Design de Bases de Dados Relacionais

- Tópicos:
  - Objetivos com o Desenho de Bases de Dados
  - Dependências funcionais
  - 1ª Forma Normal
  - Decomposição
  - Forma Normal de Boyce-Codd
  - 3ª Forma Normal
  - Dependências multivalor
  - 4ª Forma Normal
  - Visão geral sobre o processo de design
- Bibliografia:
  - Capítulo 7 do livro recomendado (7ª edição – Cap. 8 na 6ª edição)
  - Capítulos 4, 5, 6 e 7 do livro *The theory of relational databases*
    - Neste último, a matéria está muito mais detalhada, e o livro está disponível online, gratuitamente, na página do autor.

Objectivos com o Design de BDs Relacionais

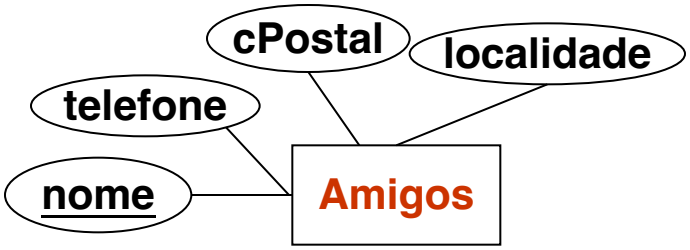
- Pretende-se encontrar “bons” conjuntos de esquemas relações, para armazenar os dados.
- Um “mau” design pode levar a:
  - Repetição de dados.
  - Impossibilidade de representar certos tipos de informação.
  - Dificuldade na verificação da integridade
- Objectivos do Design:
  - Evitar dados redundantes
  - Garantir que as relações relevantes sobre dados podem ser representadas
  - Facilitar a verificação de restrições de integridade.

1ª Forma Normal

- Um esquema R diz-se na 1ª forma normal se os domínios de todos os seus atributos são atómicos.
- Uma domínio é atomico se os seus elementos forem unidades indivisíveis.
  - Exemplo de domínios não atómicos:
    - Conjuntos (e.g. de nomes, de telefones), atributos compostos (e.g. com nome de rua, nº de porta, código postal e localidade)
    - Identificações com partes distintas (e.g. nº de carta de condução E-111222-5, nº de BI com último dígito)
- Os valores não atómicos complicam o armazenamento e encorajam repetições desnecessárias de dados.
  - Acresce que complicam desnecessariamente a definição de dependências funcionais, e não trazem nenhuma vantagem!
- Daqui para a frente, assumiremos que todas os esquemas de relações estão já na 1ª Forma Normal.

Exemplo de mau design

- Este esquema dá origem à tabela:
  Amigos = (*nome*, *telefone*, *cPostal*, *localidade*)



- Este esquema não é ”bom design”

nome	telef	cPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

Repetido!

Redundante!

- Redundância:
  - Os valores de (*cPostal, localidade)* são repetidos para cada amigo com um mesmo código postal
  - Desperdiça-se espaço de armazenamento
  - Dá azo a inconsistências
  - Complica bastante a verificação da integridade dos dados
- Dificuldade de representar certa informação
  - Não se pode armazenar informação do código postal de uma localidade sem que existam amigos dessa localidade.
    - Por vezes podem usar-se valores nulos, mas estes são difíceis de gerir.

Objectivos da Normalização

- Decidir se o esquema R já está num "bom" formato.
- Se não estiver, decompor R num conjunto de esquemas $\{R_1, R_2, ..., R_n\}$ tal que:
  - cada um deles está num "bom" formato
  - A decomposição é sem perdas
- A teoria é baseada em
  - Dependências funcionais
  - Dependências multivalor

Decomposição sem perdas

- Todos os atributos do esquema original (R) devem aparecer na decomposição em (R_1, R_2):

$$R = R_1 \cup R_2$$

- Decomposição sem perdas
  - Para todas as (instâncias de) relações r sobre o esquema R:

$$r = \Pi_{R1}(r) \bowtie \Pi_{R2}(r)$$

- Há perda de informação sse os atributos comuns a R_1 e R_2 não são chave nem em R_1 nem em R_2
- Formalmente, a decomposição de R em R_1 e R_2 é sem perdas se pelo menos uma das dependências abaixo pertence a F^+:

$$R_1 \cap R_2 \rightarrow R_1$$

$$R_1 \cap R_2 \rightarrow R_2$$

Normalização por uso de Dependências

- Quando decompomos um esquema R com dependências F, em R_1, R_2,.., R_n queremos
  - Decomposição sem perdas: Por forma a não se perder informação.
  - Preservação de dependências: Seja F_i o conjunto de dependências de F^+ que só contêm atributos de R_i.
    - A decomposição preserva as dependências se

$$(F_1 \cup F_2 \cup \ldots \cup F_n)^+ = F^+$$

    - Sem preservação de dependências, a garantia de integridade pode obrigar à computação de junções, sempre que se adicionam, apagam ou actualizam relações da base de dados. Tal pode tornar-se bastante ineficiente.
  - Não haja redundância: Veremos à frente como …

Exemplo

- $R = (Nome, CP, Local)$ com $F = \{Nome \rightarrow CP,\ CP \rightarrow Local\}$
- Decomposição de R em $R_1 = (Nome, CP)$ e $R_2 = (CP, Local)$
  - Decomposição sem perdas:

    $$R_1 \cap R_2 = \{CP\} \text{ e } CP \rightarrow CP,\ Local$$

  - Preserva as dependências:

    $(F_1 \cup F_2)^+ = F^+$ onde $F_1 = \{Nome \rightarrow CP, ...\}$ e $F_2 = \{CP \rightarrow Local, ...\}$
- Decomposição de R em $R_1 = (Nome, CP)$ e $R_2 = (Nome, Local)$
  - Decomposição sem perdas:

    $$R_1 \cap R_2 = \{Nome\} \text{ e } Nome \rightarrow Nome,\ CP$$

  - **Não** preserva as dependências:

    $(F_1 \cup F_2)^+ \neq F^+$ onde $F_1 = \{Nome \rightarrow CP, ...\}$ e $F_2 = \{Nome \rightarrow Local, ...\}$
    - não se pode verificar $CP \rightarrow Local$ sem calcular $R_1 \bowtie R_2$

Teste de Preservação de Dependências

Para verificar se $\alpha \rightarrow \beta$ é preservada na decomposição R em R_1, R_2, …, R_n aplica-se o seguinte teste:

- *result* := α
  while (alterações a *result*) **do**
    for each R_i na decomposição
      $result := result \cup ((result \cap R_i)^+ \cap R_i)$
- Se *result* contém todos os atributos em β, então $\alpha \rightarrow \beta$ é preservada.

- Aplica-se este teste a todas as dependências de F, para verificar se a decomposição preserva as dependências
- Ao contrário do cálculo de F^+ e $(F_1 \cup F_2 \cup \ldots \cup F_n)^+$ (que tem complexidade exponencial), este procedimento tem complexidade polinomial.

Exemplo

- *R = (Nome,CP,Local) com F = {Nome → CP, CP → Local}*
- *Decomposição de R em R_1=(Nome,CP) e R_2=(Nome,Local)*
  - *result* := α
    while (alterações a *result*) **do**
    for each R_i na decomposição
    result := result $\cup$ ((*result* $\cap$ R_i)$^+$ $\cap$ R_i)
  - *Nome → CP:*
    - R_1: {Nome} $\cup$ ({Nome,CP,Local} $\cap$ {Nome,CP}) = {Nome,CP}
    - R_2: já não é preciso verificar porque CP é contido em result
  - *CP → Local:*
    - R_1: {CP} $\cup$ ({CP,Local} $\cap$ {Nome,CP}) = {CP}
    - R_2: {CP} $\cup$ ({} $\cap$ {Nome,Local}) = {CP}
    - Não preserve a dependência!

Normalização por uso de Dependências

- Quando decompomos um esquema *R* com dependências *F,* em R_1, R_2,.., R_n queremos
  - Decomposição sem perdas: Por forma a não se perder informação.
  - Preservação de dependências: Por forma a facilitar a verificação da integridade.
  - Não haja redundância: Forma Normal de Boyce-Codd

Evitar Redundância

- Obter um conjunto de esquemas que evite redundâncias
- Há redundância num esquema R sse existir alguma dependência funcional $\alpha \rightarrow \beta$ não trivial definida sobre R, em que α não é superchave em R
  - Dito de outra forma: Não há redundância sse para toda a dependência funcional $\alpha \rightarrow \beta$ não trivial definida sobre R, α é superchave em R
- Logo, não há redundância em R sse para toda $\alpha \rightarrow \beta \in F^+$:
  - $\beta \subseteq \alpha$ **ou**
  - Se $\alpha \subseteq R$ e $\beta \subseteq R$ então $\alpha \rightarrow R \in F^+$ (ou seja $R \subseteq \alpha^+$)

Forma Normal de Boyce-Codd

Um esquema R diz-se na **Forma Normal de Boyce-Codd**, **BCNF**, relativamente a um conjunto de dependências F, sse para toda a dependência em F^+ da forma $\alpha \rightarrow \beta$, onde $\alpha \subseteq R$ e $\beta \subseteq R$, pelo menos uma das condições é verdadeira:

- $\alpha \rightarrow \beta$ é trivial (i.e., $\beta \subseteq \alpha$)
- α é superchave de R (i.e., $\alpha \rightarrow R$)

- Evita redundâncias
- Verificação de dependências funcionais, definidas sobre atributos de R, limita-se a verificação de chaves.

Exemplo

- $R = (A, B, C)$
  $F = \{A \rightarrow B,\ \ B \rightarrow C\}$
- R não está em BCNF:
  - $B \rightarrow C \in F$ e $\{B\}^+ = \{B,C\} \neq R = \{A,B,C\}$
- Decomposição em $R_1 = (A, B)$, $R_2 = (B, C)$
  - R_1 e R_2 estão na BCNF

    $F_1 = \{A \rightarrow B,...\}$ e $\{A\}^+ = \{A,B\} = R_1$

    $F_2 = \{B \rightarrow C,...\}$ e $\{B\}^+ = \{B,C\} = R_2$
  - Decomposição sem perdas:

    $R_1 \cap R_2 = \{B\}$ e $B \rightarrow BC \in F^+$
  - Preserva as dependências:

    $F_1 = \{A \rightarrow B,...\}$ e $F_2 = \{B \rightarrow C,...\}$ e $(F_1 \cup F_2)^+ = F^+$

Teste para BCNF

- Para determinar se uma dependência não trivial $\alpha \rightarrow \beta$ é causa de violação de BCNF
  1. calcular α^+ (fecho de atributos em α), e
  2. Verificar se inclui todos os atributos de *R*, i.e. é superchave de *R*. Se incluir, está na BCNF; caso contrário não está.
- Teste simplificado: Em vez de verificar para todas as dependências de F^+, verificar apenas para as dependências numa cobertura.
  - ✷ Se nenhuma das dependências da cobertura violar a BCNF, então nenhuma das dependências de F^+ viola a BCNF.

Teste para BCNF

- Mas cuidado com esquemas já decompostos quando há dependências que não estão na cobertura (canónica) mas que estão definidas sobre o esquema!!
  - E.g. Seja $R\,(A, B, C, D)$, com $F = \{\, A \rightarrow B,\ B \rightarrow C\}$
    - Decomposição de R em $R_1(A,B)$ e $R_2(A,C,D)$
    - Nenhuma das dependências em F contém só atributos de (A,C,D)
    - Por isso, podemos (erradamente) pensar que R_2 satisfaz BCNF.
    - Mas a dependência $A \rightarrow C \in F^+$ e portanto também pertence a F_2 , demonstrando que R_2 não está na BCNF.

Algoritmo para Decomposição BCNF

result := {*R*};
done := false;
calcular F^+;
while (**not** *done)* **do**
  if (há um esquema R_i em *result* que não está na BCNF)
    then begin
      Seja $\alpha \rightarrow \beta$ uma dependência sobre R_i tal que

$$\alpha \rightarrow R_i \notin F^+ \textbf{ e } \alpha \cap \beta = \varnothing;$$

$$result := (result - R_i) \cup (R_i - \beta) \cup (\alpha, \beta);$$

      end
    else *done* := **true;**

Nota: cada R_i está na BCNF, e a decomposição é sem perdas.

Exemplo de Decomposição BCNF

- *Amigos* = (*nome, telefone, código_postal, localidade)*
- *F* = {*nome* → *telefone, código_postal*
  código_postal → *localidade* }
- Decomposição:
  - result = {Amigos}
    - *código_postal* → *localidade* ∈ F^+
    - *código_postal* → *nome, telefone, código_postal, localidade* ∉ F^+
      - *i.e. código_postal* não é chave
    - {*código_postal*} ∩ {*localidade*} = ∅
  - result = {(*nome, telefone, código_postal), (código_postal, localidade)}*
  - Já está na BCNF.

Outro exemplo

- *R = (balcão, localidade, activos, cliente, num_emprestimo, valor)*
  F = {balcão → activos, localidade
  num_emprestimo → valor, balcão }
  Chave candidata = *{num_emprestimo, cliente}*
- Decomposição
  - *result = {R}*
    - *balcão → activos, localidade* $\in F^+$ e *balcão* não é chave em R
    - R_1 = (*balcão, activos, localidade)*
    - R_2 = *(balcão, cliente, num_emprestimo, valor)*
  - *result =* $\{R_1, R_2\}$
    - *num_emprestimo → valor, balcão* $\in F_2^+$ e *num_emprestimo* não é chave em R_2
    - R_3 = *(num_emprestimo , valor, balcão)*
    - R_4 = *(cliente, num_emprestimo)*
  - *result =* $\{R_1, R_3, R_4\}$
  - Já está na BCNF

Teste de Decomposição BCNF

- Para verificar se R_i numa decomposição de R está na BCNF, pode-se testar R_i relativamente à restrição de F a R_i (i.e. todas as dependências em F^+ que só contêm atributos de R_i).
  - No entanto, obriga ao cálculo de F^+
- Pode-se usar uma cobertura de dependências sobre R mas com o seguinte teste:

  - Para todo o subconjunto α de atributos de R_i, verificar se α^+ (fecho relativo a F) não inclui nenhum atributo de $R_i - \alpha$, ou inclui todos os atributos de R_i.
    - Se a condição do teste for violada para um subconjunto α de atributos de R_i, então a dependência funcional $\alpha \rightarrow (\alpha^+ - \alpha) \cap R_i$ pertence a F^+.
    - Usa-se essa dependência para decompor R_i

BCNF e preservação de dependências

- Nem sempre é possível obter uma decomposição BCNF que preserve as dependências.
- $R = (J, K, L)$
  $F = \{JK \rightarrow L$
  $L \rightarrow K\}$
  Duas chaves candidatas = JK e JL
- R não está na BCNF
- Nenhuma decomposição de R preserva

  $JK \rightarrow L$

Exemplo

gestores-bal

balcão	gestor-conta
Lisboa	Ana
Porto	Francisco
Faro	Maria

Redundância!

clientes gestores

balcão	cliente	gestor-conta
Lisboa	Pedro	Ana
	Carla	Francisco
Porto	Pedro	Francisco
	Pedro	Francisco
Faro	Pedro	Maria

- Considere o esquema

  Gestores = (balcão, cliente, gestor-conta),

- com as dependências:

  1. *gestor-conta → balcão (um gestor só trabalha num balcão)*
  2. *cliente, balcão → gestor-conta (cada cliente só tem um gestor por balcão)*

- Não está na BCNF (a dependência 1 não é trivial e *gestor-conta* não é superchave – não determina *cliente*) ⇒ Redundância
- Usando a dependência 1 para decompor Gestores, obtemos:

  Gestores-Bal = (balcão, gestor-conta) e *Clientes = (cliente, gestor-conta)*

- Agora já está na BCNF
- No entanto, não se preservam as dependências funcionais:
  - A dependência 2 não se pode verificar numa só relação, i.e. não podemos verificar que cada cliente só tem um gestor por balcão analisando separadamente as relações Gestores-Bal e Clientes.

Normalização por uso de Dependências

- Quando decompomos um esquema *R* com dependências *F,* em R_1, R_2,.., R_n queremos
  - Decomposição sem perdas: Por forma a não se perder informação.
  - Não haja redundância: Garantido pela Forma Normal de Boyce Codd
  - Preservação de dependências: Para que a verificação da integridade se possa fazer relação a relação
- O algoritmo para decomposição BCNF:
  - Garante uma decomposição sem perdas.
  - Garante que as relações resultantes estão na BCNF.
  - Não garante a preservação das dependências!

Motivação para a 3ª Forma Normal

- Há situações em que:
  - a BCNF não preserva as dependências, e
  - a eficiência na verificação de integridade aquando de alterações é importante
- Solução: definir uma forma normal mais fraca – 3ª Forma Normal:
  - Admite alguma redundância (o que pode trazer problemas, como veremos à frente)
  - Mas as dependências podem ser verificadas relação a relação, isoladamente
  - É sempre possível fazer uma decomposição sem perdas para a 3NF, que preserva as dependências.

Exemplo Motivador

- O esquema *Gestores = (balcão, cliente, gestor-conta)*, com as dependências:
  1. *gestor-conta* → *balcão*
  2. *cliente, balcão* → *gestor-conta*

  não estava na BCNF por causa da primeira dependência.
- As duas chaves candidatas de *Gestores são* {*cliente, balcão}* e {*gestor-conta, cliente}*. Mas:
  - Ao decompor *Gestores* com base na 1ª dependência, *balcão* vai ficar numa relação diferente daquela onde fica *cliente.*
  - Logo deixa de ser possível verificar a chave candidata {*cliente, balcão} (2ª dependência funcional)* de *Gestores* numa só relação!
- Solução:
  - Para se continuar a poder verificar a chave candidata da relação original, a solução é não permitir a decomposição de um esquema com base numa dependência que à direita contenha apenas alguns dos atributos de uma chave candidata.

3ª Forma Normal

Um esquema R está na 3ª Forma Normal (3FN) sse para toda:

$$\alpha \rightarrow \beta \in F^+$$

pelo menos uma das condições é verdadeira:

- $\alpha \rightarrow \beta$ é trivial (i.e., $\beta \subseteq \alpha$)
- α é superchave de R (i.e., $\alpha \rightarrow R \in F^+$)
- Todo atributo $A \in (\beta - \alpha)$ está contido numa chave candidata de R.

  (NOTA: cada um dos atributos pode pertencer a uma chave candidata distinta)

- A 3ª condição relaxa a BCNF para garantir que é possível uma decomposição com preservação de dependências
- Se R está na BCNF então está também na 3FN

Exemplo

- Esquema *Gestores = (balcão, cliente, gestor-conta)* com as dependências:
  1. *gestor-conta → balcão*
  2. *cliente, balcão → gestor-conta*

gestores

balcão	cliente	gestor-conta
Lisboa	Pedro	Ana
Porto	Carla	Francisco
Porto	Pedro	Francisco
Faro	Pedro	Maria

Redundância!

- Tem duas chaves candidatas: {*cliente, balcão*} e {*gestor-conta, cliente*}.
- Está na 3ªFN porque: 1. balcão ∈ {*cliente, balcão*} que é chave candidata
  2. {*cliente, balcão*} é superchave
- Não está na BCNF porque dep. func. 1. não é trivial e *gestor-conta* não é superchave.
- Decomposição BCNF:

Não preserva a dependência funcional cliente, balcão → gestor-conta

clientes

cliente	gestor-conta
Pedro	Ana
Carla	Francisco
Pedro	Francisco
Pedro	Maria

gestores-bal

balcão	gestor-conta
Lisboa	Ana
Porto	Francisco
Faro	Maria

Teste para 3FN

- Optimização: Basta verificar para uma cobertura de F (não é necessário verificar para toda a dependência em F^+)
- Usar fecho de atributos para verificar, em toda a dependência $\alpha \rightarrow \beta$, se α é superchave.
- Se α não for superchave, há que verificar se todo a atributo em β- α pertence a alguma chave candidata de *R*
  - Este teste é bastante ineficiente, pois envolve o cálculo de chaves candidatas
  - Pode demonstrar-se que verificar se um conjunto de esquemas está na 3FN tem complexidade NP-hard (muito complexo!)
  - No entanto, é possível fazer a decomposição para a 3FN em tempo polinomial

Algoritmo de Decomposição para 3FN

Seja F_c uma cobertura canónica de F;
$i := 0$;
for each $\alpha \rightarrow \beta \in F_c$ **do**
    if nenhum dos esquemas R_j, $1 \leq j \leq i$ contém $\alpha\ \beta$
    then begin
        $i := i + 1$;
        $R_i := \alpha\ \beta$
    end
if nenhum dos esquemas R_j, $1 \leq j \leq i$ contém uma chave candidata de R
then begin
    $i := i + 1$;
    R_i := uma (qualquer) chave candidata de R;
    end
return $(R_1, R_2, ..., R_i)$

No fim, remove-se esquemas R_j tal que R_j está contido noutro R_k

Exemplo

- Considere o esquema:

  Info-Gestores = (balcão, cliente, gestor-conta, gabinete)

- As dependências definidas sobre este esquema são:

  gestor-conta $\rightarrow$ *balcão, gabinete*

  cliente, balcão $\rightarrow$ *gestor-conta*

- As chaves candidatas são:

  {*cliente, balcão*} e {gestor-conta,cliente}

- Info-Gestores não está na 3FN. Considerando a dependência funcional *gestor-conta* $\rightarrow$ *balcão, gabinete*, verificamos que:
  - *gestor-conta* não é superchave;
  - *gabinete* $\notin$ chave candidata.
- O ciclo **for** do algoritmo leva à introdução dos seguintes esquemas na decomposição:

  Gestores-gab = (gestor-conta, balcão, gabinete)

  Gestores = (cliente, balcão, gestor-conta)

- Como *Gestores* contém uma chave candidata de *Info-Gestores,* {cliente,balcão}, o processo de decomposição termina.

Exemplo

- Esquema *Gestores = (balcão, cliente, gestor-conta, gabinete)* com as dependências:
  1. *gestor-conta → balcão, gabinete*
  2. *cliente, balcão → gestor-conta*

Redundância!

info-gestores

balcão	cliente	gestor-conta	gabinete
Lisboa	Pedro	Ana	1.1
Porto	Carla	Francisco	2.1
Porto	Pedro	Francisco	2.1
Faro	Pedro	Maria	3.1

- Não está na BCNF nem na 3NF

- Decomposição BCNF:

Não preserva a dependência funcional cliente, balcão → gestor-conta

gestores-bal

balcão	gestor-conta	gabinete
Lisboa	Ana	1.1
Porto	Francisco	2.1
Faro	Maria	3.1

clientes

cliente	gestor-conta
Pedro	Ana
Carla	Francisco
Pedro	Francisco
Pedro	Maria

- Decomposição 3NF:

Redundância!

gestores-gab

balcão	cliente	gestor-conta
Lisboa	Pedro	Ana
Porto	Carla	Francisco
Porto	Pedro	Francisco
Faro	Pedro	Maria

gestores

balcão	gestor-conta	gabinete
Lisboa	Ana	1.1
Porto	Francisco	2.1
Faro	Maria	3.1

Outro exemplo

- Considere o esquema:

$$R = (A,B,C)$$

- Com as dependências:

$$A \rightarrow C$$
$$B \rightarrow C$$

- A chave candidata é:

$$\{ A, B \}$$

- O ciclo **for** do algoritmo, leva à introdução dos seguintes esquemas na decomposição:

$$R_1 = (A,C)$$
$$R_2 = (B,C)$$

- Como nem R_1 nem R_2 contém a chave candidata de R o processo de decomposição adiciona ainda:

$$R_3 = (A, B)$$

Mais um exemplo

- Considere o esquema Semestre=(C,P,H,S,A,N) representando a informação relativa à componente prática de uma dada cadeira num semestre, onde C=Cadeira, P=Professor, H=Hora, S=Sala, A=Aluno e N=Nota.
- Considere as seguintes dependências funcionais:

  $F = \{C \rightarrow P,\ HS \rightarrow C,\ HP \rightarrow S,\ CA \rightarrow N,\ HA \rightarrow S,\ HSP \rightarrow CS\}$
- Primeiro determina-se uma cobertura canónica:

  $F_c = \{C \rightarrow P,\ HS \rightarrow C,\ HP \rightarrow S,\ CA \rightarrow N,\ HA \rightarrow S\}$
- Chaves candidatas: {A,H}
- Ciclo **for**: cria as seguintes 5 relações, correspondentes às 5 dependências funcionais da cobertura canónica:

  R_1=(C,P) R_2=(H,S,C) R_3=(H,P,S) R_4=(C,A,N) R_5=(H,A,S)
- Como já existe uma relação (R_5) contendo uma chave candidata ({A,H}), não é necessário criar mais relações.

Propriedade do algoritmo de Decomposição

- O algoritmo descrito garante que:
  - Todo o esquema R_i está na 3FN
  - A decomposição preserva as dependências e é sem perdas
- Note que só é garantido que se preservam as dependências no conjunto de todas as relações da decomposição. Mesmo que exista um subconjunto dessas relações que contenha todos os atributos da relação original, não se garante que, nele, as dependências sejam preservadas.
- A decomposição de $R = (A,B,C)$, com $A \rightarrow C$ e $B \rightarrow C$, em:

  $R_1 = (A, C)$ $R_2 = (B,C)$ $R_3 = (A, B)$

  preserva as dependências:
  - $F_1 = \{A \rightarrow C,...\}$, $F_2 = \{B \rightarrow C,...\}$, $F_3 = \{...\}$ e $(F_1 \cup F_2 \cup F_3)^+ = F^+$
- Mas $(F_1 \cup F_3)^+ \neq F^+$, apesar de $R_1 \cup R_3 = R$!!
- Em particular $B \rightarrow C \in F^+$ mas $B \rightarrow C \notin (F_1 \cup F_3)^+$

Propriedade do algoritmo de Decomposição

- O resultado do algoritmo não é único pois depende:
  - da cobertura canónica utilizada
  - da chave candidata de R utilizada
  - nalguns casos, também da ordem pela qual se consideram as dependências funcionais

BCNF versus 3NF

- É sempre possível decompor um esquema, num conjunto de esquemas na 3FN em que:
  - a decomposição é sem perdas
  - as dependências são preservadas
  - Mas pode haver alguma redundância!!
- É sempre possível decompor um esquema, num conjunto de esquemas na BCNF em que
  - a decomposição é sem perdas
  - não há redundância
  - Mas nem sempre se podem preservar as dependências!!

BCNF ou 3FN?

- Objectivos do design, numa primeira fase:
  - BCNF.
  - Decomposição sem perdas.
  - Preservação de dependências.
- Se tal não for possível, então há que optar por uma de
  - Não preservação de dependências
  - Alguma redundância (devido ao uso da 3FN)
- O SQL não fornece nenhuma forma directa de impor dependências que não sejam superchaves. Pode fazer-se usando **assertion** mas isso é muito ineficiente.
  - Mesmo quando a decomposição preserva as dependências, com o SQL nem sempre é possível testar de forma eficiente dependências cujo lado esquerdo não seja chave.

Teste de Dependências em várias Relações

- Se a decomposição não preserva as dependências, podemos criar uma **materialized view** para cada uma das dependências $\alpha \rightarrow \beta \in F_c$ que não é preservada
  - O Oracle permite criar este tipo de views com o comando

create materialized view

- A **materialized view** deve ser definida para a projecção em α β da junção das relações decompostas.
- A dependência funcional deve ser imposta como chave candidata da **materialized view**.
- Isto implica um overhead de
  - espaço: há que guardar as views.
  - tempo: há que manter as views actualizadas