Universidade Federal da Fronteira Sul - UFFS
Campus Chapecó
Ciência da Computação
Banco de Dados I
Prof.: Denio Duarte

Formais Normais

Antes de discutirmos as formas normais, vamos relembrar o modelo relacional.

Modelo Relacional

Edgar F. Codd publicou em 1970 no Jornal Communications of the ACM o artigo *A relational model of data for large shared data banks* que foi primeiro a descrever o modelo relacional. Nesta época os modelos de dados utilizados eram redes e hierárquicos que possuiam muitas limitações quanto ao armazenamento, consultas e otimizações. O modelo relacional obteve tanto sucesso que até hoje é representado pelos banco de dados relacionais (BD)[1] gerenciados pelos Sistemas Gerenciadores de Banco de Dados (SGBD) que são os mais utilizados no mundo. Basta citar os SGBDs *Oracle*, *SQL Server* e *DB2* para sabermos o quanto esse modelo é utilizado pelas empresas desenvolvedores de aplicações computacionais.

Características do modelo relacional:

- Organização dos dados: domínio, atributo, tupla, relação, chaves.
- Restrições: restrições que garantem a integridade dos dados e de seus relacionamentos.
- Manipulação dos dados: linguagens de consultas como SQL (implementada) e formais (álgebra relacional, cálculo relacional de tuplas).

Domínio: é o conjunto de valores permitidos para um determinado dado (ou para um atributo). Podemos ter o tipo do domínio, por exemplo, numérico, alfanumérico, lógico, etc, e a restrição do domínio que indica a faixa de valores para um determinado tipo. Por exemplo, um atributo pode ser numérico restrito aos inteiros positivos, ou alfanumérico restrito aos valores "M" ou "F". Para um determinado domínio existe um conjunto de operações válidas. Por exemplo, em um domínio numérico pode-se utilizar as operações de adição, multiplicação, divisão e subtração. Os domínios são, na maioria das vezes, definidos na criação das relações (tabelas) através da Linguagem de Definição de Dados (DDL - Data Definition Language).
Atributo: é um item de dado do banco de dados. É a menor característica de uma relação (tabela). Possui nome e domínio. Exemplo: <Endereco, Cadeia de Caracteres>. O valor de um atributo deve ser (*i*) compatível com o domínio definido para o mesmo ou $NULL$ (nulo) se o atributo não for obrigatório e (*ii*) atômico (indivisível). Essa característica é conceitual, pois podemos considerar o endereço como atômico caso não tenhamos a necessidade de separar o número do nome da rua, por exemplo.
Tuplas: é um conjunto formado pelos atributos e seu valores, ou seja, é um conjunto de pares <atributo, valor>. Geralmente, esses pares têm alguma relação entre si para formar uma tupla. Por exemplo, podemos ter uma tupla que caracteriza o cliente em um banco dados, assim, através de levantamento dos requsitos podemos concluir que a relação (ou tabela) *cliente* pode ter os atributos: *codigo*, *nome*, cpf. Assim, um exemplo de tupla nessa aplicação seria <codigo:210, nome: fulano de tal, cpf: 3334445556>

[1] A sigla correta para indicar banco de dados relacionais é BDR, porém, por abuso de notação, utilizaremos apenas BD.

Relação: é um conjunto de tuplas que possuem alguma relação entre si. Assim, a relação *cliente* é formada por um conjunto de tuplas (possivelmente vazio): {<codigo: 210, nome: fulano de tal, cpf:3334445556>, <codigo:300, nome: beltrano, cpf:1114449996>, <codigo:100, nome:grosjean, cpf:2226667771>}. A ordem que as tuplas aparecem nas relações é irrelevante e, teoricamente, não previsível. Os atributos, por sua vez, possuem ordem, assim, se sabemos que os atributos de uma tupla são *codigo*, *nome* e cpf, a relação anterior poderia ser representada como {<210, fulano de tal, 3334445556>, <300, beltrano, 1114449996>, < 100, grosjean, 2226667771>}, sabendo-se *a priori* que o primeiro valor corresponde ao código, o segundo ao nome e o terceiro ao CPF. Formalmente, como uma relação é um conjunto de tuplas, duas tuplas não podem ter os mesmos valores. Porém, uma relação pode ser uma coleção de tuplas que permite tuplas com valores iguais, neste caso, temos o conceito de *tabela* ao invés de *relação*. Uma relação possui *grau* que corresponde ao número de atributos e uma cardinalidade, que corresponde ao número de tuplas. A relação *cliente*, por exemplo, tem grau igual a 3 (atributos *codigo*, *nome* e cpf) e cardinalidade 3 (as três tuplas listadas anteriormente). A relação é vista por duas formas: o esquema e a instância. O esquema da relação cliente é $cliente(codigo, nome, cpf)$ e a instância é {<210, fulano de tal, 3334445556>, <300, beltrano, 1114449996>, < 100, grosjean, 2226667771>}.
Chave: é um conjunto de atributos de uma tupla que a identificam unicamente na relação. Vamos definir algumas noções de chave:

- Super chave (SC): é um conjunto de atributos que identifica unicamente uma tupla em uma relação.
- Chave (Ch): é uma superchave que se retirarmos um dos atributos deixa de ser uma super chave.
- Chave candidata (CC): é uma das chaves da relação que pode ser utilizada para identificar a mesma unicamente.
- Chave primária (CP): é chave candidata escolhida para identificar unicamente uma tupla.

Na relação Cliente, por exemplo, o conjunto de super chaves seria SC={*codigo nome* cpf, *codigo nome*, *codigo*, ...}. Aplicando o conceito de chave no conjunto SC, teríamos como resultado *codigo* e cpf, pois, por exemplo, se pegarmos a super chave *codigo nome* cpf e retirarmos o atributo cpf, resultando em *codigo nome*, este resultado continua sendo uma super chave, se retirarmos *nome*, resulta em *codigo* que continua sendo uma super chave. Como resultou apenas *codigo*, não é mais possível retirar atributo, *codigo* torna-se uma chave candidata. No nosso exemplo, temos duas chaves candidatas: *codigo* e cpf (que foi encontrada utilizando o mesmo raciocínio utilizado para encontrar *codigo*). Dessas duas chaves, escolhemos uma delas para ser primária, por exemplo *codigo*.
O conceito de chave estrangeira difere um pouco do conceito de chave acima pois é aplicado em inter-relações porém, como o nome próprio diz, é uma chave.

- Chave estrangeira: é um conjunto de atributos fk de uma relação R_1 que se relaciona com outro conjunto de atributos pk de uma outra relação R_2 (geralmente pk representa a chave primária de R_2). Essa relação é em nível de domínio e valor: $val(fk) = val(pk)$ ou $val(fk) = null$ e $dominio(fk) = dominio(pk)$. Observe que R_1 e R_2 podem representar a mesma relação.

  Exemplo: dado o esquema da relação $cliente(codigo, nome, cpf)$, sendo *codigo* a chave primária. Dados o esquema da relação $nota\_fiscal(numero, codcli, data)$ e a chave estrangeira *codcli* que faz referência ao atributo *codigo* da relação *cliente*. A relação $nota\_fiscal$ será consistente em relação à chave estrangeira, se todo o valor que aparece em *codcli* na

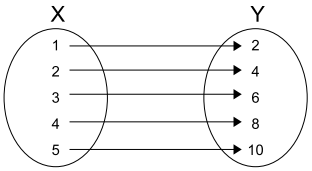
relação $nota\_fiscal$ tem um valor igual em qualquer tupla da relação *cliente* no atributo *codigo* (ou *codcli* é *null* - nulo).

Restrições: são um conjunto de regras que garante a integridade do banco de dados. Podemos classificar a integridade do banco de dados em: integridade dos dados (os valores dos dados dos atributos respeitam os domínios impostos), integridade da entidade (os valores definidos como chave primária devem ser únicos) e integridade referencial (mantem a consistência das chaves estrangeira conforme definido anteriormente).
Manipulação dos dados: a manipulação de dados é realizada através de linguagens de consulta. A linguagem de consulta implementada mais utilizada é a SQL (Structured Query Language) e a mesma é objeto de estudo mais detalhado neste curso.

Dependências Funcionais - DF

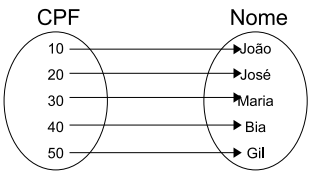
Quando se projeta um banco de dados tem-se um conjunto de esquemas relacionais (relações) e de restrições de integridade que serão utilizados para a implementação do banco. Esse projeto inicial é refinado tendo como ponto de partida as restrições de integridade, número de tuplas das relações, desempenho esperado, entre outros. As restrições aqui tratadas serão as dependências funcionais (DF).

O modelo relacional se baseou no conceito de dependência funcional da teoria de funções da matemática para implementar este tipo de restrição. Considere os conjuntos X e Y na figura abaixo:



Observe que existe uma dependência entre os valores y (elementos) do conjunto Y em relação aos elementos x do conjunto X. Essa dependência pode ser expressa pela função $f(x) = x \times 2$, dizemos que $f(x)$ é uma função de mapeamento pois $y = f(x)$. Assim, todas as vezes que encontrarmos um elemento em X e este tiver um correspondente em Y, o valor do elemento em Y deve ser o dobro de seu correspondente em X. Perceba, então, que os valores em Y dependem funcionalmente dos valores em X. Essa dependência garante que, se $18 \in Y$, então deve existir 9 em X. Caso 9 não exista em X, conclui-se que o conjunto Y não está consistente (utilizando termos do modelo relacional).
Regra 1: um banco de dados relacional é consistente em relação a suas dependências funcionais se todas as dependências definidas pelo projetista são respeitadas.

Observe os conjuntos abaixo e a função de mapeamento entre CPF e $Nome$. Essa função $f(CPF) = nome$ define a dependência funcional entre CPF e $Nome$, ou seja, todas as vezes que o $CPF = 30$, $Nome = Maria$, por exemplo.



O exemplo acima mostra a importância das DF dentro do modelo relacional pois através delas é possível garantir uma série de restrições que mantém a consistência do banco de dados. No caso acima, por exemplo, não poderá ser possivel encontrar um nome diferente de $Maria$ associado ao CPF 10. A notação utilizada para representar essa dependência é $CPF \rightarrow Nome$.

As DF são base para a definição das formas normais (FN) que, por sua vez auxiliam no refinamento de esquemas de BD.

Refinamento de Esquemas

No projeto de um BD é necessário refinar o esquema proposto até se encontrar um esquema que permite armazenar e recuperar os dados de forma eficiente. O refinamento de esquema, na maioria das vezes, é feito através de decomposição do esquema, ou seja, tem-se um relação R (ou tabela) e identifica-se que R pode armazenar, por exemplo, dados redundantes. Porém, a decomposição deve ser cuidadosa pois pode-se eliminar a redundância e ganharmos outros tipos de problemas (como perda de junção que vamos discutir mais adiante).

Vamos, inicialmente, discutir os problemas causados pela redundância.

- Armazenamento redundante: uma mesma informação é armazenada mais de uma vez. Por exemplo, o nome do aluno pode estar armazenado na tabela *Aluno* e na tabela $Matricula$.
- Anomalias de atualização: se temos cópias de dados repetidas, uma delas pode ser atualizada enquanto a outra não. Por exemplo, na tabela *Aluno* o nome *Maria de Souza* pode ser atualizado para *Maria de Souza Brito* (pois a Maria se casou), porém, o mesmo nome na tabela $Matricula$ deve ser atualizado também (esta operação pode ser trabalhosa e facilmente esquecida pelos usuários da aplicação).
- Anomalias de inserção: pode ser impossível armazenar um novo dado ou uma nova tupla por falta de conhecimento de um dos valores a ser inserido. Apresentaremos um exemplo mais tarde para ilustrar essa anomalia.
- Anomalias de exclusão: pode ser excluir um dado ou linha de uma tabela que contenha informações para outras linhas, sendo que essas últimas ficariam "orfãs". Apresentaremos um exemplo mais tarde para ilustrar essa anomalia.

Considere a relação $HorasEmp(codigo, nome, depto, nivel, valhora, horatrab)$, onde *codigo* é o código do empregado, *nome* é o nome, *depto* é o departamento onde está lotado, *nivel* é o nível do empregado, *valhora* é o valor da hora que um empregado de nivel *nivel* recebe e *horatrab* corresponde às horas trabalhadas pelo empregado. A chave primária é *codigo*. Observe que o atributo *valhora* depende funcionalmente do atributo *nivel*, ou seja, se tivermos uma ocorrência $nivel = 1$ com $valhora = 25$, todas as vezes que encontrarmos $nivel = 1$, deveremos encontrar $valhora = 25$. É o mesmo caso DF CPF e $Nome$ apresentada anteriormente. Assim, temos $nivel \rightarrow valhora$.

A tabela abaixo apresenta uma instância da relação $HorasEmp$.

codigo	nome	depto	nivel	valhora	horatrab
10	João	48	8	10	40
20	Maria	22	8	10	30
30	Gil	35	5	7	30
40	Bia	35	5	7	32
50	José	35	8	10	40
60	Carl	22	7	8	20

Baseado na DF $nivel \rightarrow valhora$, sabemos que os valores na coluna (ou do atributo) $valhora$ deve ser o mesmo para as mesmas ocorrências de $nivel$. Por exemplo, os empregados de códigos 10, 20 e 50 possuem o mesmo nível (8) e consequentemente o mesmo valor da hora (10), pois a DF $nivel \rightarrow valhora$ precisa ser respeitada.

Agora vamos apresentar exemplos utilizando a tabela acima dos problemas causados pela redundância:

- Armazenamento redundante: a associação entre $nivel$ e $valhora$ faz com que esses valores se repitam dentro da tabela, por exemplo, o valor $8 \rightarrow 10$ aparece 3 vezes.
- Anomalias de atualização: devido a DF citada, poderia-se atualizar o valor da hora do empregado 10 (João) de 10 para 12. Neste caso, os empregados 20 e 50 estariam com o valor da hora desatualizado e consequentemente a DF $nivel \rightarrow valhora$ não seria mais respeitada.
- Anomalias de exclusão: se excluirmos o empregado $Carl$ (codigo 60), perdemos a associação do nível 7 com o valor da hora 8.
- Anomalias de inserção: aproveitando o exemplo da exclusão, para inserirmos uma nova tupla com um nível ainda não existente, teríamos de "descobrir" qual o valor da hora do novo nível inserido. Como a exclusão fez perder a associação do nível 7 com o valor da hora 8, para a inserção de um novo empregado de nível 7, teríamos que identificar qual é o valor da hora para este nível.

Portanto, é desejável que esquemas não permitam redundância. As dependências funcionais e formas normais permitem que redundâncias possam ser identificadas no projeto e acertadas antes da implementação do banco.

Uma forma de eliminar a redundância é decompor o esquema de uma relação. A decomposição consiste em criar uma ou mais relações a partir da relação que possui a redundância.

Uma decomposição de um esquema relacional R consiste, então, na transformação de R em dois novos (ou mais) esquemas de relação. Dadas a função $attr(Rel)$ que retorna o conjunto de atributos de uma relação Rel e as relações R_1 e R_2 construídos a partir de R para eliminar a redundância de R, essa eliminação deve respeitar as seguintes regras:

1. R_1 e R_2 devem possuir um subconjunto de atributos de R, ou seja, $attr(R_1) \subseteq attr(R)$ e $attr(R_2) \subseteq attr(R)$; e
2. os atributos de R_1 e R_2 devem incluir todos os atributos de R, ou seja, $attr(R) \subseteq attr(R_1) \cup attr(R_2)$.

A relação $HorasEmp$ poderia ser transformada em duas novas relações:

$HorasEmp(codigo, nome, depto, nivel, horatrab)$ e $NivelEmp(nivel, valhora)$

e as instâncias dessas duas relações seriam:

HorasEmp

codigo	nome	depto	nivel	horatrab
10	João	48	8	40
20	Maria	22	8	30
30	Gil	35	5	30
40	Bia	35	5	32
50	José	35	8	40
60	Carl	22	7	20

NivelEmp

nivel	valhora
8	10
5	7
7	8

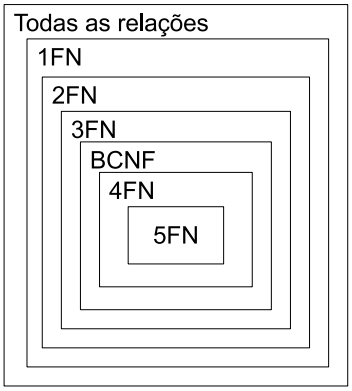
Perceba que este novo esquema não possui redundância, ou seja, anteriormente a relação $8 \rightarrow 10$ aparecia 3 vezez, agora apenas 1 na relação $NivelEmp$. Se alterarmos o valor da hora

do nivel 8, automaticamente, todos os empregados de nível 8 serão associados como o novo valor da hora. Se excluirmos o empregado 60, não perderemos o valor da hora para o nível 7. E, finalmente, só poderemos inserir linhas em $HorasEmp$ com níveis existentes em $NivelEmp$, eliminando a anomalia de inserção.

A decomposição deve ser feita cuidadosamente, pois a mesma pode causar problemas:

- A junção das relações criadas a partir da relação com redundância pode não representar a relação original. Assim, a decomposição deve manter a propriedade *junção-sem-perda*.
- Outro problema é a perda de dependências, ou seja, as dependências que eram garantidas na relação original podem não ser mais garantidas. Esta propriedade é chamada *preservação-de-dependência*.

Temos que, então, verificar se a decomposição é mesmo necessária. Para fazer-se essa verificação foram propostas as *formas normais* (FN). As formas normais foram inicialmente proposta por Codd (sim o mesmo que propôs o modelo relacional). A figura abaixo apresenta a hierarquia das formas nomais. O retângulo mais externo indica todas as relações existentes no banco de dados. Os retângulos mais internos indicam que cada forma normal vai restringindo o número de relação que obedecem as mesmas, ou seja, a 1FN (primeira forma normal) é menos restrita que a 2FN (segunda forma normal). Pela figura percebe-se também que uma relação que está na 3FN (terceira forma normal) obedece as restrições por todas as formas normais "mais externas", ou seja, 1FN e 2FN.



Neste texto estudaremos apenas as formas normais: 1FN, 2FN e 3FN. As formas normais BCNF, 4FN e 5FN ficam como estudo extra-classe se o aluno se interessar em conhecê-las.

Primeira Forma Normal (1FN)

Uma relação está na primeira forma normal se todos os seus atributos são monovalorados e atômicos. Para resolver o problema de atributos multivalorados, deve-se criar um novo atributo que individualize o dado multivalorado. Imagine a relação $Nota$ que representa as notas que os alunos tiveram nas diferentes disciplinas cursadas. O esquema dessa relação pode ser $Nota(matricula, disciplina, notas)$, sendo que $notas$ representa as várias notas que o aluno recebeu na dada disciplina. Percebe-se que o atributo $notas$ é multi-valorado, ou seja, pode possuir mais de um valor. Neste caso, a relação $Nota$ não está na 1FN. Para que a mesma respeite a 1FN, deve-se criar um atributo individual para receber cada nota. A nova relação $Nota(matricula, disciplina, prova, nota)$ resolve esse problema. Agora para cada nota de um

disciplina deve colocar de qual prova é a referida nota. As tabelas abaixo apresentam instâncias das duas relações:

matricula	disciplina	notas
851	BAN	5.5 7.8 6.0
908	SQL	5.0 6.8
104	BAN I	7.0

$\Rightarrow$

matricula	disciplina	prova	notas
851	BAN	1	5.5
851	BAN	2	7.8
851	BAN	3	6.0
908	SQL	1	5.0
908	SQL	2	6.8
104	BAN I	1	7.0

Perceba que a tabela na 1FN ocupa mais espaço em disco mas está mais correta para se fazer consultas e atualizações. As vezes, ao invés de criar uma linha para cada nota, pode optar-se por criar atributos na mesma linha. No nosso exemplo anterior, poderíamos excluir o atributo *notas* e acrescentar os atributos *nota*1, *nota*2 e *nota*3. Essa solução é mais tentadora pois resolve o problema da multivaloração mas existem casos que não é possível prever quais valores possíveis serão necessários, tornando essa solução inviável.

Um atributo atômico é um atributo que não pode ser desmembrado em parte menores ou que para a aplicação não seja necessário tal desdobramento. Por exemplo, imagine a relação $Aluno(matricila, nomecompleto)$. O atributo *nomecompleto* poderia ser desmembrado em *nome* e *sobrenome* caso seja necessário para aplicação fazer consultas por sobrenome ou impressão de relatório no formato europeu ou americano *Sr. Sobrenome nome*, por exemplo. Este raciocínio vale para vários outros casos, como por exemplo, endereço que poderia ser desmembrado em *logradouro*, *numero* e *complemento*. Geralmente, para atributos não atômicos não se cria uma nova relação ou novas linhas na relação, simplesmente substitui-se o atributo não-atômico pelos atributos que compõem o atributo original. No exemplo de endereço, substitui-se *endereco* por *logradouro*, *numero* e *compl*.

Segunda Forma Normal (2FN)

A segunda forma normal tem como base o conceito de dependência funcional. Uma relação R está na 2FN se R está na 1FN e todos os atributos não chaves dependem de toda a chave e não de parte dela. Esta definição implica que se a chave primária de uma relação R é composta por apenas um atributo, então R está, por definição, na 2FN.

Para exemplificar o emprego da 2FN considere a relação (Tabela) *Locacao*(*codcli*, *codimo*, *descimo*, *vlaluguel*, *diavecto*) abaixo:

codcli	codimo	descimo	vlaluguel	diavecto
10	22	Ap. 4 quartos	1.200,00	10
12	45	Kitchinete	430,00	7
14	33	Casa alvenaria	800,00	12
16	05	Galpão	550,00	02

Essa relação armazena dados de locação de imóveis onde a chave primária (PK - Primary Key) é *codlci codimo* representando o código do cliente e o código do imóvel, respectivamente. Os outros atributos representam a descrição do imóvel (*descimo*), o valor do aluguel (*vlaluguel*) e o dia de vencimento do aluguel (*diavecto*).

Munidos da PK temos as seguintes DF *codlci codimo* $\rightarrow$ *descimo*, *codlci codimo* $\rightarrow$ *vlalugel*, *codlci codimo* $\rightarrow$ *diavecto*, ou simplesmente *codlci codimo* $\rightarrow$ *descimo vlalugel diavecto*. Podemos perceber que todos os atributos são determinados funcionalmente pela PK, ou seja, ao encontrarmos um valor para *codlci codimo* deveremos encontrar, por exemplo, o mesmo valor para *vlaluguel*. Porém, se observarmos bem a definição de *Locacao* podemos perceber que o atributo *descimo* além de depender da PK, ele depende apenas de *codimo*, ou seja, a DF *codimo* $\rightarrow$ *descimo* também é garantida no banco de dados pois, todas as vezes que encon-

trarmos um código de imóvel, a descrição deve ser a mesma. Assim, a relação *Locacao* não se encontra na 2FN (mas se encontra na 1FN). Para colocar uma relação (tabela) na 2FN basta garantir que todos os atributos não chave dependam completamente da PK, caso isso não ocorra é necessário decompor a tabela para eliminar essa dependência parcial da chave. No nosso exemplo, basta construirmos uma tabela (relação) para eliminar o problema. O resultado abaixo apresenta uma solução para o nosso caso.

Locacao

codcli	codimo	vlaluguel	diavecto
10	22	1.200,00	10
12	45	430,00	7
14	33	800,00	12
16	05	550,00	02

Imovel

codimo	descimo
22	Ap. 4 quartos
45	Kitchinete
33	Casa alvenaria
05	Galpão

Terceira Forma Normal (3FN)

Uma relação R está na 3FN se R está na 2FN e obedece a seguinte regra: nenhum atributo pode depender transitivamente da PK. A dependência transitiva ocorre no seguinte caso: se $A \rightarrow B$ e $B \rightarrow C$, então $A \rightarrow C$.

Se considerarmos a PK da primeira tabela $HorasEmp$ e seus atributos, temos a seguinte dependência da PK: $codigo \rightarrow nome\ depto\ nivel\ valhora\ horatrab$, porém sabemos que a DF $nivel \rightarrow valhora$ também é garantida em $HorasEmp$, assim a DF $codigo \rightarrow valhora$, foi obtida pois a DF $codigo \rightarrow nivel$ é também garantida. Ou seja, as DF $codigo \rightarrow nivel$ e $nivel \rightarrow valhora$ são garantidas, portanto, por transitividade, $codigo \rightarrow valhora$ também é mantida. Como a 3FN preconiza que não pode haver dependência da PK através da transitividade, essa característica deve ser eliminada.

A solução para a garantir a 3FN é parecida com a solução para a 2FN: decompor a relação. A decomposição de $HorasEmp$ em $HorasEmp$ e $NivelEmp$ é a forma de fazer com que a relação original $HorasEmp$ respeite a 3FN (perceba que as duas relações possuem PK com apenas um atributo e estão, então, na 2FN).

Forma Normal de Boyce-Codd (BCNF)

O objetivo de decompor uma relação em outras é evitar as anomolias de projeto. A BCNF é a forma normal que garante que as anomalias de atualização não ocorrem nas relações decompostas. Uma relação R está na BCNF se R está na 3FN e obedece a regra:

- Todas as vezes que existir um DF não trivial $A_1A_2 \ldots A_n \rightarrow B_1B_2 \ldots B_m$, os atributos $A_1A_2 \ldots A_n$ formam um super chave em R.

Ou seja, o lado esquerdo de qualquer DF de uma relação deve ser uma super chave. Pode ser entendido: o lado esquerdo de qualquer DF deve conter uma chave. Lembrando: uma super chave é um conjunto de atributos (talvez apenas 1) que identifica unicamente uma tupla de uma relação. A chave é uma super chave que, ao se retirar um atributo, deixa se ser super chave.
Exemplo: dada a relação $ItemNFiscal(\underline{numnota}, \underline{codprod}, codcli, codven, qtd, prc)$, as super chaves são:
$numnota, codprod, codcli, codven, qtd$, podemos tirar o atributo prc que ainda temos a super chave $numnota, codprod, codcli, codven, qtd$, retirando qtd temos $numnota, codprod, codcli, codven$ que ainda é uma super chave. Retirando $codven$ temos $numnota, codprod, codcli$ que ainda é uma super chave, o mesmo vale para a retirada do atributo $codcli$. Quando termos $numnota, codprod$ não é mais possível retirar atributos da super chave, assim, temos a chave (ou super chave mínima). Caso, utilizarmos a chave para identificar unicamente a relação, esta chave é chamada

de chave primária. Uma relação pode ter mais de uma chave. As chaves não escolhidas como chave primária são chamadas de chaves candidatas.

Voltando a BCNF, todas as dependências funcionais de uma relação devem ter, no seu lado esquerdo, atributos pertencentes a uma super chave.

A relação $Movie1$ não respeita a BCNF. Inicialmente, devemos identificar os atributos que são chaves. Os atributos *title year starName* formam a chave de $Movie1$ e, assim, qualquer super conjunto dessa chave será uma super chave. Se considerarmos a DF apresentada anteriormente *title year* $\rightarrow$ *length genre studioName* que é respeitada em $Movie1$. O lado esquerdo dessa DF (*i.e.*, *title year*) não é uma super chave, assim, essa DF viola a BCNF e, portanto, $Movie1$ não está na BCNF.

Dada a relação $Movie2(title, year, length, genre, studioName)$ com a instância

title	year	length	genre	studioName
Star Wars	1977	124	SciFi	Fox
Star Wars	1977	124	SciFi	Fox
Star Wars	1977	124	SciFi	Fox
Gone With the Wind	1939	231	drama	MGM
Wayne's World	1992	95	comedy	Paramount
Wayne's World	1992	95	comedy	Paramount

podemos verificar que $Movie2$ respeita a BCNF pois a única chave é *title year*, ou seja, *title year* $\rightarrow$ *length genre studioName* é respeitada por $Movie2$ e nem *year* nem *title* determinam funcionalmente qualquer outro atributo. Qualquer outra super chave será um super conjunto de *title year*.