Nota: Este material representa uma cópia resumida do capítulo 19 do livro Sistemas de Gerenciamento Sistemas de Gerenciamento de Banco de Dados - Ramakrishnan Gehrke (3ª edição), versão disponível em pdf na internet.

Normalização

O processo de normalização, proposto inicialmente por Codd (1972a), leva um esquema de relação por uma série de testes para certificar se ele satisfaz certa **forma normal**. Há algumas regras para normalização do banco de dados. Cada **regra** é chamada de **forma normal**. Se a primeira regra for observada, diz-se que o banco de dados está na "primeira forma normal" (1FN). Se as três primeiras regras forem observadas, o banco de dados será considerado na "terceira forma normal" (3FN). Embora outros níveis de normalização sejam possíveis, a terceira forma normal é considerada o nível mais alto necessário para a maioria dos aplicativos. Depois, uma quarta forma normal (4FN) e uma quinta forma normal (5FN) foram propostas, com base nos conceitos de dependências multivaloradas e dependências de junção.

Normalização é o processo de organização de dados em um banco de dados. Isso inclui a criação de tabelas e o estabelecimento de relações entre essas tabelas de acordo com as regras projetadas para proteger os dados e tornar o banco de dados mais flexível, eliminando a redundância e a dependência inconsistente. A normalização de dados pode ser considerada um processo de analisar os esquemas de relação dados com base em suas **Dependência Funcionais** (**DFs)** e **chaves primárias** para conseguir as propriedades desejadas de **(1) minimização da redundância** e **(2) minimização das anomalias de inserção, exclusão e atualização:**

O armazenamento das mesmas informações de forma **redundante**, pode acarretar vários problemas:

- **Armazenamento redundante:** algumas informações são armazenadas repetidamente desperdiçando espaço em disco.
- **Anomalias de atualização:** se uma cópia de tais dados redundantes é atualizada, é gerada uma inconsistência, a não ser que todas as cópias sejam atualizadas de forma semelhante.
- **Anomalias de inserção:** pode não ser possível armazenar certas informações, a não ser que outras informações não relacionadas também sejam armazenadas.
- **Anomalias de exclusão:** pode não ser possível excluir certas informações sem perder também algumas outras informações não relacionadas.

Decomposições

Muitos problemas provenientes da redundância podem ser resolvidos substituindo-se uma relação por uma coleção de relações “menores”. Uma **decomposição** de um esquema de relação R consiste na substituição do esquema de relação por dois (ou mais) esquemas de relação, cada um contendo um subconjunto dos atributos de R e, juntos, incluindo todos os atributos presentes em R. Podemos decompor Funcion_Horistas em duas relações:

Funcion_Horistas2(cpf, nome, vaga, avaliação, horas_trabalhadas)
Salários(avaliação, horas_trabalhadas)

As instâncias dessas relações correspondentes à instância da relação Funcion_Horistas da Figura 19.1 aparece na Figura 19.2.

cpf	*nome*	*vaga*	*avaliação*	*horas_ trabalhadas*
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

avaliação	*horas_ trabalhadas*
8	10
5	7

Figura 19.2 Instâncias de Funcion_ Horistas2 e Salários.

Note que podemos registrar facilmente o salário/hora para qualquer avaliação, simplesmente adicionando uma tupla em Salários, mesmo que nenhum funcionário com essa avaliação apareça na instância corrente de Funcion_Horistas. Mudar o salário associado a uma avaliação envolve atualizar uma única tupla de Salários. Isso é mais eficiente do que atualizar várias tuplas (como no projeto original) e elimina a inconsistência em potencial.

A menos que tomemos cuidado, decompor um esquema de relação pode criar mais problemas do que resolver. Duas perguntas importantes devem ser feitas repetidamente:

- Precisamos decompor uma relação?
- Quais problemas (se houver) determinada decomposição causa?

Para ajudar com a primeira pergunta, várias formas normais foram propostas para relações. Se um esquema de relação está em uma dessas formas normais, sabemos que certos tipos de problemas não podem surgir. Considerar a forma normal de determinado esquema de relação pode nos ajudar a decidir se vamos decompô-lo ainda mais ou não. Se decidirmos que um esquema de relação deve ser decomposto ainda mais, devemos escolher uma decomposição em particular (isto é, uma coleção específica de relações menores para substituir a relação dada).

Com respeito à segunda pergunta, duas propriedades das decomposições têm particular interesse.

- A propriedade da **junção sem perda** nos permite recuperar qualquer instância da relação decomposta, a partir das instâncias correspondentes das relações menores.
- A propriedade da **preservação da dependência** nos permite impor qualquer restrição na relação original simplesmente impondo algumas restrições em cada uma das relações menores. Ou seja, não precisamos realizar junções das relações menores para verificar se uma restrição na relação original é violada.

Do ponto de vista do desempenho, consultas na relação original podem exigir que juntemos as relações decompostas. Se tais consultas são comuns, a penalidade sobre o desempenho da decomposição da relação pode não ser aceitável. Em algumas situações, a decomposição poderia realmente melhorar o desempenho. Isso acontece, por exemplo, se a maioria das consultas e atualizações examinam apenas uma das relações decompostas, que é menor do que a relação original.

Dependências Funcionais (DFs)

https://www.youtube.com/watch?v=7wPsS2uFBgA

Uma **Dependência Funcional (DF)** é um tipo de restrições de integridade (RI) que generaliza o conceito de chave. Seja R um esquema de relação e X e Y conjuntos não-vazios de atributos em R. Dizemos que uma instância r de R satisfaz a DF X -> Y (**X determina funcionalmente Y ou, simplesmente, X determina Y**) se o seguinte vale para todo par de tuplas t1 e t2 em r:

Se t1 . X = t2. X, então t1 . Y = t2.Y

- Como exemplo, considere: *Funcionários(cpf, nome, vaga, id_depto, desde)*
  Sabemos que cpf -> id_depto vale, pois cpf é a chave, e a DF id_depto -> vaga é dada como válida. Portanto, em qualquer instância válida de Funcionários, se duas tuplas têm o mesmo valor de cpf, elas devem ter o mesmo valor de id_depto (da primeira DF) e como elas têm o mesmo valor de id_depto, também devem ter o mesmo valor de vaga (da segunda DF). Portanto, a DF cpf->vaga também vale em Funcionários.

- Em uma tabela relacional, diz-se que uma coluna **C2 depende funcionalmente** de uma coluna **C1** (ou que a coluna **C1 determina a coluna C2**), **C1->C2,** quando, em todas linhas da tabela, para cada valor de C1 que aparece na tabela, aparece o mesmo valor de C2.

❖Exemplo:

✓Coluna **C** depende das colunas **A** e **B**.
✓Coluna **D** depende da coluna **A**

Dependências funcionais:

(A,B) → C
A → D

A	B	C	D
B	5	2	20
C	4	2	15
B	6	7	20
B	5	2	20
C	2	2	15
C	4	2	15
A	10	5	18
A	12	3	18
A	10	5	18
B	5	2	20
C	4	2	15
A	10	5	18
C	4	2	15

- Por exemplo, a Figura 15.7 mostra um estado em particular do esquema de relação ENSINA. Embora à primeira vista possamos pensar que Texto → Disciplina, não podemos confirmar isso a menos que saibamos que é verdadeiro para todos os estados legais possíveis de ENSINA. É, no entanto, suficiente demonstrar um **único contraexemplo** para **refutar** uma **dependência funcional**. Por exemplo, como 'Silva' leciona tanto 'Estruturas de Dados' e 'Gerenciamento de Dados', podemos concluir que o Professor não determina funcionalmente a Disciplina.

ENSINA

Professor	Disciplina	Texto
Silva	Estruturas de Dados	Bartram
Silva	Gerenciamento de Dados	Martin
Neto	Compiladores	Hoffman
Braga	Estruturas de Dados	Horowitz

Figura 15.7

A chave de Funcion_Horistas é cpf. Além disso, suponha que o atributo salário_hora seja determinado pelo atributo avaliação. Isto é, para determinado valor de avaliação, existe apenas um valor de salário_hora permitido. Essa RI é um exemplo de **dependência funcional (DF)**. Ela leva a uma possível redundância na relação Funcion_Horistas, conforme ilustrado na Figura 19.1.

cpf	*nome*	*vaga*	*avaliação*	*salário_hora*	*horas_trabalhadas*
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

Figura 19.1 Uma instância da relação Funcion_Horistas.

Se o mesmo valor aparece na coluna avaliação de duas tuplas, a RI nos informa que o mesmo valor deve aparecer também na coluna salário_hora. Essa redundância tem as mesmas conseqüências negativas de antes:

- Armazenamento redundante: o valor de avaliação 8 corresponde a salário/hora 10 e essa associação é repetida três vezes.
- Anomalias de atualização: o valor de salário_hora na primeira tupla poderia ser atualizado sem se fazer uma alteração semelhante na segunda tupla.
- Anomalias de inserção: não podemos inserir uma tupla para um funcionário, a menos que saibamos o salário/hora do valor de avaliação desse funcionário.
- Anomalias de exclusão: se excluirmos todas as tuplas com determinado valor de avaliação (por exemplo, excluirmos as tuplas de Smethurst e de Guldu), perderemos a associação entre esse valor de avaliação e seu valor de salário_hora.

Idealmente, queremos esquemas que não permitam redundância, mas no mínimo queremos identificar os esquemas que permitem. Mesmo que optemos por aceitar um esquema com alguns desses inconvenientes, talvez por causa de considerações quanto ao desempenho, queremos tomar uma decisão abalizada. Valores **nulos** não fornecem uma solução geral para os problemas de redundância, mesmo ajudando em alguns casos.

Fechamento de um Conjunto de DFs

O conjunto de todas as **DFs** implicadas por determinado conjunto F de DFs é chamado de **fechamento de F**, denotado por **F+**. As três regras a seguir, chamadas de **Axiomas de Armstrong**, podem ser aplicadas repetidamente para inferir todas as DFs implicadas por um conjunto F de DFs. Usamos X, Y e Z para denotar conjuntos de atributos sobre um esquema de relação R:

- **Reflexividade**: Se X ⊇ Y, então X -> Y.
- **Aumento**: Se X -> Y, então XZ -> YZ para qualquer Z.
- **Transitividade**: Se X -> Y e Y -> Z, então X -> Z.

Teorema 1 *Os Axiomas de Armstrong são **corretos**, no sentido de que eles geram apenas DFs em F+ quando aplicados a um conjunto F de DFs. Eles também são **completos**, no sentido de que a aplicação repetida dessas regras gerará todas as DFs no fechamento F+.* É conveniente usar algumas **regras adicionais** ao raciocinar sobre F+:

- ***União**: Se X -> Y e X -> Z, então X -> YZ.*
- ***Decomposição**: Se X -> YZ, então X -> Y e X -> Z.*

Como outro exemplo, usamos uma versão mais elaborada de Contratos:

Contratos(id-contrato, id-fornecedor, id-projeto, id-depto, id-peça, qtidade, valor)

Denotamos o esquema de Contratos como CSJDPQV. O significado de uma tupla é que o contrato com **id-contrato C** é um acordo de que o **fornecedor S (id-fornecedor)** fornecerá **Q itens da peça P (id-peça)** para o **projeto J (id-projeto)**, associado ao **departamento D (id-depto)**; o **valor V** deste contrato é igual a valor. Sabe-se que as seguintes RIs valem:

1. O id de contrato C é uma chave: C -> CSJDPQV.
2. Um projeto adquire determinada peça usando um único contrato: JP -> C.
3. Um departamento adquire no máximo uma peça de um fornecedor: SD -> P.

Diversas DFs adicionais valem no fechamento do conjunto de DFs dadas:

- De JP -> C, C -> CSJDPQV e da transitividade, inferimos JP -> CSJDPQV.
- De SD -> P e do aumento, inferimos SDJ -> JP.
- De SDJ -> JP, JP -> CSJDPQV e da transitividade, inferimos SDJ -> CSJDPQV. (A propósito, embora possa parecer tentador, não podemos concluir SD -> CSDPQV, cancelando J nos dois lados. A inferência de DF não é como a multiplicação aritmética!)

Podemos inferir várias DFs adicionais que estão no fechamento, usando aumento ou decomposição. Por exemplo, de C -> CSJDPQV, usando decomposição, podemos inferir:

C -> C, C -> S, C -> J, C -> D e assim por diante

Finalmente, temos várias DFs triviais a partir da regra da reflexividade.

???Fechamento de Atributo

Se quisermos apenas verificar se determinada dependência, digamos, X ->Y, está no fechamento de um conjunto F de DFs, podemos fazer isso eficientemente sem calcular F+. Primeiramente, calculamos o **fechamento de atributo** X+ com relação a F, que é o conjunto de atributos A tal que X ->A pode ser inferido usando-se os **Axiomas de Armstrong**. O algoritmo para calcular o fechamento de atributo de um conjunto X de atributos aparece na Figura 19.4.

$fechamento = X;$
repete até que não haja alteração: {
    se houver uma DF $U \rightarrow V$ em F, tal que $U \subseteq fechamento$,
        então configura $fechamento = fechamento \cup V$
}

Figura 19.4 Calculando o fechamento de atributo do conjunto de atributos X.

Teorema 2 *O algoritmo mostrado na Figura 19.4 calcula o fechamento de atributo X+ do conjunto de atributos X, com relação ao conjunto de DFs F.*

Esse algoritmo pode ser modificado para encontrar chaves, começando com o conjunto X que contém um único atributo e parando assim que o fechamento contiver todos os atributos no esquema de relação.

Variando o atributo inicial e a ordem em que o algoritmo considera as DFs, podemos obter todas as chaves candidatas.

Forma Normal

Normalização de Dados em Bancos de Dados (1FN, 2FN, 3FN)

Dado um esquema de relação, precisamos decidir se ele é um bom projeto ou se precisamos decompô-lo em relações menores. Tal decisão deve ser conduzida por um entendimento de quais problemas (se houver) surgem a partir do esquema corrente. Para fornecer tal condução, diversas **formas normais** foram propostas. Se um esquema de relação está em uma dessas formas normais, sabemos que certos tipos de problemas não podem surgir.

As formas normais baseadas em DFs são a primeira forma normal (1FN), segunda forma normal (2FN), terceira forma normal (3FN) e forma normal de Boyce-Codd (FNBC). Essas formas têm requisitos cada vez mais restritivos: toda relação na FNBC também está na 3FN, toda relação na 3FN também está na 2FN e toda relação na 2FN está na 1FN.

Primeira forma normal (1FN)

Uma relação está na **primeira forma normal** se todo campo contém apenas **valores atômicos**; isto é, nenhuma lista nem conjuntos. Esse requisito está implícito em nossa definição de modelo relacional.

- Eliminar grupos repetidos em tabelas individuais.
- Crie uma tabela separada para cada conjunto de dados relacionados.
- Identifique cada conjunto de dados relacionados com uma chave primária.

Não use vários campos em uma única tabela para armazenar dados semelhantes. Por exemplo, para rastrear um item de inventário que pode vir de duas fontes possíveis, um registro de inventário pode conter campos para o Código do Fornecedor 1 e o Código do Fornecedor 2.

O que acontece quando você adiciona um fornecedor terceiro? Adicionar um campo não é a resposta; isso exige modificações de programa e tabela e não acomoda facilmente um número dinâmico de fornecedores. Em vez disso, coloque todas as informações do fornecedor em uma tabela separada chamada Fornecedores e vincule o inventário a fornecedores com uma chave de número de item ou fornecedores para inventariá-los com uma chave de código de fornecedor.

Tabela não normalizada:

Student#	Advisor	Adv-Room	Class1	Class2	Class3
1022	Jones	412	101-07	143-01	159-02
4123	Smith	216	101-07	143-01	179-04

Primeira forma normal: sem grupos repetidos

As tabelas devem ter apenas duas dimensões. Como um aluno tem várias classes, elas devem ser listadas em uma tabela separada. Os campos Class1, Class2 e Class3 nos registros acima são indicações de problemas de design.

As planilhas geralmente usam a terceira dimensão, mas as tabelas não. Outra maneira de ver esse problema é com uma relação um-para-muitos, não coloque o lado único e vários lados na mesma tabela. Em vez disso, crie outra tabela na primeira forma normal eliminando o grupo repetido (Class#), conforme no seguinte exemplo:

Student#	Advisor	Adv-Room	Class#
1022	Jones	412	101-07
1022	Jones	412	143-01
1022	Jones	412	159-02
4123	Smith	216	101-07
4123	Smith	216	143-01
4123	Smith	216	179-04

Segunda forma normal (2FN)

A definição de segunda forma normal é basicamente que as **dependências parciais não são permitidas**.

- Crie tabelas separadas para conjuntos de valores que se aplicam a vários registros.
- Relacione essas tabelas com uma chave estrangeira.

Os registros não devem depender de nada além da chave primária de uma tabela (uma chave composta, se necessário). Por exemplo, considere o endereço de um cliente em um sistema de contabilidade. O endereço é necessário para a tabela Clientes, mas também para as tabelas Pedidos, Envio, Faturas, Contas a Receber e Coleções. Em vez de armazenar o endereço do cliente como uma entrada separada em cada uma dessas tabelas, armazene-o em um só lugar, na tabela Clientes ou em uma tabela Endereços separada.

Segunda forma normal: eliminar dados redundantes

Observe os vários valores Class# para cada valor Student# na tabela acima. Class# não depende funcionalmente de Student# (chave primária), portanto, essa relação não está na segunda forma normal.

As tabelas a seguir demonstram a segunda forma normal:

Alunos:

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216

Registro:

Student#	Class#
1022	101-07
1022	143-01
1022	159-02
4123	101-07
4123	143-01
4123	179-04

Terceira forma normal (3FN)

A ideia é que a terceira forma normal visa **eliminar campos não dependentes da chave**. Assim, uma relação está na 3FN se não houver **dependências parciais e transitivas**.

Seja R um esquema de relação, seja F o conjunto de DFs dadas como válidas sobre R, seja X um subconjunto dos atributos de R e seja A um atributo de R. R está na terceira forma normal se para toda DF X -> A em F, uma das seguintes declarações é verdadeira:

- A ∈ X; isto é, trata-se de uma DF trivial, ou
- X é uma superchave, ou
- A faz parte de alguma chave de R.

Para entender a terceira condição, lembre-se de que uma chave para uma relação é um conjunto mínimo de atributos que determina exclusivamente todos os outros atributos. A deve fazer parte de uma chave (qualquer chave, se houver várias). Não é suficiente que A faça parte de uma superchave, pois a última condição é satisfeita por todo atributo! O ato de **encontrar todas as chaves de um esquema de relação é conhecido como problema NP completo** e assim também é o problema de determinar se um esquema de relação está na 3FN.

Suponha que uma dependência X -> A cause uma violação da 3FN. Existem dois casos:

- X é um subconjunto de alguma chave K. Às vezes, essa dependência é chamada de **dependência parcial**.
- X não é subconjunto de nenhuma chave. Às vezes, essa dependência é chamada de **dependência transitiva**, pois significa que temos um encadeamento de dependências K -> X -> A. O problema é que não podemos associar um valor de X a um valor de K, a não ser que também associamos um valor de A a um valor de X.

As dependências parciais estão ilustradas na Figura 19.7 e as dependências transitivas estão ilustradas na Figura 19.8. Note que, na Figura 19.8, o conjunto X de atributos pode ou não ter alguns atributos em comum com CHAVE; o diagrama deve ser interpretado como indicando apenas que X não é um subconjunto de CHAVE.

Figura 19.7 Dependências parciais.

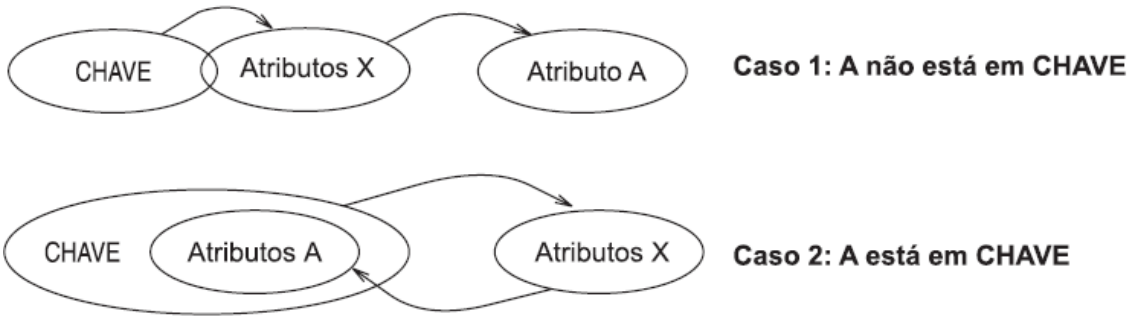

Figura 19.8 Dependências transitivas.

A motivação para a 3FN é bastante técnica. Fazendo uma exceção para certas dependências envolvendo atributos de chave, podemos garantir que todo esquema de relação pode ser decomposto em um conjunto de relações 3FN usando apenas decomposições que tenham certas propriedades desejáveis. Entretanto, ao contrário da FNBC, na 3FN alguma redundância é possível. Os problemas associados às dependências parciais e transitivas persistirão se houver uma dependência não-trivial X -> A e X não for uma superchave, mesmo que a relação esteja na 3FN, pois A faz parte de uma chave.

Os valores em um registro que não fazem parte da chave deste registro não pertencem à tabela. Em geral, sempre que o conteúdo de um grupo de campos pode se aplicar a mais de um único registro na tabela, considere colocar esses campos em uma tabela separada.

Por exemplo, em uma tabela de Recrutamento de Funcionários, o nome da universidade e o endereço de um candidato podem ser incluídos. Mas você precisa de uma lista completa de universidades para envio de mensagens em grupo. Se as informações da universidade forem armazenadas na tabela Candidatos, não haverá como listar universidades sem candidatos atuais. Crie uma tabela de Universidades separada e vincule-a à tabela Candidatos com uma chave de código da universidade.

No último exemplo, Adv-Room (o número do escritório do consultor) é funcionalmente dependente do atributo Advisor. A solução é mover esse atributo da tabela Alunos para a tabela Docentes, conforme mostrado abaixo:

Alunos:

Student#	Advisor
1022	Jones
4123	Smith

Docentes:

Nome	Room	Depto
Jones	412	42
Smith	216	42

EXCEÇÃO: a adesão à terceira forma normal, embora teoricamente desejável, nem sempre é prática. Se você tiver uma tabela Clientes e quiser eliminar todas as possíveis dependências entre campos, deverá criar tabelas separadas para cidades, CEPs, representantes de vendas, classes de clientes e qualquer outro fator que possa ser duplicado em vários registros. Em teoria, vale a pena buscar a normalização. No entanto, muitas tabelas pequenas podem degradar o desempenho ou exceder as capacidades abertas de arquivo e memória.

Pode ser mais viável aplicar a terceira forma normal apenas aos dados que mudam com frequência. Se alguns campos dependentes permanecerem, projete seu aplicativo para exigir que o usuário verifique todos os campos relacionados quando qualquer um deles for alterado.

???Forma Normal de Boyce-Codd (NFBC)

Se usarmos elipses para denotar atributos ou conjuntos de atributos e desenharmos setas para indicar DFs, uma relação na FNBC terá a estrutura ilustrada na Figura 19.5, considerando apenas uma chave por simplicidade. (Se existem várias chaves candidatas, cada chave candidata pode desempenhar o papel de CHAVE na figura, com os outros atributos sendo aqueles que não estão na chave candidata escolhida.)

Figura 19.5 DFs em uma relação na FNBC.

A FNBC garante que nenhuma redundância pode ser detectada usando-se apenas as informações da DF. Assim, ela é a forma normal mais desejável (do ponto de vista da redundância), se levarmos em conta apenas as informações da DF. Esse ponto está ilustrado na Figura 19.6.

X	Y	A
x	y_1	a
x	y_2	?

Figura 19.6 Instância ilustrando a FNBC.

Essa figura mostra (duas tuplas em) uma instância de uma relação com três atributos, X, Y e A. Existem duas tuplas com o mesmo valor na coluna X. Agora, suponha que saibamos que essa instância satisfaz uma DF X -> A. Podemos ver que uma das tuplas tem o valor a na coluna A. O que podemos inferir sobre o valor da coluna A na segunda tupla? Usando a DF, podemos concluir que a segunda tupla também tem o valor a nessa coluna.

Mas essa situação não é um exemplo de redundância? Parece que armazenamos o valor a duas vezes. Uma situação assim pode surgir em uma relação na FNBC? A resposta é: não! Se essa relação está na FNBC, porque A é diferente de X, segue-se que X deve ser uma chave. (Caso contrário, a DF X -> A violaria a FNBC.) Se X é uma chave, então y1 = y2, o que significa que as duas tuplas são idênticas. Como uma relação é definida como um conjunto de tuplas, não podemos ter duas cópias da mesma tupla e a situação mostrada na Figura 19.6 não pode surgir.

Portanto, se uma relação está na FNBC, todo campo de toda tupla registra uma informação que não pode ser inferida (usando-se apenas DFs) a partir dos valores presentes em todos os outros campos na (em todas as tuplas da) instância da relação.

Referencias:

- https://learn.microsoft.com/pt-br/office/troubleshoot/access/database-normalization-description
- Fundamentos de Sistemas de Gerenciamento de Banco de Dados - Elmasri & Navathe (6ª edição)
- Sistemas de Gerenciamento Sistemas de Gerenciamento de Banco de Dados - Ramakrishnan Gehrke (3ª edição)