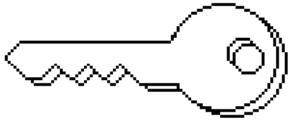
Dependências Funcionais, Normalização

Feliz Ribeiro Gouveia
fribeiro@ufp.edu.pt

Restrições de Entidade num SGBDR: Chave primária

- Uma chave primária é um subconjunto dos atributos de uma relação R
- Uma chave primária permite identificar linhas de forma única
- A definição de chaves primárias é crucial na construção do modelo de uma BDR



Chaves

- Superchave
  - Não há duas linhas com o mesmo valor da superchave
  - Pode ser reduzida e continua única
- Chave candidata
  - Não há duas linhas com o mesmo valor da chave
  - Não pode ser reduzida (é uma chave mínima)
- Chave primária
  - É uma das chaves candidatas; as outras designam-se “alternativas”

Chaves

- Chave estrangeira
  - Conjunto de atributos de uma relação que são chaves primárias noutras relações (ie, referenciam a linha pela chave primária)
  - Nenhuma dessas referências deve ser inválida, isto é, o valor da chave estrangeira deve existir como valor da chave primária

problema da integridade referencial

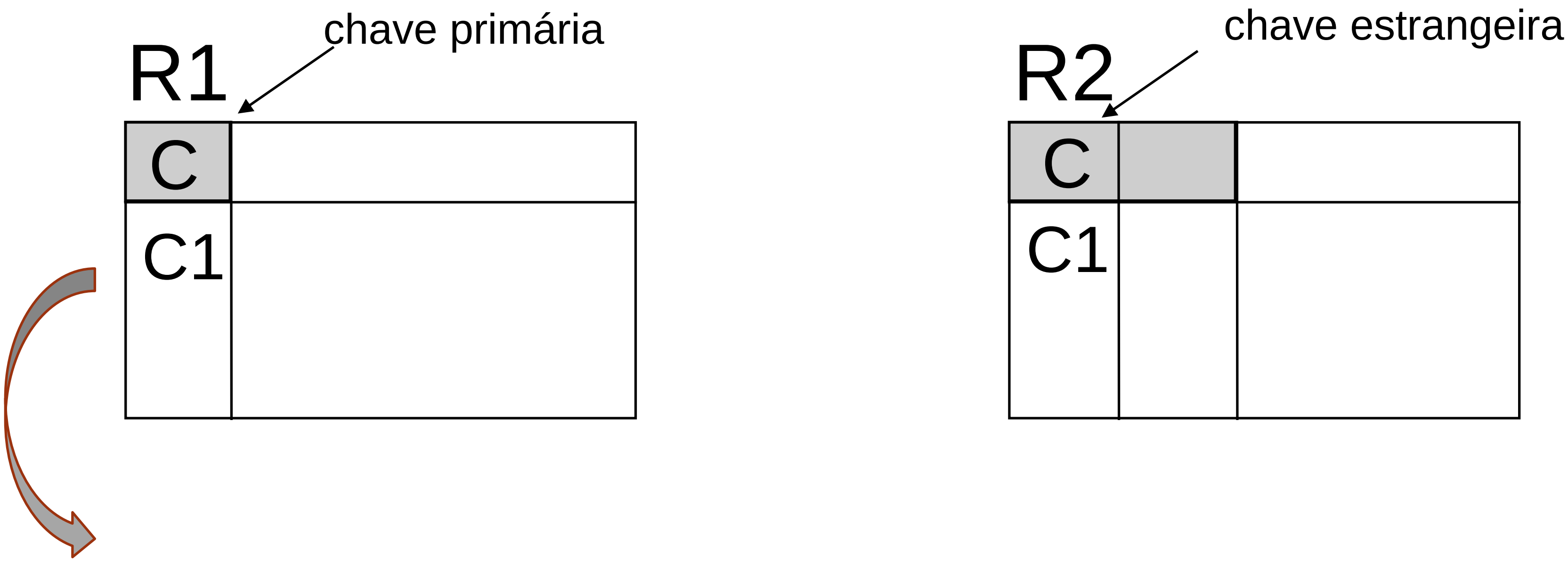
Chaves (cont)

- Atributos primos: atributos que fazem parte de uma chave candidata
- Atributos não-primos: atributos que não fazem parte de uma chave candidata

Atualização de chaves

- E quando se suprime um valor de uma chave primária referenciado numa chave estrangeira?
  - caso restrito: só se pode apagar esse valor se ele não aparecer na chave estrangeira
  - caso em cascata: apagam-se todas as referências como chave estrangeira noutras tabelas

Exemplo



C1 na tabela R1 só se pode eliminar se:
a) não existir em R2 (caso restrito)
b) eliminando em R2 (efeito cascata)

Diagramas Referenciais (DR)

- Permitem representar graficamente as chaves primárias e estrangeiras
- Ex: a chave C é primária na relação P e estrangeira na relação S:

$$S \xrightarrow{C} P$$

Propriedades do DR

- Pode haver referências cíclicas
  - As chaves estrangeira e primária podem estar na mesma tabela: grupo(aluno_id, colega_id)
- Um DR representa um relacionamento, mas estes não são exclusivamente representados por pares de DR

Dependências de inclusão (DI)

- O que vimos representam dependências de inclusão
- Não se limitam a chaves estrangeiras, por exemplo:
  - AVALIADO(aluno_id) ⊆ INSCRITO(aluno_id)
- Ou herança; um ALUNO é uma PESSOA:
  - ALUNO(ID) ⊆ PESSOA(ID)

Dependências Funcionais (DF)

- Por exemplo na tabela ALUNO temos uma dependência entre NÚMERO e NOME
  - Para um dado NÚMERO de aluno o NOME é sempre o mesmo
  - Repare que para o mesmo NOME podem existir diferentes valores de NÚMERO
- Da mesma forma, para cada NÚMERO temos sempre o mesmo valor de MORADA
- Pode identificar mais dependências?

Dependências Funcionais (DF)

- Representam restrições de integridade numa relação
  - Restringem os valores que determinados atributos podem assumir
- Podem ser temporárias (ou coincidências) ou permanentes (só estas últimas nos interessam)
  - Se uma empresa só tiver clientes do Porto, pode pensar que sabendo o produto se sabe a cidade, mas na realidade é coincidência
- A identificação das dependências pode ser difícil, principalmente em domínios complexos

DF: convenção

- As letras A, B, C,…denotam atributos
- As letras X, Y, Z,…denotam conjuntos de atributos
- ABC denota o conjunto {A, B, C}
- R denota todos os atributos da relação R

DF: definição

- Sejam X e Y subconjuntos arbitrários de atributos de uma mesma relação R. Diz-se que Y é **funcionalmente dependente** em X,

  $X \rightarrow Y$
- se e só se cada valor de X tiver associado precisamente um só valor de Y. Diz-se que X **determina** Y, ou que Y **depende** de X.
  - X é o determinante, Y é o dependente
- Dependência **funcional** porque é equivalente a uma função *f*(X) = Y

DF: definição

- Uma DF $X \rightarrow Y$ é trivial se $Y \subset X$ (está contido)
- Uma DF $X \rightarrow Y$ é não-trivial se $Y \not\subset X$
- Uma DF $X \rightarrow Y$ é completamente não-trivial se $Y \cap X = \emptyset$

DF: caso particular

- Se X é uma superchave da relação R, então X determina o conjunto de todos os atributos

$$X \rightarrow \{A_1, A_2, \ldots, A_n\}$$

- Ou seja, se uma DF contém todos os atributos de R, então o seu determinante é uma superchave de R
- Se X determina uma superchave, então X também é uma superchave
- Se $X \rightarrow Y$, e se X não é uma chave candidata, então há redundância na relação (com possíveis problemas de atualização)

DF: redundância

aluno_id	disciplina_id	nome_disciplina
1000	201	Base de dados
1005	300	Programação
1020	400	Algoritmos
1100	201	Base de dados

- Relação **inscrito**
- Chave: {aluno_id, disciplina_id}
- A dependência {disciplina_id} $\rightarrow$ {nome_disciplina} implica redundância na relação, repetindo-se nome_disciplina

Identificação das DF

- Se as DF são importantes para detetar redundâncias, como identificar todas as DF possíveis a partir de um conjunto inicial F de dependências?
  - Ou seja, como determinar todas as DF implicadas por um conjunto F?
- Diz-se que F **implica** $X \rightarrow Y$, e escreve-se $F \vDash X \rightarrow Y$ se $X \rightarrow Y$ for válida em qualquer relação que respeite F
- Uma primeira resposta foi dada por Armstrong, que mostrou que se podiam gerar novas DF

Axiomas de Armstrong

- Sejam X, Y e Z subconjuntos arbitrários de atributos de R. Prova-se o seguinte:
  - Reflexividade: se Y é um subconjunto de X, isto é $X \supseteq Y$, então $X \rightarrow Y$
    - Neste caso a DF é dita trivial. Uma DF trivial é sempre verdadeira em qualquer relação
  - Aumento: se $X \rightarrow Y$ e $Z \supseteq W$, então $XZ \rightarrow YW$
    - Por exemplo, $X \rightarrow Y$ então $XZ \rightarrow YZ$
  - Transitividade: se $X \rightarrow Y$ e $Y \rightarrow Z$ então $X \rightarrow Z$

Validade das regras

- Se uma DF pode ser derivada por estas regras, a partir de um conjunto F de DF, então ela é válida em qualquer relação que respeitar F
  - Não são geradas DF **falsas**
- As regras geram **todas** as DF possíveis
- Regra da união de DF: se $X \rightarrow Y$ e $X \rightarrow Z$
  1. $X \rightarrow XY$ (aumento de X, trivial)
  2. $YX \rightarrow YZ$ (aumento de Y)
  3. $X \rightarrow YZ$ (transitividade de 1 e 2)

Mais regras

- Exemplo anterior define a regra de união de DF:
  - se $X \rightarrow Y$ e $X \rightarrow Z$ então $X \rightarrow YZ$
- Decomposição (ou projeção):
  - Se $X \rightarrow Z$ então $X \rightarrow A$, $\forall A \in Z$
- Pseudo-transitividade:
  - Se $X \rightarrow Y$, e $WY \rightarrow Z$ então $WX \rightarrow Z$

Fecho de DF

- Seja F o conjunto de DF da relação R
- F^+, o fecho de F, é o conjunto de todas as DF que são implicadas por F:
  - $F^+ = \{X \rightarrow Y \mid F \vDash X \rightarrow Y\}$
- F^+ é único
- É possível testar se uma DF *f* é implicada por F gerando F^+ e verificando se $f \in F^+$
- Gerar F^+ tem um custo importante, dependendo do número de colunas
- Uma forma simples é mostrada a seguir

Cálculo de F^+

$F^+ \leftarrow F$

Repetir

Para cada DF $f \in F^+$

aplicar aumento e reflexividade a *f* e juntar as DF resultantes a F^+

Para cada par f_1 e $f_2 \in F^+$

se f_1 e f_2 puderem ser combinadas com transitividade, juntar a DF resultante a F^+

Até F^+ não mudar

- Vamos ver um método alternativo, com um custo inferior, a seguir

Fecho de um atributo

- X^+ é por definição o conjunto de todos os atributos Y tal que $X \to Y \in F^+$

  $X^+ \leftarrow X$

  Repetir

  Para cada DF $Y \to Z \in F$

  Se $Y \subseteq X^+$ e $Z \not\subseteq X^+$

  Juntar Z a X^+, descartar $Y \to Z$

  Até X^+ não mudar ou X^+ ter todos os atributos

- Escreve-se X^+_F se houver necessidade de especificar com qual conjunto de DF o fecho é calculado

Propriedades do fecho

- $X \subseteq X^+$
- Se $X \subseteq X'$, então $X^+ \subseteq (X')^+$
- $(X^+)^+ = X^+$

- Gerando o fecho de todos os atributos obtém-se, por definição de fecho X^+, F^+
- Calcular X^+ tem, com as devidas otimizações, um custo linear em $|F|$

Utilizações de X^+

- Se X é uma **superchave**, $X^+ = R$ por definição de chave
  - Útil para identificar chaves (veremos mais tarde)
- Para saber se $X \rightarrow Y \in F^+$, calcula-se X^+ e testa-se se $Y \in X^+$
- Pode-se calcular F^+ gerando todas as DF implicadas pelo fecho de cada uma das 2^n -1 combinações dos atributos de R
  - Para 10 atributos temos 1023 combinações

Calcular as chaves

- Como vimos, o fecho de um atributo pode ser utilizado para identificar chaves
- X é superchave se $X \rightarrow R \in F^+$, ou seja $X^+ = R$
- X é chave candidata se não existe $X' \subset X$ tal que $X' \rightarrow R \in F^+$
- Yu e Johnson demonstraram que se $|F| = k$, o número máximo de chaves é $k!$
- Devem-se testar as 2^n-1 combinações de atributos de R (algoritmo força bruta)
- Deve-se usar a cobertura canónica G de F

Algoritmo força bruta

$C \leftarrow \emptyset$, no fim contém as chaves

Para todas as permutações $A_1 \ldots A_{2^n-1}$ de R ordenadas por tamanho crescente

se $A_i^+ = R$ então

$C \leftarrow C \cup A_i$

remover do teste qualquer $A_j : A_i \subset A_j$ porque nunca pode ser chave

- O algoritmo testa exaustivamente todos os 2^n-1 conjuntos de atributos, o que é desnecessário

Saiedian e Spencer 1996

- $\mathcal{L}$ é o conjunto de atributos que só aparecem nos determinantes de algumas DF, ou em nenhuma
- $\mathcal{R}$ é o conjunto de atributos que só aparecem nos dependentes de algumas DF
- $\mathcal{B}$ é o conjunto de atributos que aparecem em ambos de lados de algumas DF

- Só devem ser testados os atributos de $\mathcal{L}$, e se necessário aumentados dos de $\mathcal{B}$
- Nunca é necessário testar os atributos de $\mathcal{R}$

Saiedian e Spencer 1996

- $\forall$ A $\in \mathcal{L}$, A é primo, e participa em **todas** as chaves candidatas
- Se os atributos de $\mathcal{L}$ formarem uma chave, isto é se $\mathcal{L}^+$ = R, pode-se parar
- Caso contrário, junta-se cada atributo de $\mathcal{B}$, um a um, a $\mathcal{L}$ e calcula-se o fecho

- Ao contrário de Elmasri e Navathe este algoritmo encontra todas as chaves

Exemplo

- $F = \{AD \rightarrow B, AB \rightarrow E, C \rightarrow D, B \rightarrow C, AC \rightarrow F\}$

$\mathcal{L} = \{A\}$ $\mathcal{B} = \{BCD\}$ $\mathcal{R} = \{EF\}$

- $\mathcal{L}^+ = A$, não é chave

Juntam-se os atributos de $\mathcal{B}$ a $\mathcal{L}$:

- $\{AB\}^+ = ABECDF$, é chave
- $\{AC\}^+ = ACDBEF$, é chave
- $\{AD\}^+ = ADBCEF$, é chave

Normalização

- Destina-se a clarificar o modelo da base de dados e a eliminar possíveis problemas de inserção, remoção e atualização
- Objectivo principal é eliminar redundâncias mantendo as restrições de integridade iniciais
- Resulta quase sempre na decomposição de relações em novas relações
- Tem que se garantir que a decomposição é adequada

Normalização: abordagens

- Em teoria, a normalização pode ser feita apenas a partir de um conjunto de atributos e de dependências funcionais (**síntese**)
  - Dispensa o diagrama E-R, as relações são definidas a partir das dependências e do conjunto de atributos
- Na prática serve para refinar a tradução de um modelo E-R em Relacional (**análise**)
  - Desenha-se o E-R
  - Faz-se o mapeamento em Relacional
  - Refina-se usando DF

Decomposição de relações

- Diz-se que uma decomposição:
  - **É sem perdas** se os tuplos originais puderem ser reconstruídos a partir das relações resultantes da decomposição, e isto sem introduzir tuplos adicionais
  - **Preserva as DF** se as relações resultantes mantiverem as mesmas restrições, sob forma de DF, que a relação original

Teorema de Heath (1971)

- Uma DF implica a existência de uma decomposição sem perdas:
  - Se $X \rightarrow Y$ e $Z = R - XY$, então R admite a decomposição sem perdas $R_1 = XY$ e $R_2 = XZ$
- Neste caso a relação original pode ser sempre reconstruída através de junções
  - $R = \pi_{R1}(R) \bowtie \pi_{R2}(R)$
- Note que "sem perdas" não significa obter menos tuplos na reconstrução; significa mais tuplos, isto é, menos informação

Preservação de dependências

- Uma decomposição de R em R_1 e R_2 preserva as dependências se e só se:
  - F é equivalente a $F_1 \cup F_2$: $F^+ = (F_1 \cup F_2)^+$
- F_i é o conjunto de DF de F^+ que só contém atributos de R_i
- Como as dependências são restrições do problema, devem ser respeitadas independentemente das relações que forem definidas
- Se não forem preservadas, a sua verificação implica junções entre relações

DF e normalização

- 1FN: os valores dos atributos são atómicos, não há grupos de repetição. Deriva da própria definição do Modelo Relacional
- 2FN: está em 1FN e todos os atributos não-primos dependem da totalidade de chaves candidatas
- 3FN: está em 2FN, e todos os atributos não-primos dependem, de forma não transitiva, de chaves candidatas (não há dependências de atributos que não são chave)

DF: Exemplo

- Uma loja regista as compras dos clientes, localizados numa dada cidade, o que implica o custo dos portes; cada compra tem um valor

chave primária

CLIENTE {C_ID, VALOR, CIDADE, <u>E_ID</u>, PORTES}

id. do cliente

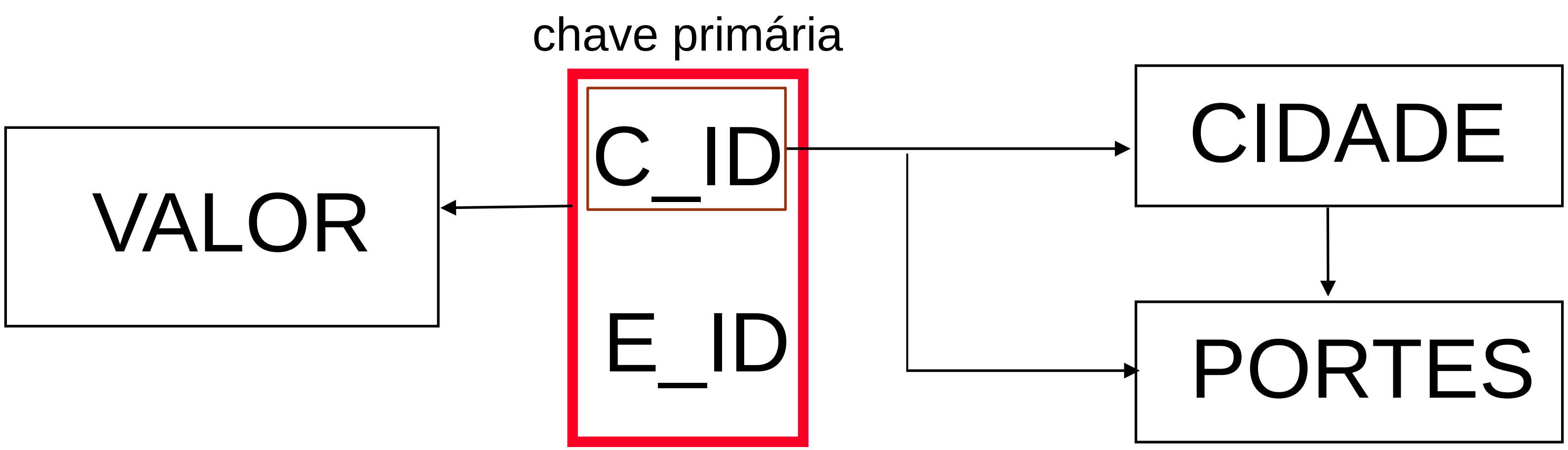
valor desta compra

cidade do cliente

encomenda

portes

DF do exemplo



- Cidade depende de C_ID, que não é a chave primária: {C_ID, E_ID}
- Portes depende de Cidade e do cliente, e não da chave primária
- Valor depende da chave primária

Problemas do exemplo

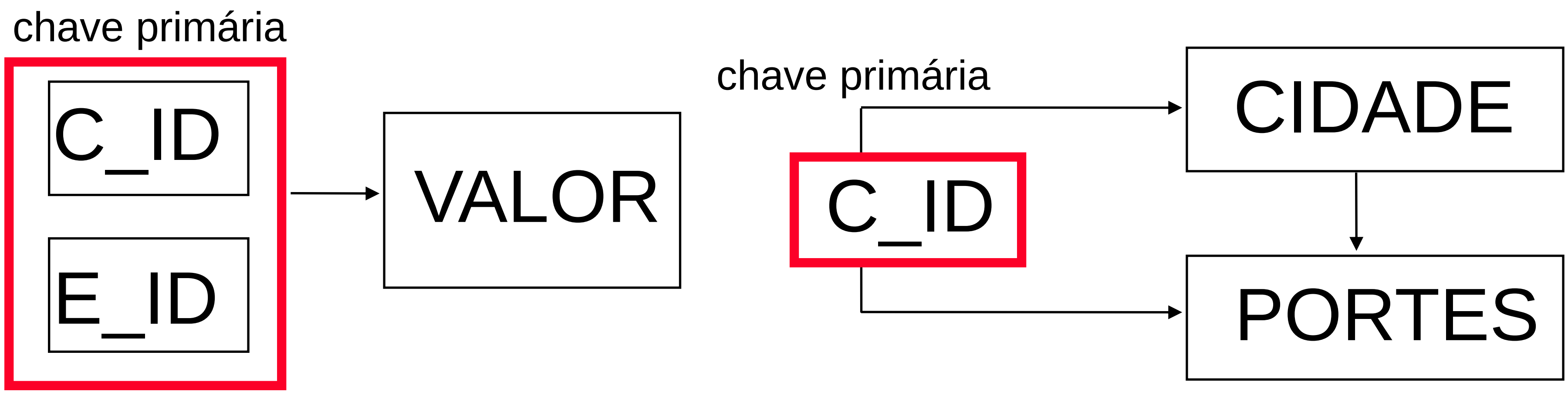
- Inserção: não se pode inserir um cliente sem que se faça uma compra
- Remoção: ao remover um cliente, apaga-se informação útil (a cidade onde está)
- Atualização: se um cliente aparece várias vezes (faz várias compras), Cidade aparece várias vezes

Alteração do exemplo

- Colocar os atributos a dependerem da **totalidade** da chave primária, para o que se decompõe a relação em duas:

CLIENTE {C_ID, CIDADE, PORTES}
ENCOMENDA {E_ID, C_ID, VALOR}

DF para as novas relações



Estão em 2FN: estão em 1FN, e os atributos não-primos dependem da totalidade da chave primária
Como a chave C_ID não é composta, a relação da direita está automaticamente em 2FN

Problemas no novo exemplo

- Inserção: não se pode inserir uma cidade e os portes até haver um cliente nessa cidade
- Remoção: quando se remove um cliente, perde-se a cidade e os portes
- Atualização: quando se inserem vários clientes na mesma cidade, os portes repetem-se

Nova alteração do exemplo

- Decompõe-se a relação cujos atributos não dependem exclusivamente da chave primária:

CLIENTE {C_ID, CIDADE}	C_ID $\rightarrow$ CIDADE
CIDADE {CIDADE, PORTES}	CIDADE $\rightarrow$ PORTES
ENCOMENDA {C_ID, E_ID, VALOR}	

Todas as relações estão em 3FN

O que vimos até agora

- Cada campo de um tuplo deve depender da Chave (1FN), de Toda a Chave (2FN) e Nada Mais do que a Chave (3FN)
- A decomposição das relações é sem perdas, isto é, a relação original pode ser reconstruída
- Vamos ver como decompor relações, usando as dependências funcionais

Definição de 3FN

- Para cada DF: $X \rightarrow Y \in F^+$
  - $Y \subseteq X$ (a DF é trivial) ou
  - X é uma superchave, ou
  - Todos os atributos em X — Y são atributos primos (isto é, fazem parte de uma chave candidata, podem ser chaves diferentes)

- Se uma relação não está na 3FN, pode ser decomposta em relações que respeitam a 3FN
  - Decomposição sem perdas
  - Preservando as dependências

Anomalias na 3FN

- Exemplo: os alunos inscrevem-se em turmas. A mesma disciplina tem várias turmas. As DF são:
  - DF_1 {disc_id, aluno_id} $\rightarrow$ {turma_id}
  - DF_2 {turma_id} $\rightarrow$ {disc_id}
- Relação possível: {aluno_id, disc_id, turma_id}
- Chaves: {aluno_id, turma_id} e {aluno_id, disc_id}
- Está em 3FN: o determinante de DF_1 é chave, o dependente de DF_2 é primo
- No entanto tem redundância…

Anomalias na 3FN

- Repete-se {turma_id, disc_id} para cada aluno nessa turma
- Se uma turma não tiver alunos, temos que usar valores nulos (NULL) mas não podemos porque aluno_id faz parte da chave

Forma normal Boyce-Codd

- Mais geral que a 3FN, deve-se testar no caso em que há várias chaves candidatas tais que:
  - as chaves têm mais do que um atributo
  - e têm pelo menos um atributo em comum
- Caso contrário, 3FN = BCNF
  - Uma relação com 2 atributos está em BCNF
- Uma relação está em BCNF se os únicos determinantes forem superchaves

Definição de BCNF

- Uma relação R está em BCNF se o determinante de cada DF não-trivial é uma superchave de R

Para cada DF: $X \rightarrow Y \in F^+$

$Y \subseteq X$ (a DF é *trivial*)

ou

X é uma superchave

Podem-se começar a testar as DF em F e só passar para F^+ se todas forem chave

Decomposição BCNF (simples)

- Aplicar a regra da união às DF de F
- Calcular F^+
- $\mathcal{R} = R$
- Enquanto houver um $R_i \in \mathcal{R}$ não em BCNF
  - Seja $X \rightarrow Y$ não trivial, existente em R_i e F_i mas onde X não é chave e $X \cap Y = \emptyset$
    - $\mathcal{R} = (\mathcal{R} — R_i) \cup \{(R_i — Y) \cup X, XY\}$

1. F_i é o conjunto de dependências de F^+ que só contêm atributos de R_i (chamado projeção de F sobre R_i): $F_i = \pi(F) = \{ X \rightarrow Y \mid X \rightarrow Y \in F^+ \text{ e } XY \subseteq R_i\}$
2. Corre em tempo exponencial devido ao cálculo dos F_i

Otimização

- Para testar violações em R basta usar F. Para testar violações nas decomposições de R devemos usar F^+
- Evita-se fragmentação excessiva das relações ao decompor em $X \rightarrow X^+$ em vez de $X \rightarrow Y$

- Seja $X \rightarrow Y$ não trivial, onde X não é chave
- Calcular X^+
- Decompor em $R_1 = X^+$ e $R_2 = X \cup (R - X^+)$
- Se $Z \rightarrow W$, não trivial, existe em R_i e F_i mas Z não é chave, decompor R_i recursivamente

Exemplo

- Decompor em BCNF por causa de $X \rightarrow Y$ ($X \cap Y = \emptyset$):
  - $R_1 = R - Y$, $R_2 = XY$
  - R_1 {disc_id, turma_id} R_2 {aluno_id, turma_id}
  - Não é possível colocar em BCNF sem perder a DF_1
  - Mesmo aluno em 2 turmas da mesma disciplina em R_2!
  - Por isso, para se verificar {aluno_id, disc_id} $\rightarrow$ {turma_id} tem que se efetuar uma junção
- Por outro lado, se a relação ficar em 3FN poderemos ter que usar valores nulos quando queremos registar turmas sem alunos inscritos

BCNF: objetivo

- A vantagem de BCNF é garantir que todas as FD são verificadas por restrições de chaves
  - Mais fácil de implementar, o próprio SGBD o faz
  - Caso contrário tem que ser programada lógica para o efeito (por exemplo com *triggers*)
- É mais restrita que 3FN porque não admite a condição adicional que os atributos em X — Y possam ser primos
- Mas nem sempre é possível decompor em BCNF e preservar as dependências

3FN e BCNF

- Qualquer esquema pode ser decomposto em 3FN e BCNF “sem perdas”
- Só 3FN preserva as dependências, com BCNF nem sempre é possível
- O risco de violar uma DF não preservada em BCNF pode ser maior do que a redundância de 3FN
  - 3FN é geralmente preferida neste caso
  - Ou então definem-se *triggers* ou asserções para poder verificar que a DF perdida é respeitada