Refinamento de Esquemas e Formas Normais (Capítulo 19)

Doutorandos: Leonardo B. Oliveira e Carlos Senna
Professor: Dr. Geovane Cayres Magalhães
Banco de Dados (MO410)

Este documento é um resumo em português do Cap. 19 (*Schema Refinement and Normal Forms*) do Livro *Ramakrishnan and Gehrke, Database Management Systems, McGraw-Hill, 3rd. Edition, 2003.* Após o estudo do mesmo, esperamos que o aluno saiba lidar com as seguintes questões:

- Quais problemas são causados pelo armazenamento redundante de informações?
- O que são dependências funcionais?
- O que são formas normais e para que elas úteis?
- Quais os benefícios da FNBC e 3FN?
- Quais são as considerações ao se decompor relações em formas normais apropriadas?
- Aonde a normalização se encaixa no processo de projeto de banco de dados?
- Existem outras dependências que podem ser úteis no projeto de banco de dados?
- **Conceitos Chave:** redundância, anomalias de inserção, exclusão e atualização; dependência funcional, Axiomas de Armstrong, clausura de dependência, clausura de atributo; formas normais, FNBC, 3FN; decomposições, junção-sem-perda e preservação-de-dependência.

Considerações

Como parte da elaboração do resumo houve um trabalho de tradução do inglês para o português. Entendemos que traduções são tarefas difíceis e subjetivas – em especial em Ciência da Computação, em que não existe ainda um padrão de tradução para alguns termos. Desta forma, optamos por explicitar na Tabela 1, em ordem alfabética, a maneira com a qual certos termos foram traduzidos.

Inglês	**Português**
augmentation	expansibilidade
closure (of a set)	clausura (de um conjunto) [a]
delete	excluir
dependency-preservation	preservação-de-dependência
dependency-preserving	preservadora-de-dependência
design	projeto
hold	valer (é valido/verdade)
lossless	sem-perda (desprovida de perda)
lossless-join	junção-sem-perda
lossy	com-perda (causadora de perda)
minimal	minimal
must	**deve/devem** (em negrito)
reasoning about FDs	lucubrando sobre DFs [b]
sound	correto
string of letters	sequência de letras

[a] Note-se que parte da literatura se refere à *closure* como "fecho". Optamos por "clausura" após verificarmos qual o termo mais comum na comunidade de banco de dados.

[b] Embora em linguagem coloquial seja utilizada a expressão elucubrar, de acordo com o dicionário Aurélio [1], o termo certo é mesmo "lucubrar" (meditar, refletir, pensar. Do latim *lucubrare*).

Tabela 1: Tradução de termos do inglês para o português

Prefácio

O projeto de banco de dados conceitual nos dá um conjunto de esquemas relacionais e restrições de integridade (RIs) que podem ser considerados como um bom ponto de partida para o projeto de banco de dados final. Este projeto inicial deve ser refinado levando em consideração as RIs, desempenho e cargas de trabalho usuais. Neste documento, discutiremos como as RIs podem ser utilizadas para refinar esquemas conceituais que foram produzidos a partir da tradução de um projeto de modelo entidade-relacionamento (ER) em uma coleção de relações.

Concentraremos em uma importante classe de restrições chamadas *dependências funcionais*. Embora outros tipos de RI – *dependências multivaloradas* e *dependências de junção*, por exemplo – também ofereçam informações úteis (algumas vezes eles revelam informações que não podem ser detectadas ao se empregar somente dependências funcionais), eles não serão abordados neste documento [1].

O restante deste trabalho está organizado da seguinte forma. A Seção 1 é uma visão geral de refinamento de esquema. Introduzimos *dependências funcionais* na Seção 2. Na Seção 3 mostramos como tirar proveito de dependências funcionais para inferir adicionais dependências funcionais de um dado conjunto de dependências. Introduzimos as formas normais para relações na Seção 4; uma forma normal satisfeita por uma relação é vista como uma medida da redundância desta relação. Uma relação com redundância pode ser refinada ao ser *decomposta*, ou ser substituída por relações menores que possuam as mesmas informações, porém sem redundância. Discutimos decomposições e suas propriedades desejáveis na Seção 5, e mostramos como decompor as relações para que respeitem as formas normais mantendo suas características e dependências funcionais na Seção 6. Finalmente, na Seção 7 aplicamos todos os conceitos apresentados usando-os no refinamento de esquemas.

[1] Pare aqueles que desejam cobrir esses outros tipos de restrições, sugerimos a seguinte bibliografia [2] (repare que também existe a tradução para o português deste livro [3])

Sumário

Lista de Figuras

Lista de Tabelas

Refinamento de Esquemas e Formas Normais

Leonardo B. Oliveira, Carlos Senna

[1]Banco de Dados (MO410)
Professor Dr. Geovane Cayres Magalhães
Instituto de Computação/Universidade Estadual de Campinas

{leob,carlos.senna,geovane}@ic.unicamp.br

Resumo. *O projeto de banco de dados conceitual oferece um conjunto de esquemas relacionais e restrições de integridade (RIs) que podem ser considerados como um bom ponto de partida para o projeto de banco de dados final. Este projeto inicial deve ser refinado levando em consideração as RIs, desempenho e cargas de trabalho usuais. Neste documento, discutiremos como as RIs podem ser utilizadas para refinar o esquema conceitual. Daremos atenção especial a uma importante classe de restrições chamadas dependências funcionais.*

1. Introdução a Refinamento de Esquemas

Nesta seção, apresentamos uma visão geral dos problemas que podem ser resolvidos ao se empregar refinamento de esquemas. A principal causa desses problemas é o armazenamento redundante de informações. Embora a decomposição possa eliminar redundância, ela também pode levar à problemas e deve ser utilizada com cautela.

1.1. Problemas Causados pela Redundância

Armazenar a mesma informação de forma redundante, isto é, em mais de um lugar dentro de um banco de dados, pode levar a diversos problemas:

- **Armazenamento Redundante:** informação é armazenada mais de uma vez.
- **Anomalias de Atualização:** se uma cópia de um dado repetido é atualizada, uma inconsistência é criada a menos que todas as cópias sejam similarmente atualizada.
- **Anomalias de Inserção:** pode não ser possível armazenar uma certa informação a menos que alguma outra, não relacionada, seja também armazenada.
- **Anomalias de Exclusão:** pode não ser possível excluir uma certa informação sem perder uma outra não relacionada.

Considere a seguinte relação:

$$\text{Hourly\_Emps}(\underline{ssn}, name, lot, rating, hourly\_wages, hours\_worked)$$

Neste documento, omitimos o tipo do atributo para sermos sucintos, uma vez que nosso foco é no agrupamento de atributos dentro da relação. Em geral, também abreviamos o nome de um atributo para uma única letra e nos referimos a um esquema relacional por uma seqüência (*string*) de letras, uma por atributo. Por exemplo, nos referimos ao esquema Hourly_Emps como $SNLRWH$ (W denota o atributo $hourly\_wages$).

A chave para Hourly_Emps é ssn. Além disso, suponha que o atributo $hourly\_wages$ seja determinado pelo atributo $rating$ (ou seja, para um dado valor de $rating$, existe apenas um valor permitido de $hourly\_wages$). Então, se duas tuplas possuem o mesmo valor

para o atributo $rating$, a RI nos diz que as tuplas **devem** possuir também o mesmo valor para o atributo $hourly\_wages$. Tal RI é um exemplo de dependência funcional. Ela pode causar redundância na relação Hourly_Emps, como ilustrado na Tabela 2.

ssn	*name*	*lot*	*rating*	*hourly_wages*	*hours_worked*
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Tabela 2: Uma instância da relação Hourly_Emps

Essa redundância acarreta nos seguintes aspectos negativos já mencionados:

- **Armazenamento Redundante:** o valor 8 de $rating$ corresponde ao valor 10 de $hourly\_wages$ e essa associação é repetida três vezes.
- **Anomalias de Atualização:** $hourly\_wages$ na primeira tupla poderia ser atualizado sem que uma atualização similar fosse efetuada na segunda tupla.
- **Anomalias de Inserção:** não podemos inserir uma tupla na instância, a menos que saibamos o valor de $hourly\_ages$ para o valor de $rating$ da tupla a ser inserida.
- **Anomalias de Exclusão:** se excluirmos todas as tuplas com um certo valor de $rating$, perderemos a associação entre aquele valor de $rating$ e o valor de $hourly\_wages$ que ele determina. Por exemplo, na Tabela 2 ao excluirmos as tuplas cujo $name$ possui Smethurst e Guldu, perderemos a associação $rating$ 5$\rightarrow$ $hourly\_wages$ 7.

Note-se que é possível o emprego de valores *null* para lidar com alguns desses problemas. Porém, devido a restrições de espaço e porque valores *null* não oferecem soluções completas (não ajudam a eliminar redundância e/ou anomalias de atualização e são capazes de lidar apenas com algumas anomalias de inserção e exclusão) não abordaremos tal estratégia nesse documento.

Finalmente, gostaríamos que esquemas não permitissem redundância. Como isso nem sempre é possível, gostaríamos de que ao menos fôssemos capazes de identificar se um dado esquema permite ou não redundância. Desta forma, pelo menos trabalharíamos, cientes, com possibilidade de redundância.

1.2. Decomposições

Intuitivamente, redundância ocorre quando um esquema relacional força uma associação que não é natural. Dependências funcionais, bem como outras RIs, podem ser utilizadas para identificar tais situações e indicar refinamentos para o esquema. A idéia essencial é que muitos problemas decorrentes de redundância podem ser tratados pela substituição de uma relação por uma coleção de relações "menores".

Uma decomposição de um esquema relacional R consiste na substituição do esquema em questão por dois ou mais esquemas relacionais tais que:

1. cada um contenha um subconjunto de atributos R;
2. juntos incluam todos os atributos de R.

Intuitivamente, o objetivo é armazenar informações contidas em uma instância de R ao armazenar projeções desta instância. Por exemplo, podemos decompor Hourly_Emps em duas relações:

$$\text{Hourly\_Emps2}(\underline{ssn}, name, lot, rating, hours\_worked)$$

Wages($\underline{rating}, hourly\_wages$)

As instâncias dessas relações correspondentes à instância da relação Hourly_Emps da Tabela 2 é mostrada na Tabela 3.

ssn	*name*	*lot*	*rating*	*hours_worked*
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

rating	*hourly_wages*
8	10
5	7

Tabela 3: instâncias de Hourly_Emps2 **e** Wages

Note-se que, agora, para registrar um valor de $hourly\_wages$ para qualquer um de $rating$ basta adicionarmos uma tupla à Wages, mesmo que não exista nenhum empregado com o $rating$ em questão. Além disso, a mudança de um valor $hourly\_wages$ associado a um de $rating$ requer a atualização de apenas uma tupla de Wages.

1.3. Problemas relacionados a Decomposições

Caso não haja cautela, a decomposição de um esquema relacional pode atrapalhar mais do que ajudar. Logo, duas questões devem ser analisadas antes de optarmos pela decomposição:

1. Realmente precisamos decompor uma relação?
2. Quais problemas (caso existam) uma decomposição pode causar?

Para ajudar a responder à primeira questão várias *formas normais* foram propostas para relações. Se uma dada relação está em um dessas formas normais, sabemos que certos tipos de problemas não podem ocorrer. Considerar a forma normal de um dado esquema relacional pode ajudar na decisão de decompô-lo ou não.

Em relação à segunda questão, duas propriedades de decomposição são particularmente importantes. A propriedade *junção-sem-perda* possibilita-nos reaver qualquer instância de uma relação decomposta a partir das instâncias correspondentes das relações menores. A propriedade de *preservação-de-dependência* nos possibilita impor qualquer restrição na relação original pela simples imposição de algumas restrições em cada uma das relações menores. Em outras palavras, não precisamos realizar junções de relações menores para checar se uma restrição da relação original é ou não violada.

Do ponto de vista de desempenho, requisições sobre a relação original podem requerer a junção das relações decompostas. Se tais requisições são freqüentes, a decomposição da relação pode não ser factível. Neste caso, pode-se optar por "conviver" com alguns problemas de redundância para não perder em desempenho. Todavia, é importante estarmos cientes dos problemas causados por tal redundância, de maneira a tentar contorná-los com formas alternativas (adicionando algumas checagens ao código da aplicação, por exemplo).

Nosso objetivo neste documento é explanar conceitos e diretrizes de projeto, baseados na teoria de dependências funcionais, que orientem um projetista a identificar formas normais, propriedades de decomposição, etc.

2. Dependências Funcionais

Uma **dependência funcional** (DF) é um tipo de RI que generaliza o conceito de *chave*. Seja R um esquema relacional e X e Y conjuntos não vazios de atributos em R. Diz-se que uma instância r de R satisfaz a DF $X \rightarrow Y$[1] se o seguinte vale para todo par de tuplas t_1 e t_2 em r:

$$\text{Se } t_1.X = t_2.X, \text{ então } t_1.Y = t_2.Y$$

Usamos a notação $t_1.X$ para referirmos à projeção da tupla t_1 nos atributos em X. Uma DF $X \rightarrow Y$ essencialmente diz que se duas tuplas acordam nos valores dos atributos de X, então elas **devem** também acordar nos valores dos atributos de Y.

A Tabela 4 ilustra o significado de $AB \rightarrow C$ ao mostrar uma instância que satisfaz essa dependência. As primeiras duas tuplas mostram que uma DF não é o mesmo que uma restrição de chave: embora a DF não seja violada, AB não é uma chave da relação. A terceira e quarta tuplas ilustram que se duas tuplas diferem no campo A ou no B, elas podem se diferir no campo C sem violar a DF.

A	B	C	D
a1	b1	c1	d1
a1	b1	c1	d2
a1	b2	c2	d1
a2	b1	c3	d1

Tabela 4: Uma instância que satisfaz $AB \rightarrow C$

Lembre-se que uma instância *legal* de uma relação deve satisfazer todas as RIs especificadas, inclusive as DFs. Ao examinar uma instância de uma relação, devemos ser capazes de dizer que uma certa DF não vale. No entanto, não podemos nunca deduzir que uma DF certamente vale ao examinar uma ou mais instâncias de uma relação, porque uma DF, como outras RIs, é uma afirmação sobre *todas* possíveis instâncias de uma relação.

Uma restrição de chave é um caso especial de uma DF em que os atributos de uma chave exercem o papel de X e o conjunto de todos os atributos na relação exercem o papel de Y. É importante notar, todavia, que a definição de uma DF não requer que o conjunto X seja minimal; a condição de "minimalidade" adicional **deve** ser satisfeita caso X seja uma chave.

3. Lucubrando sobre DFs

Dado um conjunto de DFs que valem sobre um esquema relacional R, é comum várias outras DFs valerem também sobre R. Como exemplo, considere o seguinte:

$$\text{Workers}(\underline{ssn}, name, lot, did, since)$$

Suponha que a DF $did \rightarrow lot$ vale. Sabemos que $ssn \rightarrow did$ também vale, uma vez que *ssn* é a chave. Logo, em quaisquer instâncias legais de Workers, se duas tuplas possuem o mesmo valor de *ssn*, elas devem também ter o mesmo valor *lot* (pela primeira DF); e devido ao fato delas terem o mesmo valor de *did*, elas também **devem** ter o mesmo valor de *lot* (pela segunda DF). Dessa forma, a DF $ssn \rightarrow lot$ também vale para Workers.

[1] $X \rightarrow Y$ é lido como X *funcionalmente determina* Y, ou simplesmente X determina Y.

Dizemos que uma DF f **é implicada** por um dado conjunto F de DFs se f vale em toda instância de relação que satisfaz todas dependências em f; em outras palavras, f vale sempre que todas DFs em F valem. Note-se que não é suficiente para f valer em *alguma* instância que satisfaça todas as dependências em F, f deve valer para *todas* as instâncias que satisfaçam todas as dependências em F.

3.1. Clausura de um Conjunto de DFs

O conjunto de todas as DFs implicadas por um dado conjunto F de DFs é chamado de **clausura de F** e é denotado por F^+. Uma importante questão é como inferirmos, ou computarmos, o clausura de um dado conjunto F de DFs. A resposta é simples e elegante. As três regras seguintes, chamadas de **Axiomas de Armstrong** [4], podem ser aplicadas repetidamente para se inferir todas as DFs implicadas por um conjunto F de DFs. Utilizamos X, Y e Z para denotar conjuntos de atributos sobre um esquema relacional R:

- **Reflexibilidade:** Se $X \supseteq Y$, então $X \rightarrow Y$.
- **Expansibilidade:** Se $X \rightarrow Y$, então $XZ \rightarrow YZ$ para qualquer Z.
- **Transitividade:** Se $X \rightarrow Y$ e $Y \rightarrow Z$, então $X \rightarrow Z$.

Theorem 1. *Os Axiomas de Armstrong são* **corretos** *, no sentido que eles geram apenas DFs em F^+ quando aplicados ao conjunto* F *de DFs. Eles são também* **completos***, no sentido que a aplicação repetida dessas regras irão gerar todas as DFs na clausura F^+.*

Como conseqüência dos axiomas, temos as seguintes regras adicionais:

- **União:** Se $X \rightarrow Y$ e $X \rightarrow Z$, então $X \rightarrow YZ$.
- **Decomposição:** Se $X \rightarrow YZ$, então $X \rightarrow Y$ e $X \rightarrow Z$.

Como exemplo, suponha o seguinte:

Contracts($\underline{contractid}, supplierid, projectid, deptid, partid, qty, value$)

Denotamos o esquema de Contracts como $CSJDPQV$. O significado de um tupla é que o contrato com *contractid C* é um acordo que o fornecedor *S (supplierid)* irá suprir Q itens de parte *P (partid)* para projeto *J (projectid)* associado com departamento *D (deptid)*; o valor V desse contrato é igual à *value*.

É dado que as seguintes RIs valem:

1. O *id* de contrato C é uma chave: $C \rightarrow CSJDPQV$.
2. Um projeto compra uma dada parte usando um único contrato: $JP \rightarrow C$.
3. Um departamento compra no máximo uma parte de um fornecedor: $SD \rightarrow P$.

Várias DFs adicionais valem no clausura do conjunto de DFs dadas. Como exemplos citamos:

- A partir de $JP \rightarrow C$, $C \rightarrow CSJDPQV$ e transitividade, inferimos $JP \rightarrow CSJDPQV$.
- A partir de $SD \rightarrow P$ e expansibilidade, inferimos $SDJ \rightarrow JP$.
- A partir de $C \rightarrow CSJDPQV$, usando decomposição, podemos inferimos $C \rightarrow C$, $C \rightarrow S$, $C \rightarrow J$ e assim por diante.

$clausura = X$;
repita até que não ocorram mais mudanças: {
    se existe uma DF $U \rightarrow V$ em F tal que $U \subseteq clausura$,
        então conjunto $clausura = clausura \cup V$
}

Figura 1: Computando a clausura de atributo do conjunto de atributos X

3.2. Clausura de Atributo

Desejam-se apenas checar se uma dada dependência, digamos $X \rightarrow Y$ está na clausura de um conjunto F de DFs, podemos fazê-lo eficientemente sem computar F^+ da seguinte forma. Primeiro computamos a **clausura de atributo** X^+ em relação a F, que é o conjunto de atributos de A tal que $X \rightarrow A$ pode ser inferido empregando os Axiomas de Armstrong. O algoritmo para computar a clausura de atributo de um conjunto X de atributos é descrito na Figura 1.

Theorem 2. *O algoritmo mostrado na Figura 1 computa a clausura de atributo X^+ do conjunto de atributos X em relação ao conjunto de DFs F.*

4. Formas Normais

Dado um esquema relacional, é preciso decidir se ele é um bom projeto ou se é necessário decompô-lo em relações menores. Tal decisão deve ser guiada pelo entendimento de quais os problemas, caso exista algum, existem no esquema em questão. Para tal, várias **formas normais** [5] foram propostas. Se um esquema relacional está em alguma dessas formas normais, sabe-se que certos tipos de problemas não podem ocorrer.

As formas normais baseadas em DFs são a *primeira forma normal* (1FN), a *segunda forma normal* (2FN), a *terceira forma norma* (3FN) e a *forma normal de Boyce-Codd* (FNBC). Note-se que toda relação na FNBC está também na 3FN, toda relação na 3FN está também na 2FN e toda relação na 2FN está também na 1FN. Uma relação está na 1FN se cada um dos campos contém apenas valores atômicos, isto é, sem listas ou conjuntos. A principal importância da 2FN é de caráter histórico. Já a 3FN e a FNBC são importantes do ponto de vista de projeto de banco de dados.

No restante desta seção discutiremos a FNBC [6] e em seguida a 3FN.

4.1. Forma Normal de Boyce-Codd

Seja R um esquema relacional, F o conjunto de DFs que valem em R, X o subconjunto de atributos de R e A um atributo de R. R está na $FNBC$ se, para toda a DF $X \rightarrow A$ em F, uma das seguintes afirmações é valida:

- $A \in X$; isto é, é uma DF trivial, ou
- X é uma superchave.

Intuitivamente, as únicas dependências não triviais em uma relação na FNBC são aquelas que uma chave determina um ou mais atributos. Dessa forma, cada tupla pode ser vista como uma entidade ou relacionamento identificado por uma chave e descrita pelos atributos restantes. Se utilizarmos elipses para denotar atributos ou conjunto de atributos e desenharmos arcos para indicar DFs, uma relação na FNBC possui a estrutura ilustrada na Figura 2 (para simplificar, consideramos apenas uma chave. Caso existam várias chaves candidatas, cada chave candidata pode exercer o papel de chave na figura, com os outros atributos sendo aqueles que não pertencem a chave candidata escolhida).

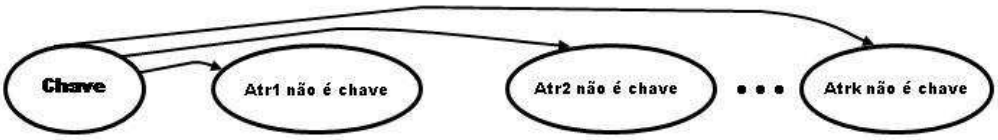

Figura 2: DFs de uma relação na FNBC

A FNBC assegura que nenhuma redundância pode ser detectada usando apenas informação de DF. Ela é então a forma normal mais desejada (do ponto de vista de redundância) se levarmos em conta apenas informação de DF. Isso está ilustrado na Tabela 5.

X	Y	A
x	y_1	a_1
x	y_2	?

Tabela 5: Instância ilustrando a FNBC

A figura mostra uma instância de relação com 3 atributos: X, Y e A. Existem 2 tuplas com o mesmo valor na coluna X. Agora, suponha que soubéssemos que essa instância satisfaz uma DF $X \rightarrow A$. Note-se que uma das tuplas possui o valor a na coluna A. O que podemos inferir sobre o valor na coluna A da segunda tupla? Usando a DF podemos concluir que a segunda tupla também tem o valor a em sua coluna. (Note-se que isso é realmente a única inferência que podemos fazer sobre valores nos campos das tuplas ao usarmos DFs.)

Mas não seria isso um exemplo de redundância? Parece que armazenamos o valor a duas vezes. Pode tal situação ocorrer em uma relação de FNBC? A resposta é NÃO! Se essa relação está na FNBC, porque A é distinto de X, então X tem que ser uma chave. (Caso contrário, a DF $X \rightarrow A$ violaria a FNBC). Por sua vez, se X é uma chave, então $y_1 = y_2$, ou seja, as duas tuplas são idênticas. Uma vez que uma relação é definida como sendo um *conjunto* de tuplas, não podemos ter duas cópias da mesma tupla e a situação da Tabela 5 não pode ocorrer.

Dessa forma, se uma relação está na FNBC, todo campo de todas tuplas possui uma informação que não pode ser inferida (usando apenas DFs) a partir dos valores em todos outros campos (de todas as tuplas) na instância da relação.

4.2. Terceira Forma Normal

Seja R um esquema relacional, F o conjunto de DFs que valem em R, X o subconjunto de atributos de R e A um atributos de R. R está na 3FN se, para toda a DF $X \rightarrow A$ em F, uma das seguintes afirmações é valida:

- $A \in X$; isto é, é uma DF trivial, ou
- X é uma superchave, ou
- A é parte de alguma chave para R

A definição da 3FN é similar a da FNBC, com exceção da terceira condição. Toda relação na FNBC está também na 3FN. Para entender a terceira condição, lembre-se que uma chave de uma relação é um conjunto de atributos *minimal* que determinam inequivocamente todos outros atributos. A deve fazer parte de uma chave (de qualquer chave, caso haja mais de uma). Não é suficiente para A fazer parte de uma superchave, porque a última condição é satisfeita por todo atributo! É sabido que o problema de encontrar

todas as chaves de uma relação é NP-completo e, por sua vez, o problema de determinar se um esquema relacional está na 3FN também o é.

Suponha que a dependência $X \rightarrow A$ cause uma violação da 3FN. Existem duas possibilidades:

- X *é um subconjunto próprio de alguma chave* K. Tal dependência é algumas vezes chamada de **dependência parcial**. Neste caso, armazenamos pares (X, A) de forma redundante.
- X *não é um subconjunto próprio de qualquer chave*. Tal dependência é algumas vezes chamada de **dependência transitiva**, pois significa que temos uma cadeia de dependências $K \rightarrow X \rightarrow A$. O problema é que não podemos associar um valor de X com um valor de K a menos que associemos também um valor de A com um valor de X. Essa condição leva a anomalias de inserção, exclusão e atualização.

Dependências parcias e transitivas estão ilustradas na Figura 3 e Figura 4, respectivamente. Note-se que na Figura 4 o conjunto de atributos pode ou não pode ter alguns atributos em comum com **chave**; o diagrama deveria ser interpretado como indicador apenas de que X não é um subconjunto de **chave**.

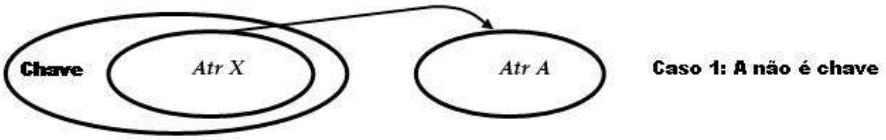

Figura 3: Dependências Parciais

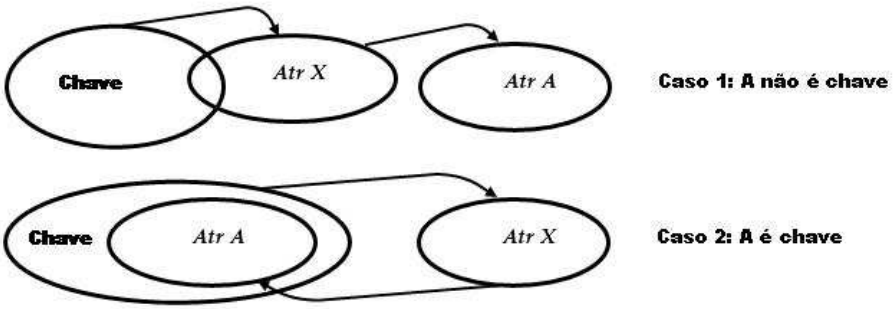

Figura 4: Dependências Transitivas

A motivação para a 3FN é bem técnica. Com exceção de certas dependências envolvendo atributos de chave, podemos assegurar que todo esquema relacional pode se decomposto em uma coleção de relações na 3FN usando apenas decomposições que possuem certas propriedades desejáveis (Seção 5). Tal garantia não existe para relações na FNBC; a definição da 3FN possui menos requerimentos que a de FNBC justamente para tornar essa garantia possível.

Ao contrário da FNBC, é importante salientar que a 3FN permite certo grau de redundância. Os problemas associados com as dependências parcial e transitiva persistem na 3FN se existe um dependência não trivial $X \rightarrow A$ e X não é uma super chave.

Finalmente, lembramos que a definição da 2FN é essencialmente que dependências parciais não são permitidas. Logo, se a relação está na 3FN (que não permite nem dependências parciais, nem transitivas), está também na 2FN.

5. Propriedades de Decomposições

Decomposição é uma ferramenta que nos permite eliminar redundância. É importante, todavia, checar se a decomposição não introduz novos problemas. Particularmente, devemos checar se uma decomposição nos permite reaver a relação original e se ela nos permite checar RIs eficientemente [7]. Nesta seção discutiremos essas propriedades.

5.1. Decomposições Junção-Sem-Perda

Seja R um esquema relacional e F o conjunto de DFs que valem em R. Uma decomposição de R em dois esquemas com conjunto de atributos X e Y é dita ser uma decomposição junção-sem-perda em relação a F se, para toda instância r de R que satisfaz as dependências em F, $\pi_X(r) \bowtie \pi_Y(r) = r$. Em outras palavras, podemos reaver a relação original a partir das relações decompostas.

Essa definição pode ser facilmente estendida para cobrir a decomposição de R em mais que duas relações. É fácil de ver que $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$. Em geral, entretanto, a outra direção não vale. Se tomarmos projeções da relação e as recombinarmos usando junção natural, é comum obtermos algumas tuplas que não estavam na relação original. Essa situação é ilustrada na Tabela 6.

S	P	D
s_1	p_1	d_1
s_2	p_2	d_2
s_3	p_1	d_3

S	P
s_1	p_1
s_2	p_2
s_3	p_1

P	D
p_1	d_1
p_2	d_2
p_1	d_3

S	P	D
s_1	p_1	d_1
s_2	p_2	d_2
s_3	p_1	d_3
s_1	p_1	d_3
s_3	p_1	d_1

Tabela 6: Instâncias ilustrando decomposições com-perda; da esquerda para direita: instância r, $\pi_{SP}(r)$, $\pi_{PD}(r)$, $\pi_{SP}(r) \bowtie \pi_{PD}(r)$

Ao trocarmos a instância r mostrada na Tabela 6 com as instâncias $\pi_{SP}(r)$ e $\pi_{PD}(r)$, perdemos informação. Em particular, suponha que as tuplas em r denotem relacionamentos. Não podemos mais afirmar que os relacionamentos (s_1, p_1, d_3) e (s_3, p_1, d_1) não valem. A decomposição de esquema SPD em SP e PD é então com-perda (*lossy*) se a instância r mostrada na figura é *legal*, ou seja, se essa instância poderia ocorrer na empresa que está sendo modelada.

Todas as decomposições utilizadas para eliminar redundância **devem** *ser sem-perda (lossless).* O seguinte teste é simples e muito útil.

Theorem 3. *Seja R um esquema relacional F o conjunto de DFs que valem em R. A decomposição de R em relações com conjunto de atributos R_1 e R_2 é sem-perda, se e somente se, F^+ contém ou o DF $R_1 \cap R_2 \rightarrow R_1$ ou o DF $R_1 \cap R_2 \rightarrow R_2$.*

Em outras palavras, os atributos comuns a R_1 e R_2 devem conter uma chave para R_1 ou uma chave para R_2. Se uma relação é decomposta em mais que duas relações, um algoritmo eficiente (de tempo polinomial no tamanho do conjunto de dependência) pode ser empregado para testar se a decomposição é ou não sem-perda, mas não iremos discutí-lo.

Considere a relação Hourly_Emps novamente. Ela possui os atributos $SNLRWH$ e a DF $R \rightarrow W$ causa a violação da 3FN. Tratamos essa violação decompondo a relação em $SNLRH$ e RW. Uma vez que R é comum a ambas relações decompostas e $R \rightarrow W$ vale, essa decomposição é junção-sem-perda.

Este exemplo ilustra uma observação que é conseqüência do Teorema 3: se uma DF $X \rightarrow Y$ vale sobre uma relação R e $X \cap Y$ é vazio, a decomposição de R em $R - Y$ e XY é sem-perda.

X aparece em ambos $R - Y$ (uma vez que $X \cap Y$ é vazio) e XY, e é uma chave para XY.

Outra importante observação, que sustentamos sem provar, tem haver com decomposições repetidas. Suponha que uma relação R é decomposta em R_1 e R_2 através de uma decomposição junção-sem-perda, e que R_1 é decomposto em R_{11} e R_{12} por meio de outra decomposição junção-sem-perda. Então, a decomposição de R em R_{11}, R_{12} e R_2 é junção-sem-perda; ao juntar R_{11} e R_{12}, podemos reaver R_1, e ao juntar R_1 e R_2, podemos reaver R.

5.2. Decomposição Preservadora-De-Dependência

Outra propriedade importante no projeto de banco de dados relacional é a preservação-da-dependência. Quando ocorre uma atualização no banco, o sistema deve checar se a mesma criará uma relação *ilegal*. Se checarmos de modo eficiente tais atualizações, poderemos projetar esquemas capazes de validar atualizações sem que junções sejam realizadas.

Considere a relação Contracts com os atributos $CSJDPQV$ da Seção 3.1. As DFs são as seguintes: $C \rightarrow CSJDPQV$, $JP \rightarrow C$ e $SD \rightarrow P$. Devido a fato que de SD não é uma chave, a dependência $SD \rightarrow P$ viola a FNBC.

Podemos decompor Contracts em duas relações com esquemas $CSJDQV$ e SDP para lidar com essa violação; essa decomposição é junção-sem-perda. Há um problema sutil, porém. Podemos facilmente impor a RI $JP \rightarrow C$ quando uma tupla é inserida na relação Contracts assegurando que nenhuma tupla possui os mesmos valores de JP (como a tupla inserida), mas diferentes valores de C. Uma vez decomposta a relação Contracts em $CSJDQV$ e SDP, impor essa restrição requer uma junção cara de duas relações sempre que uma tupla for inserida em $CSJDQV$. Dizemos, então, que essa decomposição não é preservadora-de-dependência.

Intuitivamente, uma decomposição preservadora-de-dependência nos permite impor todas as DFs ao examinar uma única instância de relação em cada inserção ou modificação de tupla (observe que exclusões não causam violação de DFs). A fim de definirmos precisamente o que é uma decomposição preservadora-de-dependência, devemos antes introduzir o conceito de projeção de DFs.

Seja R um esquema relacional que é decomposto em dois esquemas com conjuntos de atributos X e Y, e seja F o conjunto de DFs sobre R. A **projeção** de F em X é o conjunto de DFs na clausura F^+ (não apenas F!) que envolve apenas atributos em Y. Denotamos a projeção de F nos atributos X como F_X. Note-se que a dependência $U \rightarrow V$ em F^+ está em F_X apenas se todos atributos em U e V estiverem em X.

A decomposição de um esquema R com DFs F em esquemas com conjuntos de atributos X e Y é **preservadora-de-dependência** se $(F_X \cup F_Y)^+ = F^+$. Ou seja, se pegarmos as dependências em F_X e F_Y e computarmos a clausura da união, reavemos todas dependências da clausura de F. Logo, precisamos impor apenas as dependências em F_X e F_Y; todas as DFs em F^+ são então satisfeitas. Para impormos F_X, precisamos examinar apenas a relação X (ao modificarmos essa relação). Analogamente, para impormos F_Y, precisamos examinar apenas a relação Y.

6. Normalização

Após ver os conceitos e regras das formas normais e decomposição no projeto de bancos de dados, estudaremos a seguir algoritmos para converter relações para a FNBC ou 3FN.

A seguir apresentamos um algoritmo para decompor a relação R com um conjunto F de DFs numa coleção de relações na FNBC:

1. Suponhamos que R não está na FNBC. Seja $X \subset R$, A um atributo simples em R e $X \rightarrow A$ uma DF que viola a FNBC. Decomponha R em $R - A$ e XA.
2. Se $R - A$ ou XA não estão na FNBC, reaplique a decomposição recursivamente.

$R - A$ denota o conjunto de atributos não A em R e XA é $X \cup A$. Desde que $X \rightarrow A$ viola FNBC, não é uma dependência trivial, pois A é um atributo simples, e A não está em X, isto é, $X \cup A$ é vazio, então cada decomposição do passo 1 é uma junção-sem-perda. Como as decomposições resultantes contém estritamente subconjuntos exclusivos, o processo termina levando a uma coleção de relações que estão na FNBC.

Para exemplificar o uso do algoritmo vamos considerar a relação Contracts com os atributos $CSJDPQV$ onde C é chave e as DFs $JP \rightarrow C$ e $SD \rightarrow P$. Usando $SD \rightarrow P$ como base, decompomos Contracts em SDP e $CSJDQV$. Supondo que temos também a restrição que cada projeto tenha um único fornecedor $J \rightarrow P$, a relação $CSJDQV$ não está na FNBC. Então decompomos $CSJDQV$ em JS e $CJDQV$. Assim, os esquemas SDP, JS e $CJDQV$ estão na FNBC e representam a decomposição junção-sem-perda de $CSJDPQV$, conforme ilustra a Figura 5.

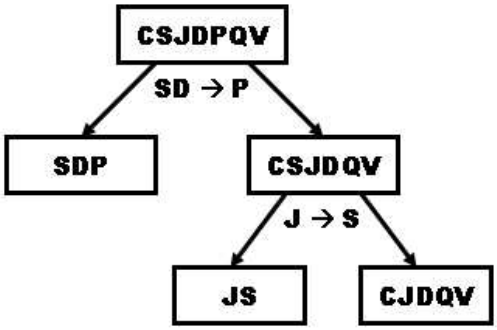

Figura 5: Decomposição de $CSJDQV$ em SDP, JS e $CJDQV$

Essa decomposição não preserva a dependência, pois $JP \rightarrow C$ não pode ser alcançado sem uma junção.

Outro aspecto importante a considerar quando estamos decompondo na FNBC é a ordem na qual são analisadas as DFs. No exemplo podemos fazer a decomposição de $CSJDPQV$ em JS e $CJDPQV$ baseado na DF $J \rightarrow S$. As dependências que restam sobre $CJDPQV$ são $JP \rightarrow C$ e a dependência de chave $C \rightarrow CJDPQV$. Como JP é uma chave, $CJDPQV$ está na FNBC. Assim JS e $CJDPQV$ representam uma decomposição junção-sem-perda de Contracts dentro das relações FNBC.

As vezes pode simplesmente não existir uma decomposição na FNBC que preserve a dependência. Por exemplo, considere a relação SBD que denota que o marinheiro (S) reserva o barco (B) numa data (D). Se temos as DFs $SB \rightarrow D$ e $D \rightarrow B$, SBD não está na FNBC, porque D não é uma chave. Se tentarmos decompô-la, não estaremos preservando a dependência $SB \rightarrow D$.

6.1. Decomposição na 3FN

Claramente o algoritmo para decomposição junção-sem-perda na FNBC pode ser usado na 3FN (a decomposição será feita com um número menor de etapas). Mas essa abordagem não garante a preservação da dependência. Antes de analisarmos essa questão vamos introduzir um novo conceito.

6.1.1. Cobertura minimal para um conjunto de DFs

A cobertura minimal para um conjunto F de DFs é um conjunto G de DFs tal que:

1. Toda dependência em G está na forma $X \rightarrow A$, onde A é um atributo simples.
2. A clausura F^+ é igual para a clausura G^+.
3. Se obtemos um conjunto H de dependências a partir de G através da eliminação de uma ou mais dependências, ou pela eliminação de atributos a partir da dependência G, então $F^+ \neq H^+$.

Intuitivamente a cobertura minimal para um conjunto F de DFs é um conjunto equivalente de dependências que é minimal em dois aspectos: (1) Toda dependência é a menor possível, isto é, cada atributo no lado esquerdo é necessário, e o lado direito é um atributo simples. (2) Toda dependência é necessária para a clausura ser igual a F^+.

Por exemplo, tomemos F como sendo o conjunto de dependências:

$$A \rightarrow B, ABCD \rightarrow E, EF \rightarrow G, EF \rightarrow H \text{ e } ACDF \rightarrow EG$$

Decompomos $ACDF \rightarrow EG$ em $ACDF \rightarrow E$ e $ACDF \rightarrow G$. Mas $ACDF \rightarrow G$ está implícita em $A \rightarrow B$, $ABCD \rightarrow E$ e $EF \rightarrow G$. Portanto pode ser eliminada. De forma similar $ACDF \rightarrow E$ também pode ser eliminada. Como $A \rightarrow B$, podemos substituir $ABCD \rightarrow E$ por $ACD \rightarrow E$, e dessa forma temos a seguinte cobertura minimal para F:

$$A \rightarrow B, ACD \rightarrow E, EF \rightarrow G \text{ e } EF \rightarrow H$$

Como vimos no exemplo, o algoritmo para obtermos a cobertura minimal é:

1. Colocar as DFs na forma padrão.
2. Minimizar o lado esquerdo de cada DF.
3. Eliminar as redundâncias.

6.1.2. Decomposição Preservadora-De-Dependência na 3FN

Tomemos R sendo a relação com um conjunto F de DFs que é uma cobertura minimal, e tomemos $R_1, R_2, ..., R_n$ como sendo uma decomposição junção-sem-perda de R. Para $1 \leq i \leq n$, supomos que cada R_i está na 3FN e F_i denota a projeção de F nos atributos de R_i, como segue:

- A identidade do conjunto N de dependências em F não está preservada, isto é, não está inclusa na clausura das uniões de $F_i s$.
- Para cada DF $X \rightarrow A$ em N, crie uma relação XA e adicione-a na decomposição de R.

Como exemplo, vamos considerar a relação Contracts com os atributos $CSJDPQV$ e as DFs $JP \rightarrow C$, $SD \rightarrow P$ e $J \rightarrow S$. Se decompomos $CSJDPQV$ em SDP e $CSJDQV$, temos que SDP está na FNBC, mas $CSJDQV$ não está na 3FN. Se decompomos $CSJDQV$ em JS e $CJDQV$, temos que as relações SDP, JS e $CJDQV$ estão na 3FN, e são decomposição junção-sem-perda. Entretanto a dependência $JP \rightarrow C$ não foi preservada. Resolvemos o problema adicionado a relação CJP na decomposição.

7. Refinamento de Esquemas no Projeto do Banco de Dados

Vamos ver como as abordagens para normalização podem ser usadas na prática.

7.1. Restrições num Conjunto de Entidades

Considere a relação Hourly_Emps, onde a restrição é que o atributo *ssn* é a chave:

$$\{ssn\} \rightarrow \{ssn, name, lot, rating, hourly\_wages, hours\_worked\}$$

Vamos escrever essa DF como $S \rightarrow SNLRWH$, e adicionar a DF na qual *hourly_wages* é determinada pelo atributo *rating* $R \rightarrow W$. Essa DF leva a redundância de armazenamento, e não pode ser expressa nos termos do modelo ER. Somente DFs que determinam todos os atributos da relação (isto é, são restrições de chave), podem ser expressas no modelo ER. Dessa forma nós não podemos identificá-la quando considerarmos Hourly_Emps como uma entidade durante a modelagem ER.

7.2. Restrições num Conjunto de Relacionamentos

O exemplo acima mostra como DFs podem ajudar a refinar decisões subjetivas tomadas durante o projeto ER. Vamos supor que temos as entidades partes (P), fornecedores (S) e departamentos (D), relacionadas por Contracts. Usaremos o esquema $CQPSD$ para Contracts, onde Q indica a quantidade de partes (P) compradas de um fornecedor (S) por um departamento (D). Temos ainda a restrição de que o departamento compra uma parte de cada fornecedor $SD \rightarrow P$. Novamente temos redundância, e podemos resolvê-la decompondo Contracts em $CQSD$ e SDP. Intuitivamente a relação SDP grava as partes adquiridas pelo departamento do fornecedor, e a relação $CQSD$ grava as informações adicionais do contrato. Note que não é fácil modelar naturalmente a relação $CQSD$ como entidade ou relacionamento na modelagem ER.

7.3. Identificando Atributos de Entidades

Esse exemplo mostra como o exame cuidadoso das DFs pode ajudar a entender os relacionamentos e mostrar que atributos podem ser associados a entidades de forma errada durante o projeto ER. O diagrama ER da Figura 6 apresenta o conjunto de relacionamentos Works_In acrescido de uma restrição de chave na qual um empregado (E) pode trabalhar em mais de um departamento (D).

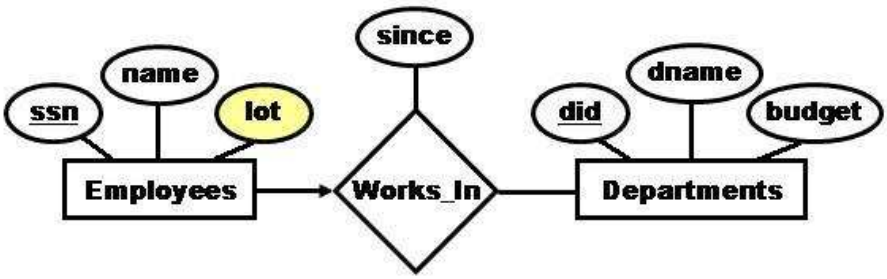

Figura 6: O conjunto de relacionamentos Works_In

O diagrama pode ser convertido em dois relacionamentos:

$$\text{Workers}(\underline{ssn}, name, lot, did, since)$$

$$\text{Departments}(\underline{did}, dname, budget)$$

Agora supomos que todos os empregados de um departamento estão associados a um mesmo lote. Essa restrição não está expressa no diagrama ER da Figura 6. É um outro

exemplo de DF: $did \rightarrow lot$. Essa redundância pode ser eliminada decompondo Workers em duas novas relações:

$$\text{Workers2}(\underline{ssn}, name, did, since)$$

$$\text{Depto\_Lots}(\underline{did}, lot)$$

Examinando as relações Departments e Dept_Lots vemos que ambas estão baseadas na mesma chave e descrevem a mesma Entidade. Então deslocamos o atributo *lot* para a relação Departments e refinamos o diagrama, conforme apresentado na Figura 7 a seguir.

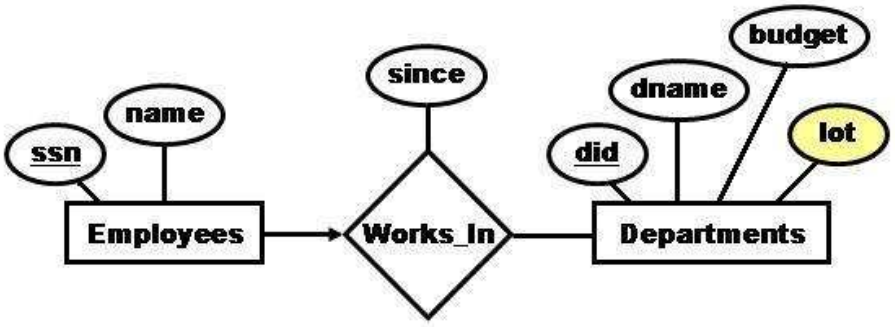

Figura 7: O conjunto de relacionamentos Works_In **refinado**

E o novo diagrama refinado é transladado no modelo relacional pelas relações:

$$\text{Workers2}(\underline{ssn}, name, did, since)$$

$$\text{Departments}(\underline{did}, dname, budget, lot)$$

7.4. Identificando conjuntos de Identidades

Vamos usar novamente o esquema Reserves com os atributos S (marinheiro-sailor), que tem uma reserva para o B (barco) no dia D. A esses atributos adicionamos o atributo C (cartão de crédito), e vamos discutir como a DF pode ajudar a decidir quando um conceito pode ser modelado como Entidade ou como um Atributo.

Vamos supor que o marinheiro use apenas um cartão de crédito para as reservas. Essa restrição é expressa através da DF $S \rightarrow C$. Essa restrição indica uma redundância, uma vez que toda reserva armazena o número do cartão de crédito do marinheiro. A solução é decompor Reserves em duas relações com os atributos SBD e SC. Uma abordagem no projeto ER é introduzir a Entidade Credit_Cards, com o atributo *cardno* e o conjunto de relações Has_Card associando marinheiros (S) e Credit_Cards. Uma segunda opção de abordagem é fazer *cardno* como atributo do marinheiro (S), mas isso implica em que cada marinheiro pode ter apenas um cartão de crédito. O importante desse exemplo é ver que no projeto modelamos *cardno* como um atributo de Reserves e depois, levando em consideração as DFs, refinamos a tabela resultante.

Referências

[1] Aurélio Buarque de Holanda Ferreira. *Novo Dicionário da Língua Portuguesa*. Academia Brasileira de Letras, segunda edition, 1986.

[2] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems, 2nd Edition.*, chapter Relational Database Design Algorithms and Further Dependencies. Benjamin/Cummings, 1994.

[3] Ramez Elmasri and Shamkant B. Navathe. *Sistemas de Banco de Dados - Fundamentos e Aplicacoes*. LTC, 2002.

[4] William Ward Armstrong. Dependency structures of data base relationships. In *IFIP Congress*, pages 580–583, 1974.

[5] E. F. Codd. Normalized data base structure: A brief tutorial. In *ACM SIGFIDET Workshop on Data Description, Acess and Control*, 1971.

[6] E. F. Codd. Recent investigations into relational data base systems. In *IFP Congress*, 1974.

[7] Ronald Fagin. Normal forms and relational database operators. In *ACM SIGMOD Conference*, pages 153–160, 1979.