}$ não coincide com o de $\overline{R2}$. Comparam-se, então, t[*11*] com t[*21*]. Embora estes valores coincidam, os atributos restantes não têm valores comuns. A comparação segue com t[*11*] e t[*31*], que é mal sucedida. Assim, como a tupla t não temm correspondente em $\overline{R2}$, ela não pertence ao resultado da diferença.

Seguindo o raciocínio exposto acima para as tuplas restantes de $\overline{R1}$, temos:

$\overline{R_3}$:

A	B	C
		C'
0	1	1 2
2	3	1 4

5.6. ALGORITMO DE UNIÃO *(UNIÃO)*

O algoritmo de união supõe que as duas relações envolvidas tenham o mesmo esquema. A sua execução resume-se em criar uma nova relação de mesmo esquema que "concatena" o conteúdo das duas relações.

ALGORITMO DE UNIÃO

Sejam $\overline{R_1}$ e $\overline{R_2}$ duas relações sob esquemas idênticos e $\overline{R}$ o resultado da união dessas duas relações.

(a) *Construir a tabela ESQ_R, cujo conteúdo é igual ao de ESQ_R1 e ESQ_R2.*

(b) *Em DADOS_R são colocados todos os valores contidos em DADOS_R1, mantendo-se os identificadores.*

(c) *Junte a estes dados o resultado da diferença $\overline{R_2}$ - $\overline{R_1}$, usando o ALGORITMO DE DIFERENÇA (secção 5.5), mantendo-se a seqüencia dos identificadores.*

Deste modo, para as relações $\overline{R_1}$ e $\overline{R_2}$ abaixo, tem-se $\overline{R_3}$ como resultado final da operação de *união*.

$\overline{R_1}$:

A	B
	B'
0	1 2
1	2

$\overline{R_2}$:

A	B
	B'
0	2 3
1	2
3	4

ESQ_R1 ≈ *ESQ_R2*:

	A	B	B'
2	-1	1	0
3	1	2	1
4	0	1	0
5	1	1	2
6	*100	*200	*2*1

DADOS_R1:

1	1100	0
2	1211	1
3	1221	2
4	2100	1
5	2211	2

DADOS_R2:

1	1100	0
2	1211	2
3	1221	3
4	2100	1
5	2211	2
6	3100	3
7	3211	4

$\overline{R}$ = *UNIÃO* $\overline{R_1}$ $\overline{R_2}$:

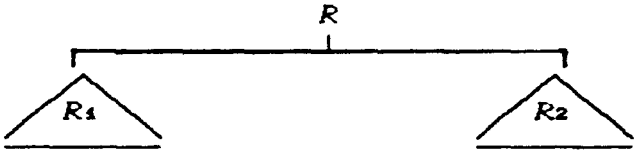
A	B
	B'
0	1 2
1	2
0	2 3
3	4

DADOS_R:

1	1100	0
2	1211	1
3	1221	2
4	2100	1
5	2211	2
6	3100	0
7	3211	2
8	3221	3
9	4100	3
10	4211	4

5.7. ALGORITMO DO PRODUTO CARTESIANO *(TIMES)*

Para efeitos do algoritmo abaixo, supõe-se que os esquemas das duas relações, $\overline{R_1}$ e $\overline{R_2}$, envolvidas na operação não contenham nenhum atributo em comum. O esquema R da relação resultante é dado pela concatenação das árvores de esquema de $\overline{R_1}$ e $\overline{R_2}$, ou seja,



Cada tupla de $\overline{R_1}$ é concatenada com todas as tuplas de $\overline{R_2}$. Desse modo, considerando que $\overline{R_1}$ tenha m tuplas e $\overline{R_2}$, n, R contém (m x n) tuplas.

ALGORITMO DO PRODUTO CARTESIANO

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas relações tal que* $R_1 \cap R_2 = \emptyset$ *e* $\overline{R}$ *a relação resultante do produto cartesiano dessas duas relações.*

(a) *Construir ESQ_R concatenando ESQ_*R_1 *e ESQ_*R_2*, com as seguintes alterações:*

linha 2: a posição correspondente ao atributo chave de $\overline{R_2}$ *passa a ser 0 (zero).*

linha 3: todas as colunas referentes aos atributos atômicos e compostos do primeiro nível de $\overline{R_2}$ *são modificadas de modo a continuar a seqüência das posições dos atributos de* $\overline{R_1}$*, segundo uma P.A. de razão 1.*

linha 6: os identificadores dos atributos de $\overline{R_2}$ *têm seus valores mais significativos alterados segundo a linha 3 da nova tabela de esquema.*

(b) *Construir DADOS_R, considerando que os identificadores dos valores em DADOS_*R_1 *e DADOS_*R_2 *devem ser alterados segundo ESQ_R, de modo a garantir a seqüencialidade e ordem das tuplas de* $\overline{R}$*. Assim, para cada tupla* t *de* $\overline{R_1}$*, executam-se os seguintes subpassos:*

(b1) *Armazenam-se os valores de* t *em DADOS_R seguindo a disposição em DADOS_*R_1*, alterando-se as marcas mais significativas dos identificadores convenientemente.*

(b2) *Sendo* u *uma tupla de* $\overline{R_2}$*, armazenam-se seus valores em DADOS_R, obedecendo à ordem de DADOS_*R_2 *e a nova forma genérica dos identificadores (linha 6 de ESQ_R). As marcas mais significativas são alteradas convenientemente.*

(b3) *Repete-se os subpassos (b1) e (b2) acima até o final das tuplas de* $\overline{R_2}$*, onde, a cada instância,* u *passa a representar a próxima tupla de* $\overline{R_2}$ *(inicialmente* u *representa a primeira tupla de* $\overline{R_2}$*).*

5.8. ALGORITMO DE JUNÇÃO NATURAL *(JOIN)*

O resultado do algoritmo apresentado depende dos dois esquemas envolvidos, digamos R_1 e R_2, seguindo uma das regras abaixo:

(i) Se não há atributos comuns entre R_1 e R_2, ou seja, $E_{R1} \cap E_{R2} = \emptyset$, o resultado é dado pelo *PRODUTO CARTESIANO (seção 5.2.8)* das respectivas relações. O esquema resultante é a união dos esquemas R_1 e R_2.

(ii) Se os esquemas possuem um subesquema comum $R_1' = R_2' = R$, a relação resultante do algoritmo será formada pelas subtuplas comuns a $\overline{R_1'}$ e a $\overline{R_2'}$, compostas com os correspondentes valores dos atributos restantes de $\overline{R_1}$ e $\overline{R_2}$. O esquema resultante é a união de R_1 com $R_2 - R$.

Para a obtenção do resultado dado em (ii), devem ser feitas comparações entre as tuplas das sub-relações $\overline{R_1'}$ e $\overline{R_2'}$ de esquema R, respectivamente contidas em $\overline{R_1}$ e $\overline{R_2}$. Para isso, o *ALGORITMO COMPARA_TUPLAS*, apresentado na *seção 5.4*, deve ser ligeiramente adaptado, passando a comparar subtuplas (correspondentes às tuplas da intersecção).

ALGORITMO DE JUNÇÃO NATURAL

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas relações quaisquer e* $\overline{R}$ *a relação resultante da junção de intersecção dessas duas relações.*

Seja COMUM a subárvore comum às árvores de esquema de $\overline{R_1}$ *e* $\overline{R_2}$.

Sejam $DIST_1 = E_{R1} - COMUM$ *e* $DIST_2 = E_{R2} - COMUM$.

(a) *Se COMUM é <u>vazio</u>, executa-se o ALGORITMO DO PRODUTO CARTESIANO (seção 5.2.8) sobre* $\overline{R_1}$ *e* $\overline{R_2}$ *e ignora-se os passos seguintes. Se COMUM é <u>distinto</u> <u>de</u> <u>vazio</u>, segue-se normalmente com os passos*

abaixo.

(b) Construir ESQ_R tal que suas colunas sejam correspondentes aos atributos dados por $R_1 \cup DIST_2$. Os campos dessa nova estrutura mantêm seus valores anteriores, com exceção das seguintes alterações:

linha 2: a coluna referente à chave de $\overline{R_2}$ passa a ser 0, não se admitindo, portanto, a composição das chaves de $\overline{R_1}$ e $\overline{R_2}$.

linha 3: as colunas referentes aos atributos de nível 1 de $DIST_2$ devem ter seus valores alterados, de modo a dar continuidade à P.A. de razão 1 em relação aos atributos de nível 1 de R_1.

linha 6: os identificadores dos elementos de DIST2 têm seus valores mais significativos alterados segundo os valores da linha 3 dos atributos de nível 1 da nova tabela de esquema.

(c) Enquanto houver subtuplas t em $\overline{R_1}$ faça:

(c1) Considere uma tupla u de $\overline{R_2}$ (inicialmente, a primeira).

(c2) Use o ALGORITMO COMPARA_TUPLAS (seção 5.4) para comparar t com u, ignorando a existência de chaves.

(c3) Se a comparação foi bem sucedida, ou seja, t[COMUM] = u[COMUM], armazene em DADOS_R os valores de $DADOS\_R_1$, correspondentes a t, em ordem, seguidos dos valores de $DADOS\_R_2$ referentes a u[$DIST_2$]. Os valores armazenados em DADOS_R, que constituem uma tupla de $\overline{R}$, devem ter seus identificadores alterados convenientemente, refletindo a seqüencialidade e composição dos valores. Elimine u das próximas iterações.

Para exemplificar o algoritmo acima, considere as relações abaixo:

$\overline{R_1}$:

A	C	D				J
		E	F		G	
			H	I		
1	2	1	2	4	9	3
			1	8		
		6	7	8	10	

$\overline{R_2}$:

B	D				C
	E	F		G	
		H	I		
2	6	7	8	10	2
	1	2	4	9	
		1	8		

DADOS_R1:

1	110000	1
2	120000	2
3	131100	1
4	131211	2
5	131212	4
6	131221	1
7	131222	8
8	131300	9
9	132100	6
10	132211	7
11	132212	8
12	132300	10
13	140000	3

DADOS_R2:

1	110000	2
2	121100	6
3	121211	7
4	121212	8
5	121300	10
6	122100	1
7	122211	2
8	122212	4
9	122121	1
10	122122	8
11	122300	9
12	130000	2

No exemplo acima, COMUM é formado pelos atributos de nível 1 C e D (seguindo a ordem de R_1), sendo o primeiro simples e o segundo composto. D obedece a restrição de conter pelo menos um atributo simples.

Sendo t a única tupla de $\overline{R_1}$ e u a tupla de $\overline{R_2}$, compara-se,

inicialmente, o valor de C, ou seja, t[*120000*] com u[*130000*]. Como tal comparação referente ao único atributo simples de COMUM foi bem sucedida, passa-se a comparar os valores de D.

Compara-se, então, os valores de E t[*131100*] e u[*121100*]. São distintos, mas t[*131100*] e u[*122100*] são iguais. Para esta subtupla, os valores de G são iguais: t[*131300*] e u[*122300*].

Para os valores do atributo F, as seguintes comparações também são bem sucedidas: t[*131211*] = u[*122111*], t[*131212*] = u[*122212*], t[*131222*] = u[*122121*] e t[*131222*] = u[*122122*].

Prosseguindo-se a comparação com os valores de D, tem-se a seguinte relação resultante:

A	C	D				J	B
		E	F		G		
			H	I			
1	2	1	2	4	9	3	2
			1	8			
		6	7	8	10		

5.9. ALGORITMO DE SELEÇÃO *(SELECT)*

O algoritmo de seleção supõe que sempre será recuperada a tupla inteira correspondente à condição da seleção, aplicado de forma semelhante ao MRN. Assim, seleção sobre atributos simples é feita comparando-se cada valor do atributo e sobre atributos compostos, é realizada por comparação de listas.

Considere a seguinte relação:

$\overline{EMP}$:

EMP#	E_INFO		CARGO
	E_NOME	E_IDADE	
2510	Rafael Beth	10 25	operário
4910	Sandro Ana	15 17	chefe
3613	Geraldo	15	gerente

A operação

SELECT (E_IDADE = 10) EMP

tem como resultado a tupla de chave *2510*.

No caso de

SELECT (E_IDADE > 10) EMP

o resultado será composto de todas as tuplas da relação $\overline{EMP}$.

Quando a condição de seleção recair sobre um atributo multivalorado, que possui um conjunto de valores para cada tupla, pode-se usar o operador de inclusão *in*, como no exemplo abaixo:

SELECT ([E_NOME = Sandro, E_IDADE = 15] *in* E_INFO) EMP

ou, simplesmente,

SELECT ([Sandro, 15] *in* E_INFO) EMP,

pois todos os atributos componentes de E_INFO são especificados, seguindo a ordem de composição. A tupla resultante dessa operação é a identificada por 4910.

Nem todos os atributos componentes de um atributo composto precisam ser especificados. Nesse caso, ignoram-se os valores dos atributos omitidos na seleção das tuplas.

Quando a condição de seleção envolve conjunto de valores,

poder-se-ia ter uma representação adicional exemplificada pela operação abaixo:

SELECT ([E_IDADE = <17, 15>] *in* E_INFO) EMP,

que seleciona a tupla dada por 4910.

Se ao invés de "<...>", que especifica um campo de um atributo multivalorado, tivéssemos "{...}", o resultado seria formado pelas tuplas com 17 e/ou 15, a saber, as identificadas por 2510 e 4910.

Esse tipo de extensão para conjuntos de valores torna-se muito complexo quando envolve mais de um componente de um atributo composto.

ALGORITMO DE SELEÇÃO

Sejam $\overline{R_1}$ uma relação aninhada e $\overline{R}$ a relação resultante de uma operação de seleção sobre $\overline{R_1}$.

(a) *Construir ESQ_R, cujo conteúdo será idêntico ao de ESQ_R₁.*

(b) *Considere os identificadores genéricos (linha 6 de ESQ_R₁) de todos os atributos que compoem a condição de seleção.*

  (b₁) *Para cada marca desses identificadores, fazer uma iteração buscando na estrutura de dados os valores que satisfazem à condição.*

  (b₂) *Para cada conjunto de valores que satisfaz o conjunto de condições, selecionar a tupla correspondente e armazená-la em DADOS_R, alterando os dígitos mais à esquerda de modo a manter a ordem das tuplas.*

  (b₃) *Se nenhuma tupla for selecionada, o resultado da seleção será vazio.*

Para exemplificar o algoritmo acima, considere a relação $\overline{EMP}$, dada anteriormente, com as correspondentes estruturas de armazenamento de esquema e de dados apresentadas a seguir:

ESQ_EMP:

1	EMP#	E_INFO	E_NOME	E_IDADE	CARGO
2	-1	2	0	0	0
3	1	2	1	2	3
4	0	1	0	0	0
5	1	1	2	2	1
6	*100	*200	*2*1	*2*2	*300

DADOS_EMP:

1	1100	2510
2	1211	Rafael
3	1212	10
4	1221	Beth
5	1222	25
6	1300	operário
7	2100	4910
8	2211	Sandro
9	2212	15
10	2221	Ana
11	2222	17
12	2300	chefe
13	3100	3613
14	3211	Geraldo
15	3212	15
16	3300	gerente

Assim,

SELECT (EMP# > 1800) EMP

busca todos os valores do atributo de identificador *#100* com valor maior que *1800*, variando a marca para assumir valores de 1 a 3.

5.10. ALGORITMO DE PROJEÇÃO (*PROJECT*)

O algoritmo de projeção apresentado retorna todas as subtuplas da relação original correspondentes aos atributos especificados, eliminando as duplicadas.

O algoritmo supõe que a chave sempre esteja entre os atributos a serem projetados, evitando problemas para o caso desta relação ser usada posteriormente. Nada impede, no entanto, que se exclua a chave do resultado, desde que a relação não seja usada para operações futuras.

Considere a relação $\overline{EMP}$ da *seção 5.9*. A operação

PROJECT (EMP#, E_NOME, CARGO) EMP

seria transformada em

PROJECT (EMP#, E_INFO/E_NOME, CARGO) EMP

onde a barra "/" indica que apenas o atributo E_NOME de E_INFO é considerado.

Assim, o resultado da consulta acima seria:

EMP#	E_INFO	CARGO
	E_NOME	
2510	Rafael Beth	operário
4910	Sandro Ana	chefe
3613	Geraldo	gerente

ALGORITMO DE PROJEÇÃO

Seja $\overline{R_1}$ *uma relação aninhada com* m *tuplas e* $\overline{R}$ *a relação resultante de uma operação de projeção sobre* $\overline{R_1}$, *envolvendo atributos* $X = (X_1, \ldots, X_n)$ *de* R_1, *onde* X_1 *é chave.*

(a) *Gerar ESQ_R, que será composta pelos atributos* $X_1, \ldots, X_n$. *Para a linha 6 os identificadores dos atributos X de* R_1 *devem ter seus valores discriminantes alterados convenientemente.*

(b) Considerar *os identificadores genéricos dos atributos de X de* $\overline{R_1}$. *Para cada marca fazer uma iteração, sendo que as iterações das marcas mais à esquerda devem conter as das marcas mais à direita (ordem de aninhamento das iterações). Quando houver mais de um atributo envolvido na operação, as marcas correspondentes à mesma posição e ao mesmo nível são iteradas num mesmo laço.*

(d) *Para cada conjunto de valores* C *encontrado, verificar se um conjunto igual já pertence ao resultado. Acrescentar* C *a DADOS_R se já não existir subtupla igual. Para testar igualdade, usar o ALGORITMO COMPARA_TUPLAS (seção 5.4), adaptado para comparar subtuplas. Para a construção de DADOS_R, o conjunto de valores encontrados deve obedecer à ordem em que estão dispostos*

os atributos na operação.

Assim, a operação

PROJECT (EMP#, CARGO) EMP

determina os identificadores de esquema **100* e **300*, correspondendo aos atributos EMP# e CARGO. A marca deve assumir valores de 1 a 3, seguindo a ordem (1100, 1300), (2100, 2300), (3100, 3300).

Desse modo, o resultado dessa operação é dado por:

$\overline{EMP2}$:

EMP#	CARGO
2510	operário
4910	chefe
3613	gerente

ESQ_EMP2:

	EMP#	CARGO
1		
2	-1	0
3	1	2
4	0	0
5	1	1
6	*1	*2

DADOS_EMP2:

1	11	2510
2	12	operário
3	21	4910
4	22	chefe
5	31	3613
6	32	gerente

Seja, agora:

PROJECT (EMP#, E_INFO) EMP.

Sabendo-se que o identificador de esquema de EMP# é **100* e que E_INFO é composto por dois outros atributos, E_NOME e E_IDADE, com os identificadores de esquema **2*1* e **2*2*, respectivamente, quando a primeira marca gerar iteração com valor 1, deve-se procurar *1100* e *12*1*, juntamente com *12*2*, variando-se a segunda marca de 1 a 2.

6. IMPLEMENTAÇÃO DO SISTEMA PROPOSTO

6.1. INTRODUÇÃO

Neste capítulo são descritas as estruturas de dados escolhidas para a implementação do sistema proposto no *capítulo 4*. A seleção dessas estruturas objetivou manter a *clareza* dos níveis conceitual e lógico do MRNN também no nível físico.

As estruturas selecionadas têm por objetivo apenas validar os algoritmos propostos do *capítulo 5* sobre as estruturas lógicas do *capítulo 4*. Não há, portanto, preocupações com o desempenho ou otimização de espaço ou tempo de acesso.

As estruturas foram todas implementadadas em memória. Escolheu-se listas ligadas, estruturas dinâmicas, como base para maior facilidade de modificação e reorganização. Para a implementação em disco, outras considerações devem ser feitas, como, por exemplo, localidade ou facilidade de modificação de esquema. Neste caso, pode-se utilizar a estrutura proposta por Setzer [Set86], descrita brevemente na *seção 7.3.2*.

As estruturas selecionadas permitem que o sistema trate uniformemente relações aninhadas e relações planas. As operações da álgebra escolhida podem ser transformadas em seqüências de busca, inserção, eliminação e modificação de valores e ponteiros das estruturas internas.

Estas estruturas e as operações correspondentes foram implementadas em um ambiente do tipo PC/DOS em linguagem Pascal, visando validar as propostas dos capítulos 3 a 5.

O armazenamento das estruturas é feito com um encadeamento simples de registros, cujo nó cabeça-de-lista referencia o registro

correspondente à primeira relação inserida no sistema.

A cada estrutura de relação $\overline{R} = (R, r)$ corresponde um registro da forma:

<prox_rel>	<nome_rel>	<esq_rel>	<dados_rel>

onde:

<prox_rel>: *aponta* a próxima estrutura de relação armazenada;

<nome_rel>: contém o nome da relação;

<esq_rel>: *aponta* para a estrutura de armazenamento do esquema da relação: *ESQ_R*;

<dados_rel>: *aponta* para a estrutura de armazenamento dos dados da relação: *DADOS_R*.

As estruturas para a implementação de *ESQ_R* e *DADOS_R* são descritas nas seções subseqüentes. Para exemplificar tais estruturas, considere a relação $\overline{PROJ}$ *abaixo (já apresentada no capítulo 4)* com suas tabelas básicas de armazenamento:

$\overline{PROJ}$:

DEPTO	PROJETOS				EQPTO	
	PNUM	PNOME	PMEMBROS MNOMES	PVERBA	EQUANT	ETIPO
Bio	B608	FXP	Carlos Lauro	680.000	5 20	PC/AT PC/XT
	B700	Neve	Jairo Lauro	700.000	1	386/SX
Mat	M101	Mat1	Cesar Paulo Decio	104.000	2 1	PC/XT PC/AT

6.2. IMPLEMENTAÇÃO DA TABELA DE ESQUEMA

As colunas da tabela de esquema (*ESQ_R*) de uma relação são armazenadas sob a forma de registros encadeados. Tais registros são definidos como segue:

<prox_col>	<nom_atr>	<tip_atr>	<pos_atr>	<mult>	<niv_atr>	<id_atr>

onde:

<prox_col>: *aponta* para a próxima coluna de *ESQ_R*;

<nom_atr>: contém o nome do atributo correspondente;

<tip_atr>: contém um valor ≥ 0 correspondente ao tipo do atributo quanto a sua composição (simples/composto) ou um valor < 0 para um atributo chave (linha 2 de *ESQ_R*);

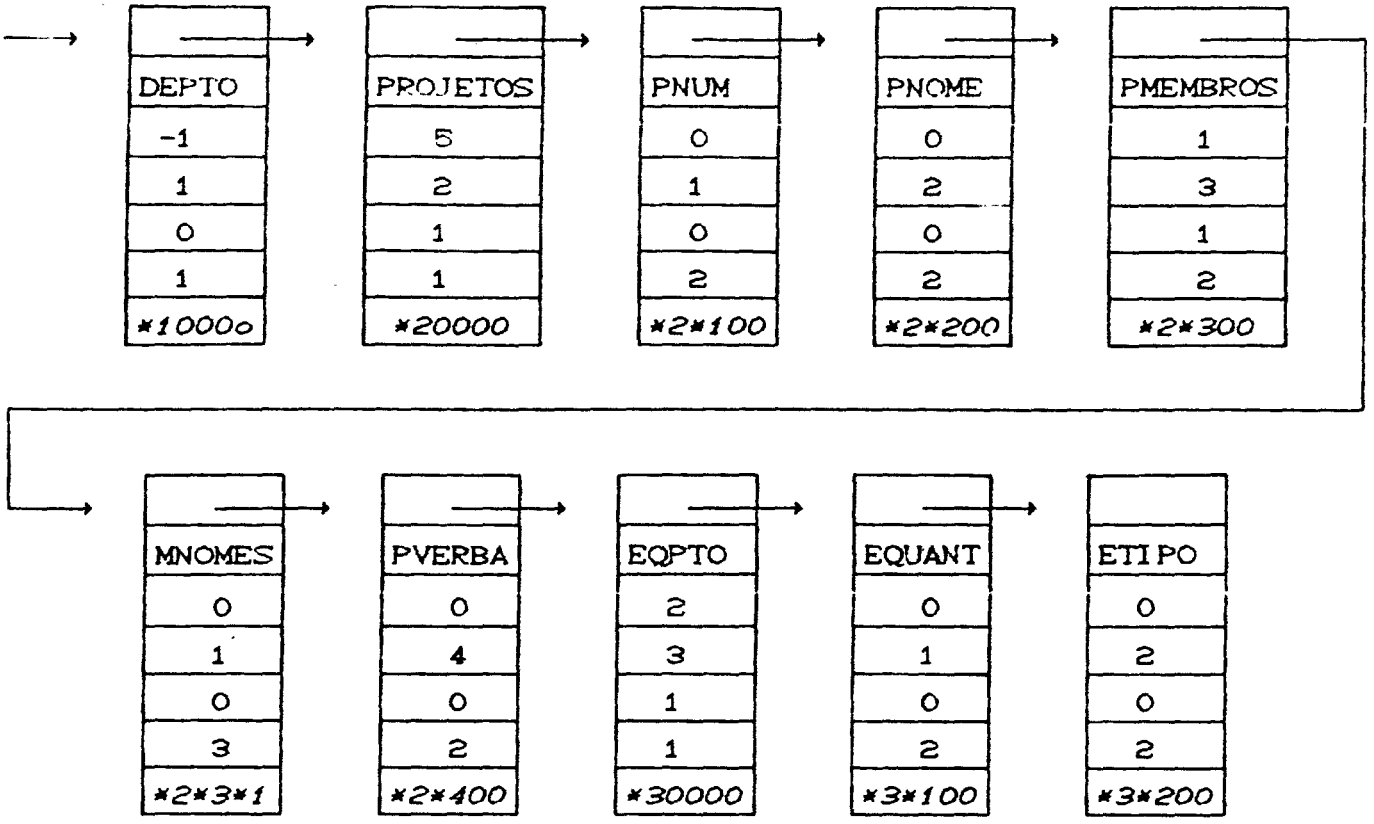
<pos_atr>: contém um valor inteiro referente à posição do atributo em relação aos seus irmãos (linha 3);

<mult>: campo do tipo *booleano*, cujo valor é *verdadeiro* ("1") se o atributo for multivalorado e *falso* ("0") se o atributo for monovalorado (linha 4);

<niv_atr>: indica o nível em que está o atributo em relação ao grafo do esquema (linha 5).

<id_atr>: contém o identificador genérico do atributo (linha 6).

Veja como ficaria a implementação de *ESQ_PROJ* (*seção 4.2*):



6.3. IMPLEMENTAÇÃO DA TABELA DE DADOS

A estrutura de dados apresentada a seguir suporta o armazenamento dos valores de uma relação, além das operações de busca, eliminação e modificação de dados.

A estrutura proposta consiste numa lista duplamente ligada, cujos nós podem referenciar nomes de atributos ou valores de atributos. Todos os seus nós são registros da forma:

<num_tup>	<tid>	<val_arm>	<ap_tup>	<ap_ant>	<ap_atr>

onde seus campos são dados por:

<num_tup>: indica o número de tuplas/subtuplas que tem o

atributo. Seu valor só é significativo em nós que referenciam nomes de atributos.

<tid>: contém o identificador do valor da relação. Logo, seu valor não tem sentido em nós referentes a nomes de atributos.

<val_arm>: contém um valor da relação. Caso o registro se refira a um nome de atributo, esse campo é vazio.

<ap_tup>: aponta para a próxima tupla/subtupla da relação, no caso do valor armazenado ser um dado significativo. Quando isso não acontece, ou seja, *<val_arm>* é vazio, aponta a primeira tupla/subtupla do atributo composto em questão.

<ap_ant>: aponta para o nó anterior do encadeamento.

<ap_atr>: aponta para o nó que pertence ao atributo irmão (mesmo nível e mesmo pai) imediatamente à direita no grafo do esquema. Se esse atributo irmão for simples, o nó apontado conterá um valor significativo da relação; se for composto, o nó terá *<val_arm>* vazio, pois não conterá nenhum dado.

Resumidamente, podemos ver no quadro abaixo, os campos que contêm valores significativos em cada um desses tipos de registro. Poderia-se pensar em também se ter dois tipos de registros ao nível físico, já que alguns campos ficam vazios em cada um deles. Embora isso minimize o uso da memória disponível, pode dificultar certas consultas, como será visto ainda nesta seção.

	<num_tup>	<tid>	<val_arm>	<ap_tup>	<ap_ant>	<ap_atr>
registro indicador de atributo composto	X	X		X	X	X
registro indicador de dado		X	X	X	X	X

A estrutura de lista proposta contém um nó cabeça-de-lista, segundo a forma geral dada acima, cujo campo *<tid>* contém o valor *01* seguido de *(kx2-2)* zeros, onde *k* é o nível do grafo do esquema. O campo *<num_tup>* conterá o número de tuplas da relação; inicialmente seu valor é *zero*.

Para que seja possível a compreensão da estrutura, bem como dos algoritmos aqui apresentados, é fundamental que se entenda a sua relação com os identificadores.

Suponha, então, o identificador de um certo dado **X**, representado por

$$a_1b_1a_2b_2 \ldots a_nb_n,$$

onde os elementos a_i referem-se às tuplas/subtuplas e os b_i aos atributos.

Para se encontrar **X**, o percurso na estrutura de dados deve seguir os pares de valores indicados no identificador, da esquerda para a direita (ignorando-se os zeros), segundo os seguintes critérios:

(i) Desloca-se "verticalmente" na estrutura, a_i passos, de acordo com o *i* em questão, ou seja, a partir do registro atual (último registro acessado), avança-se a_i vezes os campos *<ap_tup>* dos próximos registros. O último registro indicado passa, então, a ser o atual.

(ii) Desloca-se "horizontalmente" $(b_i - 1)$ passos, de acordo com o *i* em questão (o mesmo de *(i)*), ou seja, a partir

do registro atual, avança-se (bi - 1) vezes os campos <*ap_atr*> dos próximos atributos.

Como exemplo, suponhamos o valor Neve de $\overline{PROJ}$ (*seção 4.1*), cujo identificador é 122200. Para se encontrar esse valor na estrutura de armazenamenmto dos dados, deveria-se seguir os seguintes passos:

(1) avançar 1 campo <*ap_tup*> a partir do nó cabeça-de-lista;

(2) avançar 1 campo <*ap_atr*> a partir do último registro indicado;

(3) avançar 2 campos <*ap_tup*> a partir do último registro apontado;

(4) avançar 1 campo <*ap_atr*> a partir do último registro referenciado.

6.3.1. Geração da Estrutura de Armazenamento dos Dados

A geração e a manutenção dessa estrutura é feita dinamicamente. Para se conservar a hierarquia de composição dos atributos, são usados dois tipos de registros ao nível lógico, conforme mencionado anteriormente em *6.3*.

Uma relação vazia, isto é, sem dados, é aquela que contém somente o nó cabeça-de-lista.

Apresentamos no apêndice B um algoritmo para a geração da estrutura de armazenamento de dados de uma relação.

Esse algoritmo também é usado para se *inserir* dados na relação. No caso de algum atributo do esquema não possuir valor definido, utiliza-se um valor nulo, digamos λ, para que a estrutura do atributo seja criada. Desse modo, a estratégia de percurso permanece inalterada e a estrutura suportará inserções de subtuplas (se atributo composto) e modificação de valores (se atributo simples). Deve-se ressaltar que no MRN não é possível a inserção de subtuplas.

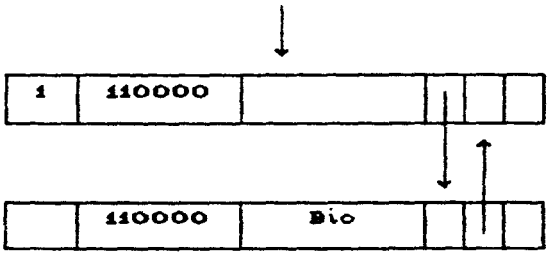
O algoritmo sempre inserirá tuplas/subtuplas (completas) nas

últimas posições convenientes da estrutura de armazenamento, o que corresponderá à última tupla/subtupla na relação.

Convém lembrar que a ordem de inserção dos dados é fundamental. Os valores são armazenados da esquerda para a direita, dando prioridade aos elementos das listas e dos conjuntos de valores de cada atributo composto.

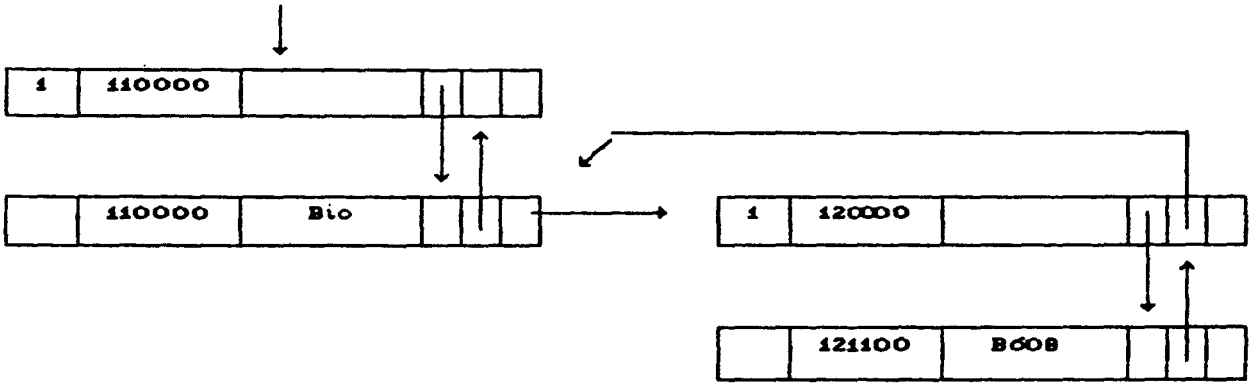
Para exemplificar o algoritmo, tome a relação $\overline{PROJ}$. A estrutura *DADOS_PROJ*, contém, inicialmente, o nó cabeça-de-lista.

O primeiro valor **X** a ser inserido é Bio, pertencente ao atributo DEPTO, cujo identificador é **10000*. Por ser o primeiro valor de DEPTO, o identificador de Bio é *110000* e *DADOS_PROJ* passa a ter a seguinte representação:



O segundo valor a ser armazenado é B608, que pertence ao atributo PNUM, cujo identificador é **2*100*. O caracter "2" do identificador indica que o registro contendo B608, digamos **R**, deve estar à direita do registro que contém Bio, ou seja, referenciado por *<ap_atr>* desse último registro. Como PNUM pertence ao atributo PROJETOS, inclui-se um registro **REG** precedendo **R**, cuja finalidade é criar o caminho até as subtuplas de PROJETOS. Por se tratar de um registro referente ao nome de um atributo que conterá valores da primeira tupla de $\overline{PROJ}$, **REG** contém o identificador *120000* e seu campo *<ap_tup>* é que referenciará **R**. Desse modo, obtém-se a seguinte estrutura:

primeira tupla de $\overline{\text{PROJ}}$, REG contém o identificador *120000* e seu campo *<ap_tup>* é que referenciará R. Desse modo, obtém-se a seguinte estrutura:

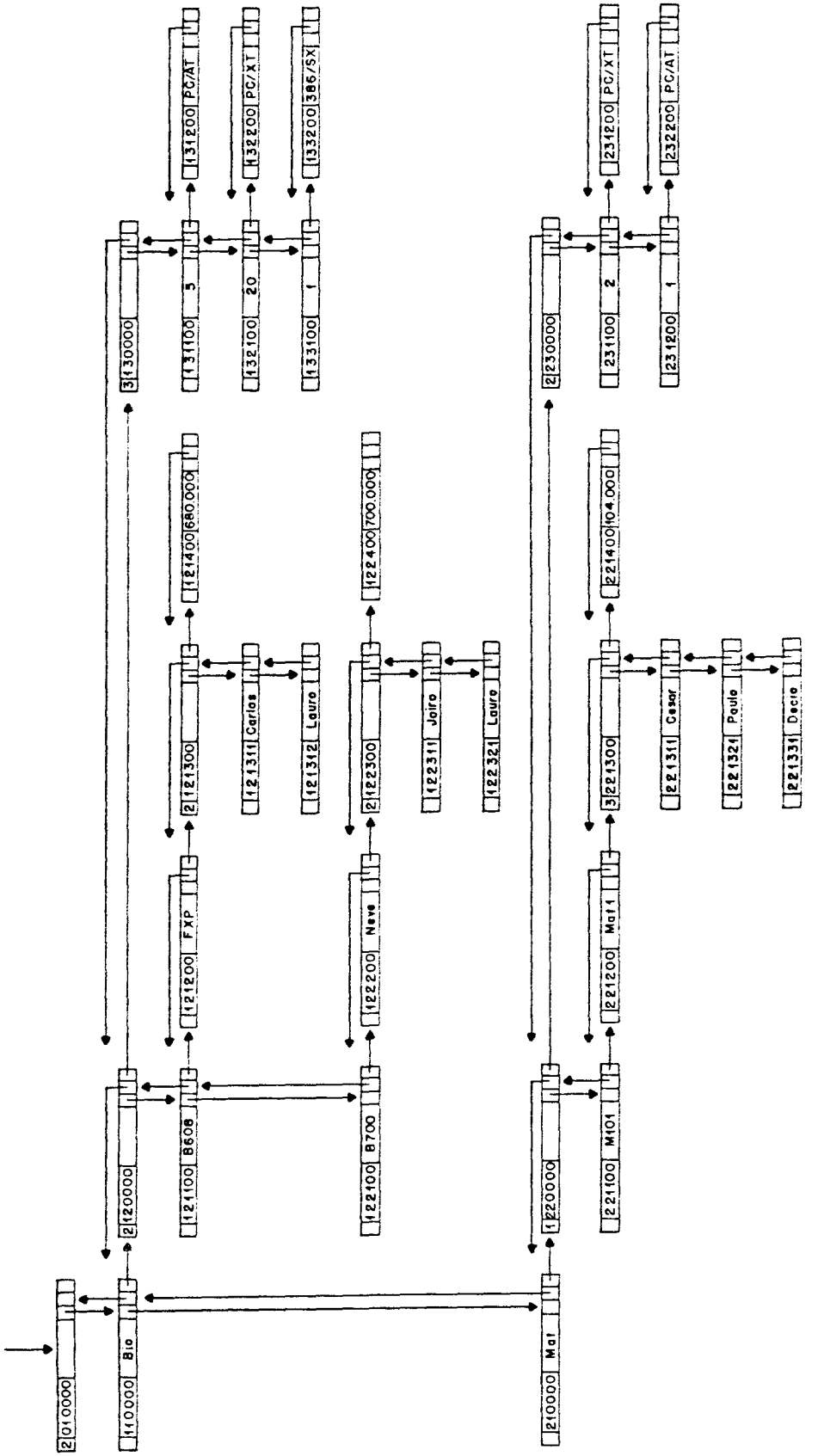


Após sucessivas execuções do algoritmo para os valores restantes, *DADOS_PROJ* adquire a configuração apresentada na página seguinte.

O *ALGORITMO DE ARMAZENAMENTO DE DADOS* é suficiente quando os dados estão sendo armazenados no início da geração da estrutura de dados. Uma vez que esta já foi gerada e deseja-se acrescentar uma tupla ou subtupla à relação, deve-se antes *buscar* o registro adequado, o qual precederá o primeiro a ser inserido.

Se uma tupla for inserida, deve-se avançar *<ap_tup>* a partir do registro cabeça-de-lista até o último registro de *DADOS_R* e, então, executa-se o *ALGORITMO DE ARMAZENAMENTO DE DADOS* (*apêndice B*).

Se uma subtupla S estiver para ser inserida, considere que A seja seu primeiro atributo e P, o pai de A. Deve-se, então, encontrar na estrutura de dados o registro que referencia P. A partir dele avança-se *<ap_tup>* até o último registro armazenado nesse sentido (correspondente à última subtupla de P armazenada). O *ALGORITMO DE ARMAZENAMENTO DE DADOS* é então executado, inserindo todos os valores da subtupla S na estrutura. A busca do registro que corresponde ao


2|010000
1|10000|Bio
2|120000
1|121100|B608
1|121200|FXP
2|121300
1|121311|Carlos
1|121312|Laura
1|121400|680.000
3|130000
1|131100|5
1|131200|PC/AT
1|132100|20
1|132200|PC/XT
1|133100|1
1|133200|386/SX
1|210000|Mat
1|220000
1|221100|M101
1|221200|Mat1
3|221300
1|221311|Cesar
1|221321|Paula
1|221331|Decio
1|221400|104.000
2|230000
1|231100|2
1|231200|PC/XT
1|231800|1
1|232200|PC/AT
1|122100|B700
1|122200|Neve
2|122300
1|122311|Jairo
1|122321|Laura
1|122400|700.000

Caso se queira apenas um valor X em uma subtupla composta de mais valores, como foi já foi usado um valor nulo no campo *<val_arm>* quando da inserção da subtupla, para que se preservasse a estrutura, basta que esse valor nulo seja *modificado* para X (ver *seção 6.3.4*).

6.3.2. Busca de Dados

A busca de um dado na estrutura proposta pode ser feita de duas maneiras: através de seu valor ou de seu identificador.

Para se providenciar uma busca por identificador, basta que se tenha o identificador completo de um certo dado. A busca por valor é extremamente lenta nas estruturas usadas, a menos que o atributo seja indexado. Como dito antes, a implementação busca apenas validar os algoritmos e, portanto, nenhum sistema de indexação foi utilizado.

Os dois tipos de busca mencionados acima podem não atender a todos os casos desejados, como, por exemplo, o caso de se querer encontrar o valor Lauro em MNOMES, sabendo-se que a chave é Bio. Como há dois Lauro em duas subtuplas distintas, o percurso estabelecido poderia parar quando se encontrasse o primeiro ou quando se esgotassem as subtuplas. Tudo depende dos objetivos reais da funcionalidade da busca.

Para os algoritmos do *capítulo 5*, a busca que é mais utilizada requer simplesmente que se aponte para o registro que contém o identificador procurado. Para tanto, a busca por identificadores é a mais adequada.

A seguir, é apresentado o *ALGORITMO DE BUSCA POR IDENTIFICADOR*.

O algoritmo tem por critério que toda busca comece a partir do nó cabeça-de-lista. Poder-se-ia pensar em encontrar um dado, ou seja, um registro da lista, a partir do último registro referenciado. As vantagens que isso aparentemente pode oferecer devem ser balanceadas com o fato de serem necessários vários tipos de

O algoritmo tem por critério que toda busca comece a partir do nó cabeça-de-lista. Poder-se-ia pensar em encontrar um dado, ou seja, um registro da lista, a partir do último registro referenciado. As vantagens que isso aparentemente pode oferecer devem ser balanceadas com o fato de serem necessários vários tipos de comparações com identificadores, para que o algoritmo seja consistente. Como na busca por identificador o percurso é muito simples, não há necessidade de maiores sofisticações.

ALGORITMO DE BUSCA POR IDENTIFICADOR

Suponha uma estrutura de relação $\overline{R}$ = (R, r), *com suas respectivas estruturas de armazenamento ESQ_R e DADOS_R.*

Seja **ID** *o identificador do elemento a ser encontrado. Seja* **p** *uma variável usada para se percorrer as posições de* **ID**, *da esquerda para a direita, inicialmente com o valor 1 (primeira posição). Seja* **v(p)** *o valor de* **ID** *correspondente à posição* **p**.

Suponha uma variável AP_ATUAL, que é um apontador que navega na estrutura, cujo valor referencia o registro (ATUAL) que está sendo lido.

Repetir os passos abaixo até **v(p)** *ser igual a zero ou até o fim de* **ID**.

(a) *Avance o apontador* <ap_tup> *dos registros* **v(p)** *vezes a partir de ATUAL. O último registro encontrado passa a ser o ATUAL.*

(b) *Incremente 1 ao valor de* **p**.

(c) *Avance o apontador* <ap-atr> *dos registros* (**v(p)** - 1) *vezes a partir de ATUAL. O último registro encontrado passa a ser o ATUAL.*

(d) *Incremente 1 ao valor de* **p**.

6.3.3. Eliminação de Dados

A eliminação de um dado da tabela (relação) r reflete-se na estrutura de dados de duas maneiras:

(1) Se o dado a ser eliminado é o primeiro elemento de um atributo simples numa tupla/subtupla, o registro que o contém não pode ser simplesmente retirado; isso quebraria a ordem dos atributos para o percurso na estrutura. Nesse caso, seu valor simplesmente é "apagado" do registro. É o que aconteceria se, por exemplo, FXP fosse retirado de *DADOS_PROJ*.

(2) Se o dado a ser eliminado pertence a um atributo composto, como já há um registro que mantém a posição do atributo, todo o registro é eliminado, bem como toda a lista de elementos apontado por seu campo *<ap_atr>*. O campo *<num-tup>* do registro referente ao atributo composto é, então, decrementado de 1. Os ponteiros *<ap_tup>* dos registros superior e inferior na hierarquia são reajustados. É o caso, por exemplo, de B608. Para retirá-lo, toda a subtupla encabeçada por ele deixa de existir.

No caso (2) acima, é necessário que se reajuste o valor da marca discriminante dos identificadores nos registros pertencentes às subtuplas seguintes.

Caso se queira eliminar todas as subtuplas de um atributo composto, basta que o campo *<ap_tup>* do registro do atributo (de valor nulo) deixe de referenciá-las. Não se deve esquecer de atualizar o campo *<num_tup>* desse registro.

Ao se eliminar um valor chave da estrutura, elimina-se toda a tupla correspondente da relação. Conseqüentemente, o campo *<num_tup>* do nó cabeça-de-lista é decrementado de 1.

6.3.4. Modificação de Dados

A modificação de um valor é trivial. Basta que se faça uma busca na estrutura, por identificador ou por valor, e que se altere o campo *<val_arm>* do registro para o valor desejado.

No caso de se desejar eliminar apenas um valor de uma subtupla, deixando intacto seu restante, ou seja, a lista apontada por *<ap_atr>* de seu registro, basta modificar o valor de seu campo *<val_arm>* para o valor nulo utilizado.

Vale lembrar que atributos chave não podem ter valor nulo.

7. Conclusão e Sugestões para Trabalhos Futuros

7.1. CONCLUSÃO

Esta tese propôs um conjunto de estruturas e algoritmos para implementação de operações em álgebra não-normalizada. Estas estruturas e algoritmos foram parcialmente implementados em um protótipo em linguagem Pascal num ambiente PC/DOS, usando listas duplamente encadeadas para armazenar tabelas de esquema e dados.

Dentre as principais contribuições da dissertação, destacam-se:

a) Revisão bibliográfica sobre o modelo NF^2, tanto do ponto de vista teórico quanto de implementação, estabelecendo paralelo com sistemas orientados a objetos. Esta revisão faz igualmente comparações entre as diversas propostas de normalização.

b) discussão sobre operadores da álgebra relacional aninhada, comparando as propostas de diversos autores e selecionando um conjunto adequado a implementar.

c) propostas de estruturas para construção de esquemas e relações não-normalizadas, com especificação detalhada de algoritmos para implementar as operações da álgebra NF^2.

d) descrição de estruturas dinâmicas para suportar os operadores, e sua implementação, criando, assim, um núcleo de operações em álgebra não-normalizada. A implementação mostra a viabilidade dos algoritmos algébricos propostos, sem, no entanto, visar desempenho. Salienta-se que há pouquíssima coisa descrita na

literatura sobre algoritmos e implementação deste tipo de sistema, já que os autores preferem se deter sobre a descrição funcional dos operadores.

7.2. CONSIDERAÇÕES FINAIS

O modelo relacional normalizado (MRN), restrito à 1NF de Codd, embora bem fundamentado matematicamente, não se mostra eficiente em várias aplicações práticas reais. É o caso de aplicações que requerem a conservação da hierarquia natural de seus dados, os quais podem ser valores singulares ou conjuntos de valores (singulares ou não). Como exemplos, tem-se os ambientes para gerenciamento de dados científicos e de engenharia, para automação de escritórios, gerenciamentos de textos, etc.

O fato do MRN não ser o modelo "adequado" a certos tipos de aplicação não significa que se tenha de "deixá-lo de lado", "jogando fora" toda a sua fundamentação, flexibilidade e poder semântico. A proposta do MRNN também não vem destruir essas boas qualidades do MRN, e sim, expandi-las. O MRNN "liberou" o MRN da primeira forma normal, que dificultava o uso desse modelo para certas aplicações devido aos seus registros planos.

O MRNN é, praticamente, uma integração entre o modelo relacional clássico e o modelo hierárquico, conservando as boas propriedades do MRN. Essa integração é propícia para a representação dos conceitos de *orientação a objetos*, como visto no *capítulo 1 (seção 1.2)*; cada supertupla pode corresponder a um *objeto complexo*.

De um modo geral, o tratamento de relações aninhadas é um tema de pesquisa muito recente. Ao longo desta dissertação pôde-se observar a quantidade de propostas teóricas sobre o MRNN, enquanto que o MRN é tratado uniformemente por todos que o utilizam. Isso pode ser *sentido* já nas diferentes representações das tabelas relacionais aninhadas (*seção 1.1*). As diferentes correntes de normalização (*capítulo 2*) também são exemplos dessa heterogeneidade de tratamento,

o que até pode ser justificado pela extensão do MRNN. Esse fato, aliás, faz com que muitos pesquisadores apresentem suas propostas usando nomes como "*modelo NI^2 estendido*".

Embora essa diversifição seja refletida também no sistema algébrico do modelo, as operações usadas no MRNN são extensões das operações definidas no MRN. O conjunto de operações aninhadas é acrescido, ainda, de operadores que *aninham* (*NEST*) e *desaninham* (*UNNEST*) atributos, enriquecendo a álgebra relacional. Para efeitos desta tese, foi selecionada a álgebra de Fischer e Thomas [FiT83] descrita no *capítulo 3*, devido à sua simplicidade e extensão.

As consultas sobre um BD em NF^2 são também beneficiadas. Alguns trabalhos trazem extensões da linguagem SQL, como pôde ser visto na *seção 3.2*, mostrando como o potencial dessa linguagem pode ser aumentado no MRNN.

É de se esperar que um modelo poderoso torne-se mais difícil de ser gerenciado. De fato, as estruturas de dados do MRNN são, obviamente, mais complexas que as do MRN. As estruturas de dados simples, dependendo da aplicação, não podem refletir naturalmente restrições de consistência [PiT86]. Além disso, elas não implicam, necessariamente, em operações simples (exemplo: consultas). Isso significa que, usando-se estruturas simples, a complexidade de determinadas operações pode aumentar, ao contrário do que ocorre no modelo NF^2, onde as operações permanecem gerenciáveis sobre as suas estruturas, mesmo complexas.

Assim como no MRN, a implementação do modelo relacional aninhado exige muito mais do que simples tabelas para a representação dos dados. Também é preciso que sejam gerados caminhos de acesso, um pouco mais complexos no MRNN devido aos aninhamentos, os quais permitem a efetivação das operações relacionais de um modo eficiente.

7.3. SUGESTÕES PARA TRABALHOS FUTUROS

Pretende-se dar continuidade aos estudos desta tese, objetivando-se concluir um sistema de armazenamento em NF^2. A seguir

são apresentados os tópicos que, a princípio, constituem a meta desses estudos.

7.3.1. Estruturas Adicionais

Pretende-se gerar um núcleo de BD sobre o qual poderão ser processadas as operações relacionais estendidas, conforme os algoritmos do *capítulo 3*. Atualmente, encontram-se implementadas as estruturas de armazenamento propostas no *capítulo 4* (*ESQ_R* e *DADOS_R*), bem como as operações de atualização dos dados (*capítulo 4*) e algumas operações relacionais estendidas (*capítulo 3*).

Embora as estruturas de dados apresentadas anteriormente possibilitem a execução das operações da álgebra relacional aninhada (*capítulo 5*), poder-se-ia pensar em estruturas adicionais que agilizassem o processamento das consultas. Em sistemas de BD relacionais convencionais, é comum o uso de técnicas de indexação para melhorar o tempo de acesso. Obviamente, tal rapidez de processamento se dá em detrimento do espaço disponível.

Para as operações direcionadas por valor, como é o caso da seleção, do aninhamento e da junção, sugerimos como extensão uma estrutura que facilita o acesso aos dados, inspirada em [DeG88].

O objetivo dessa estrutura é fornecer a posição exata (identificador) de um certo valor na estrutura de dados de uma relação. Deve-se lembrar que um mesmo valor pode aparecer mais de uma vez em mais de uma relação. Nesses casos, haverá mais de um identificador para tal valor. Essa estrutura é única para todas as relações do BD e funciona de forma semelhante a índice para achar todas as ocorrências de um valor.

Os valores atômicos do BD devem ter seus *domínios* especificados, de modo a facilitar o acesso. Como exemplo, considere o BD que contém a estrutura de relação $\overline{PROJ}$, apresentada na *seção 4.1*. Seus valores poderiam pertencer a domínios conhecidos como: *nomes* (nomes dos membros dos projetos), *nomes_Depto* (valores do atributo DEPTO), *eqptos* (tipos de equipamentos), etc.. Essa estrutura deverá

indexar apenas os valores correspondentes aos domínios acessados mais freqüentemente para consulta.

A estrutura sugerida, que chamaremos de VAL_ATOMIC (lembra "valores atômicos"), é um encadeamento de registros, distribuídos segundo os domínios do BD.

Basicamente, VAL_ATOMIC possui quatro níveis:

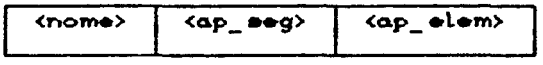
1º nível: contém os nomes dos domínios do BD;

2º nível: contém valores atômicos, classificados segundo os domínios do nível 1;

3º nível: contém os nomes das relações às quais os valores do nível 2 pertencem;

4º nível: contém uma lista ligada de identificadores, associada a cada valor atômico; uma para cada relação.

Os registros dos três primeiros níveis têm a seguinte estrutura básica:



onde:

<nome> é o campo que contém ou um nome de domínio (1º nível) ou um valor atômico (2º nível) ou, ainda, um nome de relação (3º nível);

<ap_seg> é o campo que contém um apontador para o nome de domínio seguinte (se 1º nível) ou para o próximo valor atômico (se 2º nível) ou, ainda, para o próximo nome de relação (se 3º nível);

<ap_elem> é o campo que aponta para um valor atômico do domínio (se 1º nível), para um nome de relação (se 2º nível) ou para uma lista ligada de identificadores (se 3º nível).

No quarto nível, os registros não precisam de dois campos

apontadores. Dessa forma, seus registros não contêm *<ap_elem>*. O campo *<nome>* agora contém um identificador de tupla e *<ap_seg>*, um apontador para o identificador seguinte da relação.

Para maior rapidez de acesso às informações de VAL_ATOMIC, os dados, em todos os níveis, devem estar ordenados.

Considerando, então, o BD que contém $\overline{PROJ}$ (*seção 4.1*), VAL_ATOMIC poderia ser parcialmente representada (a apresentação completa é redundante) conforme a ilustração na página seguinte.

Desse modo, se houvesse uma outra relação no BD, digamos $\overline{R}$, contendo *Carlos* entre seus valores, o registro correspondente contendo PROJ em VAL_ATOMIC, no terceiro nível, passaria a ter seu campo *<ap_seg>* apontando para:

onde os valores *<nomei>*, $1 \leq i \leq n$, representam os possíveis identificadores de *Carlos* em $\overline{R}$.

A adição de VAL_ATOMIC às três estruturas do sistema proposto nesse capítulo facilita, sem dúvida, a recuperação dos dados. No entanto, não se pode esquecer que suas informações devem ser interpretadas juntamente com as informações da tabela de identificadores de cada relação envolvida. É o caso, por exemplo da junção de duas relações, onde os atributos comuns ocupam posições diferentes em ambas relações.

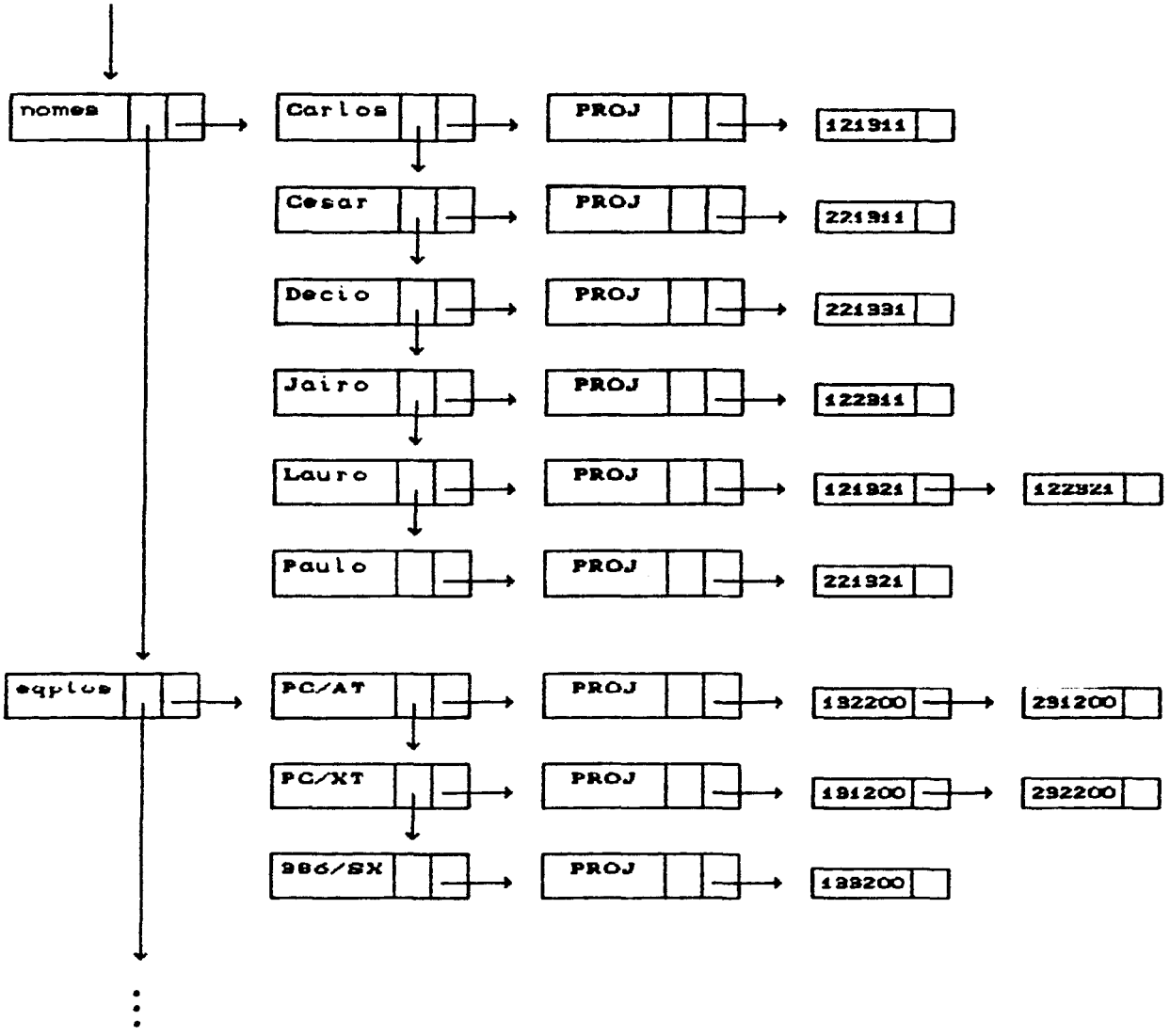
A estrutura inspiradora de VAL_ATOMIC é proposta em [DeG88], na arquitetura ANDA (*Acronymn for Nested Database Architecture*). Os autores, **Deshpande e Gucht**, apresentam uma estrutura de árvore, VALTREE, com cinco níveis básicos.

Os dois primeiros níveis da VALTREE são semelhantes aos dois primeiros de VAL_ATOMIC. No terceiro nível são armazenados todos os atributos aos quais um determinado valor do segundo nível pertence. O quarto nível contém o nome da estrutura de relação à qual um certo atributo do terceiro nível pertence. Finalmente, o quinto nível é semelhante ao quarto nível de VAL_ATOMIC; contém os identificadores

para cada valor, segundo seus atributos.

A especificação explícita dos atributos aos quais um certo valor pertence não foi considerada em VAL_ATOMIC, pois os identificadores (4º nível) já contêm esse tipo de informação embutida.

Exemplo de VAL_ATOMIC:



Com o uso de estruturas como VAL_ATOMIC e VALTREE, tem-se uma generalização das técnicas de *arquivos invertidos* e *índices de junção* [Val87] (ver também *seção 1.4.1*), ambas já utilizadas em sistemas relacionais convencionais para facilidade e rapidez de acesso aos dados.

7.3.2. Estruturas de Armazenamento em Disco

As estruturas físicas propostas não apresentam bom desempenho para armazenamento em disco. Desta forma, deverão ser estudadas alternativas às estruturas do capítulo 6. Uma dessas alternativas é a proposta de [Set86], onde as tuplas são armazenadas em registros de tamanho variável, com compressão dos dados pela eliminação de zeros à esquerda de campos numéricos e de brancos à esquerda de campos alfanuméricos, além de suprimir dados vazios.

Os valores de uma tupla são armazenados seqüencialmente num registro, considerando a ordem que aparecem na relação, da esquerda para a direita. Imediatamente à esquerda de cada valor existe um *byte* para a contagem de *bytes* do campo que segue. Para tratar valores de atributos multivalorados, um *byte* adicional, à esquerda do anterior, controla o número de ocorrências em cada tupla/subtupla. Atributos compostos têm tratamento semelhante: para cada nível de composição há um contador do espaço total por elas ocupado em *bytes*. Esse processo permite um número arbitrário de *encaixamento* de multivalorações e composições [Set86].

Essa representação é uniforme, pois não existe nenhuma indicação discriminante do tipo do atributo (simples, composto ou multivalorado) nos campos dos registros. Isso se deve a existência de um Dicionário/Diretório de Dados que descreve as relações e os atributos.

7.3.3. Normalização do Sistema

Para se evitar redundâncias no sistema e garantir uma boa representação semântica, pretende-se acoplar ao Sistema uma *interface* que receba do usuário os atributos utilizados, assim como as dependências funcionais e multivaloradas, completas e embutidas, sobre eles e resulte numa decomposição *NNF (Nested Normal Form)* (ver *capítulo 2*). Poder-se-á, então, avaliar se o custo adicional da normalização será compensado pelo cusato de processamento das operações.

7.3.4. Linguagem de Manipulação dos Dados

Para a padronização e eficiência das consultas sobre o sistema, é essencial a definição de uma linguagem clara, que represente as noções de composição e multivaloração do MRNN. Uma possibilidade é usar uma extensão da SQL, conforme visto na *seção 3.2*. A princípio não se pensa em otimização algébrica.

7.3.5. Generalização do Conceito de Chave

Como ressaltado na *seção 5.3*, a hipótese de chave simples e monovalorada pode ser violada se aplicada uma operação de *UNNEST*.

Uma direção futura é, portanto, passar a considerar chaves compostas, o que altera minimamente a descrição lógica dos algoritmos, mas implica mudanças na implementação. As operações afetadas são as que utilizam *COMPARA_TUPLAS ou COMPARAÇÃO REDUZIDA*. Para estes algoritmos de comparação, a única modificação consiste em considerar chaves compostas ao selecionar tuplas. Além disso, como vários atributos passarão a fazer parte da chave, há mudança na *linha 2* da tabela de esquema, onde tais atributos receberão valor

-1.

Um problema maior, caso se deseje manter o conceito de chave, é a designação de uma nova chave se uma operação de desaninhamento tornar isso necessário. Isto depende, no entanto, do conhecimento das dependências que afetam a relação. Uma automatização total da álgebra com manutenção de chave exigiria desta forma algoritmos que tivessem conhecimentos não só do esquema, como também das dependências existentes.

Uma alternativa para contornar este último problema seria abandonar a exigência de chave. Neste caso, os algoritmos de comparação de tuplas seriam muito mais demorados.

Em quaisquer destes casos, as estruturas físicas propostas e os algoritmos que as manipulam deixam de ser aplicáveis.

8. Referências Bibliográficas

[AbB84]. ABITEBOUL, S., BIDOIT, N. Non First Normal Form Relations to represent hierarchically organized data. Proceedings of the 3.th ACM SIGACT-SIGMOD, Waterloo, Ontário, Canadá, p.191-199, 1984.

[ABU79]. AHO, A.V., BEERI, C., ULLMAN, J.D. The Theory of Joins in Relational Databases. ACM Transactions on Database System, V.4, N.3, p.297-314, 1979.

[AMM83]. ARISAWA, H., MORIYA, K., MIURA, T. Operations and the Properties on Non-First-Normal-Form Relational Databases. Proceedings VLDB, Singapura, p.197-204, 1983.

[ASK80]. ASTRAHAN, M.M., SCHKOLNICK, M., KIM, W. Performance of the System/R Acess Path Selection Mechanism. IFIP Congress, 1980. Citado por [SPS87].

[BaH84]. BABAD, Y.M., HOFFERT, J. Even no data has value. Communications of the ACM, V.27, N.8, p.748-756, 1984.

[BBG78]. BEERI, C., BERNSTEIN, P.A., GOODMAN, N. A Sophisticated's Introduction to Database Normalization Theory. VLDB Conference, p.113-124, 1978.

[BeK84]. BEERI, C., KIFER, M. Comprehensive approach to the design of relational database schemes. Proceedings of the 10.th International Conference on VLDB. Singapore, p.196-207, Aug, 1987. Citado por [RoK87].

[Bid87]. BIDOIT, N. The VERSO algebra or how to answer queries with fewer joins. Journal of Computer and System Sciences, V.35, N.3, p.321-364, Dec, 1987. Citado por [RoK88].

[Cod70]. CODD, E.F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, V.13, N.6, p.377-387, Junho, 1970. Citado por [Set86].

[Cod72]. CODD, E.F. Further Normalization of the Data Base Relational Model. Courant Computer Science Symposia 6, "Data Base Systems", Prentice Hall, Englewood Cliffs, p.33-64, 1972. Citado por [Set86].

[Cod86]. CODD, E.F. O seu SGBD é realmente relacional? (Parte 1)- Seu SGBD funciona de acordo com as regras? (Parte 2). DataNews, fev, 1986.

[Dad88]. DADAM, P. Advanced Information Management (AIM): Research in Extended Nested Relations. IEEE Database Engineering, Special Issue on Nested Relations, V.11, N.3, Sept, 1988.

[Dad86]. DADAM, P. et al. A DBMS Prototype to Support Extended NF^2 Relations: An Integrated View on Flat Tables and Hierarchies. SIGMOD Record, V.15, N.2, p.356-367, 1986.

[Dat84]. DATE, C.J. Introdução a Sistemas de Bancos de Dados. Rio de Janeiro: Editora Campus, 1984, 513p.

[DeG88]. DESHPANDE, A., GÜCHT, D.V. An Implementation for Nested Relational Databases. Proceedings of the 14^{th} VLDB Conference, Los Angeles-California, p.76-87, 1988.

[DeL87]. DESHPANDE, V., LARSON, P.A. An Algebra for Nested Relational Databases. Technical Report, University of Waterloo, Waterloo-Canadá, Nov, 1987. Citado por [DeG88].

[Dit86]. DITTRICH, K.R. Object-Oriented Database Systems. Proceedings of the 1986 International Workshop on Object-Oriented Database Systems, Pacific Grove, Ca, USA, p.23-26, Sept., 1986.

[Fag79]. FAGIN, R. Multivalued Dependencies and a New Normal Form for Relational Databases. ACM Transactions on Database Systems, V.12, N.3, p.262-278, Sept., 1977.

[FaV84]. FAGIN, R., VARDI, M.Y. The theory of Data Dependencies An Overview. Letture Notes *in* CS, 11th.Colloquium on Automata, Language & Programming, Jul, 1984.

[FiT83]. FISCHER, P.C., THOMAS, S.J. Operators for Non-First-Normal-Form Relations. Proceedings COMPSAC, p.464-475, 1983.

[FuS88]. FURTADO, A.L., SANTOS, C.S. Organização de Banco de Dados. Rio de Janeiro-RJ: Editora Campus, 1988, 281p.

[Gho86]. GHOSH, S.P. Statiscal Relational Tables for Statistical Database Management. IEEE Transactions on Software Engineering, V.SE.12, N.12, p.1106-1116, dec, 1986.

[HaO88]. HAFEZ, A., OZSOYOGLU, G. The Partial Normalized Storage Model of Nested Relations. Proceedings of the 14th. International conference on VLDB, Los Angeles-USA, 1988.

[HaO89]. HAFEZ, A., OZSOYOGLU, G. Storage Structures for Nested Relations. Thechnical Report, Dept. of Computer Science, Case Western Reserve University, p.31-38, 1989.

[Här78]. HÄRDER, T. Implementing a Generalized Acess Path Structure for a Relational Database System. ACM TODS, V.3, N.3, p.285-298, 1978. Citado por [SPS87].

[JaS82]. JAESCHKE, G., SCHEK, H.J. Remarks on the Algebra of Non-First-Normal-Form Relations. Proceedings of the SIGACT-SIGMOD Symposium on Principles of Database Systems, Los Angeles, p.124-128, March, 1982. Citado por [FiT83].

[Jhi88]. JHINGRAN, A. A Performance Study of Query Optimization Algorithms on a Database System Supporting Procedures. Proceedings of the 14th VLDB Conference, Los Angeles, California, p.88-99, 1988.

[JaK84]. JARKE, M., KOCH, J. Query Optimization in Database Systems. ACM Computing Surveys, June, 1984. Citado por [Jhi88].

[Kat84]. KATSUNO, H. An extension of conflit-free multivalued dependencies. ACM Transactions on Database Systems, V.9, N.2, p.309-326, June, 1984. Citado por [RoK87].

[ken83]. KENT, W. A Simple Guide to Five Normal Forms in Relational Database Theory. Communications of the ACM, V.26, N.2, p.120-125, 1983.

[Kim90]. KIM, W. Object-Oriented Databases: Definition and Research Directions. IEEE Transactions on Knowledge and Data Engeneering, V.2, N.3, p.327-341, Sept., 1990.

[Kir88]. KIRKPATRICK, J.E. The Natural Join of Nested Relations. Thechnical Report, Air Force Institute of Technology, AFIT, ENG, Wright-Patterson AFB, OH 45433, 1988. Citado por [RoK88].

[LeC86]. LEHMANN, T.J., CAREY, M.J. A Study of Index Structures for Main Memory Database Management Systems. Proceedings VLDB, Kyoto, 1986. Citado por [SPS87].

[LoP83]. LORIE, R.A., PLOUFFE, W. Complex Objects and Their Use in

Design Transactions. Proceedings of Annual Meeting - Database Week: Engeneering Design Applications (IEEE), San Jose-California, p.115-121, May, 1983. Citado por [Dad86].

[Mak77]. MAKINOUCHI, A. A Consideration on Normal Form of Not-Necessarily-Normalized in the Relational Data Model. Proceedings of the the ACM International Conference on VLDB, Tokyo, p.447-453, Oct., 1977. Citado por [Fit83].

[McG77]. McGGE, W.C. The Information Management System IMS/VS - Part 1: General Structure and Operations. IBM System J., V.16, N.2, 1977. Citado por [Ha089].

[Mel87]. MELO, R.N.. Bancos de Dados Não Convencionais. In: VI JORNADA DE ATUALIZAÇÃO DE INFORMÁTICA, Salvador, BA: SBC, 1987, 23p..

[Mis83]. MISSIKOFF, M. A Domain based internal schema for relational Database. Proceedings of ACM-SIGMOD 1983, International Conference on Management of Data, San Jose, p.215-224, 1982. Citado por [DeG88].

[MiS83]. MISSIKOFF, M., SCHOLL, M. Relational queries in domain based DBMS. Proceedings of ACM-SIGMOD 1983, International Conference on Management of Data, San Jose, p.219-227, 1983. Citado por [DeG88].

[Oli88]. OLIVEIRA, D.L. Aspectos Teóricos do Modelo Relacional de Dados e Projeto de Relações. São José dos Campos-SP: ITA, 1988, 155p. Dissertação de Mestrado.

[Oll71]. OLLE, T.W. Introduction to 'Feature Analisys of Generalized Data Base Management Systems. CACM, V.14, N.5, May, 1971. Citado por [Ha089].

[OtH85]. OTT, N., HORLÄNDER, K. Removing Redundant Join Operations in Queries Involving Views. Information Systems, V.10, N.3, 1985. Citado por [SPS87].

[OzO]. OZSOYOGLU, Z.M., OZSOYOGLU, G. A Query Language for Statistical Databases. p.171-187.

[OzY87]. OZSOYOGLU, Z.M., YUAN, L.Y. A New Normal Form for Nested relations. ACM Transactions on Database System, V.12, N.1, p.111-136, 1987.

[Per87]. PERNUL, G. An Unnormalized Relational Data Model Based on User Views. SIGMOD RECORD, V.16, N.2, p.51-60, Sept., 1987.

[PiT86]. PISTOR, P., TRAUNMUELLER, R. A Database Language for Sets, Lists and Tables. Information Systems, V.11, N.14, p.323-336, 1986.

[Rot86]. ROTH, M.A. Theory of Non-First-Normal-Form Relational Databases. Texas: The University of Texas at Austin, May, 1986. Dissertação (Ph.D. Thesis). Citado por [RoK88].

[RoK88]. ROTH, M.A., KIRKPATRICK, J.E. Algebras for Nested Relations. p.39-47, 1988.

[RoK87]. ROTH, M.A., KORTH, H.F. The Design of non-1NF Relational Databases into Nested Normal Form. Proceedings of SIGMOD, p.143-159, 1987.

[RKB87]. ROTH, M., KORTH, H.F., BATORY, D.S. SQL, NF: A Query Language for ¬ 1NF Relational Databases. Information Systems, V.12, N.1, p.99-114, 1987. Citado por [RoK88].

[RKS84]. ROTH, M.A., KORTH, H.F., SILBERSCHATZ, A. Theory of non-first-normal-form relational database. TR-84-36,

Dept. of. Computer Science, University of Texas at Austin, Dec, 1984. Citado por [OzY87].

[RKS88]. ROTH, M.A., KORTH, H.F., SILBERSCHATZ, A. Extended Algebra and Calculus for Nested Relational Databases. ACM Transactions on Database Systems, V.13, N.4, p.389-417, Dec., 1988.

[ScP82]. SCHEK, H.J., PISTOR, P. Data Structures for an integrated data base management and information retrieval system. Proceedings of the 8.th International Conference on VLDB, Cidade do México, p.197-207, 1982. Citado por [Set86].

[ScS86]. SCHEK, H.J., SCHOLL, M.H. The Relational Model with Relation-Valued Attributes. Information Sustems. V.11, N.2, 1986. Citado por [SPS88].

[ScS80]. SCHKOLNIK, M., SORENSON, P. Denormalization: A Performance Oriented Database Design Techinique. Proceedings AICA, Bologna, Itália, 1980. Citado por [SPS87].

[ScS81]. SCHKOLNIK, M., SORENSON, P. The Effects of Denormalization on Database Performance. Res.Pep RJ 3082 (38128), IBM Res. Lab. San Jose, Califórnia, 1981. Citado por [SPS87].

[SPS87]. SCHOLL, M.H., PAUL, H.B., SCHEK, H.J. Supporting Flat Relationals by a Nested Relational Kernel. Proceedings of the 13.th VLDB Conference, Brighton, p.137-146, 1987.

[Set86]. SETZER, W.W. - Projeto Lógico e Projeto Físico de Banco de Dados. In: IV ESCOLA DE COMPUTAÇÃO, UFMG. Belo Horizonte-MG, 1986.

[Per87]. PERNUL, G. An Unnormalized Relational Data Model Based on User Views. SIGMOD RECORD, V.16, N.2, p.51-60, 1987.

[Tak88]. TAKAHASHI, T. Introdução à Programação Orientada a Objetos. Edição EBAI, Curitiba-PR, 1988, 148p..

[TLX90]. TAKAHASHI, T., LIESENBERG, H.K.E., XAVIER, D.T. Programação Orientada a Objetos. In: VIII ESCOLLA DE COMPUTAÇÃO, São Paulo-SP, 1990, 335 p..

[TeA81]. TENEBAUM, A.M., AUGENSTEIN, M.J. Data Structures Using Pascal. Englewood Cliffd-New Jersey: Prentice-hall, Inc., 1981, 545 p.

[Tho83]. THOMAS, S. A Non-First-Normal-Form Relational Database Model. Vanderbilt University, August, 1983. Ph.D. Dissertation. Citado por [Fit83].

[Wie83]. WIEDERHOLD, G. Database Design. 2nd ed. McGraw-Hill, 1983. Citado por [Ha089].

[Wie86]. WIEDERHOLD, G. Views, Objects, and Databases. Computer (USA), V.19, N.12, p.37-44, Dec, 1986.

[Yu086]. YUAN, L., OZSOYOGLU, Z. Unifying functional and multivalued dependencies for relational database design. Proceedings of Fifty ACM SIGACT-SIGMOD: Symposium on Principles of Database Systems, Cambridge, p.183-190, March, 1986. Citado por [RoK87].

APÊNDICE A

ALGORITMO COMPARA_TUPLAS

Sejam $\overline{R_1}$ *e* $\overline{R_2}$ *duas relações e* **a1** *o identificador de uma tupla* **t** *de* $\overline{R_1}$ *dada, onde* $1 \leq a \leq n$ *, onde* **n** *é o número de tuplas de* $\overline{R_1}$ *(lembre que os caracteres à direita de* **1** *são zeros!).*

Seja **z(id)** *o valor de uma tupla* **z** *identificado por* **id**.

Seja **AUX** *um vetor contendo os valores das marcas mais significativas das chaves de todas as tuplas de* $\overline{R_2}$ *já comparadas com sucesso em execuções anteriores do algoritmo.*

(a) *Percorre-se as chaves de* $\overline{R_2}$ **(b1)** *até se encontrar um* **b** *pertencente a uma tupla* **u**, $1 \leq b \leq m$ *(= número de tuplas de* $\overline{R_2}$ *), tal que* $b \notin AUX$ *e* **t(a1) = u(b1)**. *Caso isso não ocorra, a comparação foi mal sucedida e interrompe-se o algoritmo.*

(b) *Para cada atributo simples de* $\overline{R_1}$, *toma-se seu identificador genérico, digamos* **id**, *e comparam-se os valores dados por* **t(id1)** *e* **u(id2)**, *onde:*

> **id1** *é igual a* **id** *com os valores das marcas iguais as do último elemento considerado de* $\overline{R_1}$ *de mesmo nível.*
>
> *e*
>
> **id2** *é igual a* **id** *com os valores das marcas iguais as do último elemento considerado de* $\overline{R_2}$ *de mesmo nível.*

Caso alguma comparação falhe, interrompe-se o algoritmo.

(c) *Para cada atributo composto* **C** *de* $\overline{R_1}$ *de nível* **k** *executa-se os*

passos abaixo, usando-os recursivamente para os descendentes compostos de C.

(d) Verifica-se se o número de elementos do conjunto de valores de C em $\overline{R_1}$ coincide com o número de elementos do candidato a conjunto correspondente em $\overline{R_2}$. Caso isso não aconteça, interrompe-se o algoritmo.

(e) Seja id o identificador genérico do primeiro atributo componente simples de C. Com exceção da marca discriminante, as demais marcas recebem valores iguais aos do último identificador de $\overline{R_1}$ de nível k-1 (se k > 1). A marca discriminante será iterada de 1 até o valor da linha 5 correspondente ao atributo (Tabela de Esquema), segundo uma variável i.

(e1) A cada instância de i, id tem as marcas, com exceção da discriminante, recebem valores iguais aos do último identificador de $\overline{R_2}$, conforme (c). A marca discriminante será iterada até que se encontre um valor igual ao referenciado em $\overline{R_1}$ ou até se esgotarem as possibilidades, segundo a Tabela de Esquema (correspondente linha 5); neste último caso, interrompe-se o algorimo.

(e2) Para cada atributo simples restante de D, execute o passo (b).

(f) Coloca-se, na próxima posição livre do vetor AUX, o valor da marca discriminante da chave de $\overline{R_2}$, referente à tupla comparada (com sucesso), de modo a eliminá-la de comparações posteriores.

ALGORITMO DE COMPARAÇÃO REDUZIDA

Sejam $\overline{R_1}$ e $\overline{R_2}$ duas relações e a1 o identificador de uma tupla t de $\overline{R_1}$ dada, onde $1 \leq a \leq n$ (≡ número de tuplas de $\overline{R_1}$) e b1 o

identificador de uma tupla **u** *de* $\overline{R_2}$ *dada, onde* $1 \leq b \leq n$ (≡ *número de tuplas de* $\overline{R_2}$). (*Lembre que os caracteres à direita de* **1** *são zeros!*).

Seja **z(id)** *o valor de uma tupla* **z** *identificado por* **id**.

Seja COMUM a subárvore comum às árvores de esquema de $\overline{R_1}$ *e* $\overline{R_2}$.

Seja **AUX** *um vetor contendo os valores das marcas mais significativas das chaves de todas as tuplas de* $\overline{R_2}$ *já comparadas com sucesso em execuções anteriores do algoritmo.*

(a) *Para cada atributo simples de COMUM, digamos* **S**, *seja* **id1** *seu identificador genérico relativamente à* $\overline{R_1}$ *e* **id2**, *relativamente à* $\overline{R_2}$.

(a_1) *Encontrar* $b \notin$ *AUX tal que a comparação de* **id1** *com* **id2** *tendo os valores de suas marcas mais significativas iguais à* **a** *e* **b**, *respectivamente, seja bem sucedida.*

Caso isso não ocorra, a comparação é mal sucedida e o algoritmo é abortado.

(b) *Para cada atributo composto* **C** *de* $\overline{R_1}$ *de nível* **k** *executa-se os passos abaixo recursivamente para os descendentes compostos de* **C**.

(b_1) *Verificar se o número de elementos do conjunto de valores de* **C** *em* $\overline{R_1}$ *coincide com o número de elementos do candidato a conjunto correspodente em* $\overline{R_2}$. *Caso isso não ocorra, retorna-se ao passo* **(a_1)**.

(b_2) *Para cada atributo simples* **S** *de* **C**, *seja* **id1'** *seu identificador genérico relativamente à* $\overline{R_1}$ *e* **id2'** *relativamente à* $\overline{R_2}$. *Com exceção da marca discriminante, os demais recebem os valores válidos nos últimos identificadores referenciados respectivamente em* $\overline{R_1}$ *e* $\overline{R_2}$. *Itere a marca discriminante de* **id1'** *de modo a percorrer todas as subtuplas de* **C** *em* $\overline{R_1}$ (*relativamente à subtupla mais externa, se houver*).

A cada instância dessa iteração, correspondente a uma subtupla t' de C em $\overline{R_1}$*, encontrar uma subtupla de C em* $\overline{R_2}$ *tal que os valores de S coincidam em* $\overline{R_1}$ *e* $\overline{R_2}$*. Se isso não ocorrer, a comparação é mal sucedida e o algoritmo é abortado.*

(e) *Coloca-se, na próxima posição livre do vetor, o valor da marca discriminante da chave de* $\overline{R_2}$*, referente à tupla comparada (com sucesso), de modo a eliminá-la das comparações posteriores.*

APÊNDICE B

ALGORITMO DE ARMAZENAMENTO DOS DADOS

Suponha uma estrutura de relação $\bar{R} = (R, r)$. *Os valores de r devem ser armazenados seguindo-se a ordem da esquerda para a direita e de cima para baixo sobre a disposição dos dados na tabela. Essa ordem deve respeitar a composição dos atributos, de modo que os valores das tuplas/subtuplas fiquem agrupados.*

Seja AP_ATUAL uma variável apontador que percorre a estrutura e que referencia o último registro em evidência. O registro apontado por AP_ATUAL é referenciado por ATUAL. Suponha, inicialmente, que o registro ATUAL seja o nó cabeça-de-lista.

Sejam **ID_AT**, **ID** *e* **ID1** *variáveis que conterão valores de identificadores, respectivamente, do identificador de ATUAL, do valor a ser inserido e um valor auxiliar.*

Seja **sig(I)** *o número de elementos significativos distintos de zero do identificador* **I**. *Seja* **v(p)** *o valor da posição* **p** *de um dado identificador.*

Para efeitos dos passos abaixo, quando se tratar de um identificador **I** *tal que* **sig(I) = 2**, *seus elementos significativos são considerados somente* <u>discriminantes</u> *e não "mais significativos" (exemplo: 120000). Nesse caso, as condições que excluem explicitamente o valor discriminante são consideradas "verdadeiras".*

Toda vez que o termo <u>registro</u> *for usado no algoritmo, deve-se supor que sua forma é a mesma apresentada em 4.3.*

Para a inserção dos valores de r, segundo a ordem já mencionada acima, os passos abaixo são repetidos até a exaustão dos

valores.

(a) *Seja* X *o valor a ser armazenado, pertencente ao atributo* A.

(b) *Seja* ID_AT *o valor do campo* <tid> *de ATUAL.*

(c) *Toma-se o identificador do atributo* A *em IDENT_R e o armazena-se em* ID.

(d) *Se* sig(ID) > sig(ID_AT) *e, supondo-se que o valor discriminante de* ID_AT *ocupe a posição* p, v(p) *de* ID_AT ≠ *de* v(p) *de* ID, *executam-se os subpassos seguintes:*

(d1) *Atribui-se* ID *a* ID1, *com seus discriminantes (valor e marca) substituídos por zeros. As demais marcas são substituídas pelos valores correspondentes de* ID_AT.

(d2) *Cria-se um registro* R *com* <tid> *contendo* ID1 *e* <num_tup> *contendo o valor zero.*

(d3) <ap_ant> *de* R *aponta ATUAL.*

(d4) <ap_atr> *de ATUAL passa a apontar* R.

(d5) R *passa a ser o registro ATUAL.*

(e) *Se* sig(ID) < sig(ID_AT) *e, com exceção do valor discriminante, os valores referentes aos atributos (posições pares) de* ID *forem* <u>iguais</u> *aos de* ID_AT, *então executa-se um dos seguintes subpassos. Suponha que o valor discriminante de* ID *ocupe a posição* p.

(e1) *Se* v(p) *de* ID *for* <u>maior</u> *que* v(p) *de* ID_AT, *então:*

(e11) *Avança-se* <ap_ant> *a partir de ATUAL até se encontrar um registro* REG *de identificador* I *tal que* sig(I) = sig(ID) *e cujo valor discriminante seja igual ao de* ID *decrementado de* 1.

(e12) REG *passa a ser o registro ATUAL.*

(e2) *Se* v(p) *de* ID *for* menor *que* v(p) *de* ID_AT, *então:*

(e21) *Avança-se* <ap_ant> *a partir de ATUAL até se encontrar um registro* REG *de identificador* I *tal que* sig(I) = sig(ID) *e cujo valor discriminante seja igual ao de* ID.

(e22) REG *passa a ser o registro ATUAL.*

(f) *Se* sig(ID) < sig(ID_AT) e *se, com exceção do valor discriminante, algum dos valores significativos das posições pares (referentes aos atributos) de* ID *for* maior *que o da correspondente posição em* ID_AT, *então:*

(f1) *Marca-se a posição* p *do primeiro valor diferente em ambos identificadores, percorrendo-os da esquerda para a direita.*

(f2) *Avança-se* <ap_ant> *a partir de ATUAL até se encontrar o último registro* REG *nesse sentido, tal que* v(p) *de seu identificador seja* igual *a* v(p) *de* ID.

(f3) REG *passa a ser o registro ATUAL.*

(f4) *Atualiza-se* ID_AT.

(f5) *Retorna-se ao passo* (d).

(g) *Se* sig(ID) = v(ID_AT), *deve-se executar um dos seguintes subpassos:*

(g1) *Se, com exceção do valor discriminante, todos os valores significativos de* ID *e* ID_AT *referentes aos atributos (posições pares) forem* iguais e *o valor discriminante de* ID *for* menor *que o de* ID_AT, *então:*

(g11) *Avança-se* <ap_ant> *a partir de ATUAL até se encontrar o primeiro registro, cujo identificador continha o valor discriminante* igual *ao de* ID, *na mesma posição* p..

(g12) REG *passa a ser o registro ATUAL.*

(g2) *Se, com exceção do valor discriminante, algum dos valores significativos referentes aos atributos (posições pares) em* ID *e* ID_AT *for diferente, nas correspondentes posições, então:*

(g21) *Seja* p *a posição do primeiro valor diferente da esquerda para a direita.*

(g22) ID1 *recebe* ID *com os discriminantes (valor e marca) iguais a zero; as demais marcas recebem os correspondentes valores de* ID_AT.

(g23) *Avança-se* <ap_ant> *a partir de ATUAL até que se encontre um registro* REG *nesse sentido, de identificador* I, *tal que* sig(I) = sig(ID1) *e cuja posição* p *contenha o valor* v(p) *de* ID *decrementado de* 1.

(g24) REG *passa a ser o registro ATUAL.*

(g25) *Atualiza-se a variável* ID_AT.

(g26) *Retorna-se ao passo* (d).

(h) *Atualiza-se o valor de* ID_AT *segundo o registro ATUAL vigente.*

(i) *Com exceção da marca discriminante, as marcas de* ID *recebem os mesmos valores das posições correspondentes em* ID_AT.

(j) *Se a marca discriminante de* ID, *que ocupa a posição* p, *corresponde a um valor igual a zero em* ID_AT, *executam-se os subpassos:*

(j1) *A marca referenciada acima recebe o valor* 1, *o que significa que* X *é o primeiro valor da primeira subtupla de* A.

(j2) *Cria-se um registro* REG *com* <tid> *contendo* ID *e* <val_arm> *contendo* X.

(j3) <ap_tup> *de ATUAL aponta* REG.