Dependências funcionais e normalização

- 1ª Forma Normal
- 2ª Forma Normal
- Objectivos na Concepção de Bases de Dados
- Dependências funcionais
- Decomposição
- Forma Normal de Boyce-Codd
- 3ª Forma Normal
- Dependências multi-valor
- 4ª Forma Normal
- Visão geral sobre o processo de concepção

1ª Forma Normal

- Um esquema *R* diz-se na 1ª forma normal se:
  - os domínios de todos os seus atributos são atómicos;
  - não pode haver repetição de registos.
- Um domínio é atómico se os seus elementos forem unidades indivisíveis.
  Exemplo de domínios não atómicos:
  - Atributos “naturalmente” compostos: Nomes, Endereços, etc.
  - Atributos com várias partes: Números de telefones com indicativos; B.I. com o número de validação.
- Os valores não atómicos complicam o armazenamento e encorajam repetições desnecessárias de dados.

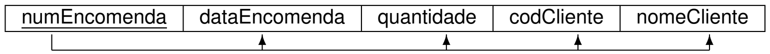
Daqui para a frente, assume-se que todas os esquemas de relações estão já na 1ª Forma Normal.

2ª Forma Normal

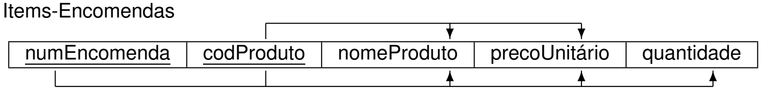
- Um esquema *R* diz-se na 2ª forma normal se:
  - está na 1ª forma normal;
  - cada atributo não chave tem de depender da chave da tabela na totalidade, e não apenas de uma parte dessa chave.
    - se a chave primária é simples (um só atributo), então a relação está na 2ª forma normal.
    - se a chave primária é composta (mais do que um atributo) e existe um atributo que depende somente de parte da chave primária, então a relação não está na 2ª forma normal.

Por exemplo:

Encomendas



Items-Encomendas



Daqui para a frente, assume-se que todas os esquemas de relações estão já na 2ª Forma Normal.

Objectivos na Concepção de Bases de Dados

Pretendem-se encontrar “bons” conjuntos de esquemas de relações para armazenar os dados.

Um “má” concepção pode levar a:

- Repetição de dados.
- Inconsistências devidas às operações de introdução, alteração, apagar de dados.
- Impossibilidade de representar certos tipos de informação.
- Dificuldade nas verificações de restrições de integridade.

Objectivos na Concepção:

- Evitar dados redundantes.
- Garantir que as relações relevantes sobre dados podem ser representadas.
- Facilitar a verificação de restrições de integridade.

Exemplo

Considere o esquema simples:

N.S.S.	Nome	Classificação	Vencimento/h	Horas Trab.
123-22	Abel	8	10	40
231-31	Silva	8	10	30
131-24	Sousa	5	7	30
434-26	Guiomar	5	7	32
612-67	Miguel	8	10	40

`Vencimento/h` depende de `Classificação`: este tipo de dependências (funcionais) entre atributos levanta problemas de:

- redundância:
  - 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 de uma nova categoria de Classificação/Vencimento sem que haja um funcionário nessa categoria.

Decomposição de Esquemas de Relações

As dependências funcionais podem servir para identificar, e para indicar o caminho para uma melhor concepção global

Substituir uma (ou mais) relações por um conjunto de relações "mais pequenas"

N.S.S.	Nome	Classificação	Horas Trab.
123-22	Abel	8	40
231-31	Silva	8	30
131-24	Sousa	5	30
434-26	Guiomar	5	32
612-67	Miguel	8	40

Classificação	Vencimento/h
8	10
5	7

menos redundância; mais fácil manter a consistência dos dados; é possível acrescentar novos pares `Classificação/Vencimento`.

Problemas com a Decomposição de Esquemas de Relações

Ao fazer-se uma decomposição é necessário analisar se:

- a decomposição é necessária?
- a decomposição cria novos problemas?

- Formas normais
- Decomposição sem perdas
- Preservação das relações:
  - as restrições mantêm-se sem que seja necessário fazer junções entre relações;
  - as restrições verificam-se nas relações "menores".

Exemplo

Considere o esquema simples:

Amigos = (<u>nome</u>, telefone, codigoPostal, localidade)

nome	telefone	codigoPostal	localidade
Maria	1111	2815	Caparica
João	2222	1000	Lisboa
Pedro	1112	1100	Lisboa
Ana	3333	2815	Caparica

- Redundância: os valores de `codigoPostal` e `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 hajam amigos dessa localidade. Podem usar-se valores nulos, mas estes são difíceis de gerir.

Decomposição

Decompor o esquema Amigos em:

`Amigos1 = (nome,telefone, codigoPostal)`
`CPs = (codigoPostal, localidade)`

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 que "façam sentido" sobre o esquema R:

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

Note-se que o "façam sentido" depende do problema concreto.

Exemplo de decomposição sem perdas

Decomposição de `Amigos` em `Amigos1` e `CPs`:

r

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

$\Pi_{\text{Amigos1}}(r)$

nome	telef.	CPostal
Maria	1111	2815
João	2222	1000
Pedro	1112	1100
Ana	3333	2815

$\Pi_{\text{CPs}}(r)$

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$$\Pi_{\text{Amigos1}}(r) \bowtie \Pi_{\text{CPs}}(r) = r$$

Exemplo de decomposição com perdas

Decomposição de `CPs` em: `CP1 = (CPostal)` e `Locs = (localidade)`

r

CPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

$\neq$

$\Pi_{\text{CP1}}(r) \bowtie \Pi_{\text{Locs}}(r)$

CPostal	localidade
2815	Caparica
2815	Lisboa
1000	Caparica
1000	Lisboa
1100	Caparica
1100	Lisboa

$\Pi_{\text{CP1}}(r)$

CPostal
2815
1000
1100

$\Pi_{\text{Locs}}(r)$

localidade
Caparica
Lisboa

- Perdeu-se a informação de qual os CPs das localidades!
- Decompor parecia bom para evitar redundâncias.
- Mas decompor demais pode levar à perda de informação.

Outro exemplo com perdas

Decomposição de `Amigos` em: `Amigos2 = (nome,telefone,localidade)` e `Loc = (localidade,CPostal)`.

r

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

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

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

$\Pi_{\text{Amigos2}}(r)$

nome	telef.	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{Loc}}(r)$

localidade	CPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

- Perdeu-se a informação de qual é o CP do João (e do Pedro)!
- O que torna esta decomposição diferente da primeira?

Temos de ter critérios que nos permitam decompor uma relação, sem perda de informação.

Objectivo: chegar a um conjunto da seguinte forma

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

Dependências funcionais

- Restrições sobre o conjunto de relações possíveis.
- Exige que os valores num conjunto de atributos determinem univocamente os valores noutro conjunto de atributos.
- São uma generalização da noção de chave.
- Seja R o esquema duma relação e $\alpha \subseteq R$ e $\beta \subseteq R$. A dependência funcional:

$$\alpha \rightarrow \beta$$

  é verdadeira em R sse, para toda a relação possível (i.e. "que faça sentido") $r(R)$, sempre que dois tuplos t_1 e t_2 de r têm os mesmos valores em α, também têm os mesmos valores em β:

$$\forall_{t_1, t_2 \in r} t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$

Dependências Funcionais (continuação)

- De forma equivalente.
  A dependência funcional $\alpha \rightarrow \beta$ é verdadeira em R sse

$$\forall_{a \in \mathrm{dom}(\alpha)} \Pi_\beta(\sigma_\alpha = a(r))$$

  tem no máximo 1 tuplo.
- Exemplo: Seja $r(A, B)$:

A	B
1	4
1	5
3	7

  Nesta instância, $A \rightarrow B$ não é verdadeira, mas $B \rightarrow A$ é.

Dependências Funcionais

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1
a_2	b_1	c_3	d_2

$$AB \rightarrow C \qquad AB \not\rightarrow D$$

- AB não é uma chave
- A verificação para uma dada instância da relação não valida uma dependência funcional
- As dependências funcionais são restrições de integridade que tem de ser satisfeitas por todos os valores possíveis no esquema de relações.

Dependências Funcionais

Casos extremos

- $\{\} \to \alpha$
  Só se verifica se na relação r todos os tuplos têm o mesmo valor em α (nunca deve ser permitido).
- $\alpha \to \{\}$
  Verifica-se para toda a relação r e conjunto de atributos α.

Dependência Trivial Diz-se que uma dependência é trivial se é satisfeita por todas as relações (quer façam sentido ou não) sobre um esquema.
Por exemplo:

$\texttt{nomeCliente,numEmprestimo} \to \texttt{nomeCliente}$
$\texttt{nomeCliente} \to \texttt{nomeCliente}$

Em geral, $\alpha \to \beta$ é trivial se $\beta \subseteq \alpha$.

Dependências Funcionais

Chaves, são dependências funcionais.

- K é uma **super-chave** no esquema R sse $K \to R$.
- K é uma **chave candidata** em R sse $K \to R$, e para nenhum $\alpha \subset K$, $\alpha \to R$.

As dependências funcionais permitem expressar restrições, que não podem ser expressas somente através dos conceitos de chave.
Por exemplo, em `(nomeCliente, numEmprestimo, nomeBalcao, quantia)`.

- Espera-se que as seguintes dependências sejam verdadeiras:

  $\texttt{numEmprestimo} \to \texttt{quantia}$
  $\texttt{numEmprestimo} \to \texttt{nomeBalcao}$
- Mas não se espera que a dependência abaixo seja verdadeira:

  $\texttt{numEmprestimo} \to \texttt{nomeCliente}$

Uso de Dependências Funcionais

Usam-se dependências funcionais para:

- testar (instâncias de) relações, para verificar se “fazem sentido” de acordo com as dependências funcionais.

Definição

Se uma relação r torna verdadeiras todas as dependências dum conjunto F, então diz-se que r ***satisfaz*** *F.*

- Especificar restrições sobre as relações.

Definição

Diz-se que F é verdadeira em R se todas as relações (possíveis) sobre R ***satisfazem as dependências*** *em F.*

Nota: Uma instância particular duma relação pode satisfazer uma dependência funcional mesmo que a dependência não seja verdadeira no esquema. Por exemplo, uma instância particular (em que, por acaso, nenhum empréstimo tenha mais que um cliente) satisfaz: $\texttt{numEmprestimo} \to \texttt{nomeCliente}$.

Fecho de um Conjunto de Dependências Funcionais

Dado um conjunto F de dependências, há outras dependências que são logicamente implicadas por F. Por exemplo, se $A \to B$ e $B \to C$, então, ter-se-á $A \to C$.

Definição (Fecho)

Ao conjunto de todas as dependências funcionais implicadas por F chama-se ***fecho*** *de F (denotado por F^+).*

Podem encontrar-se todas as dependências em F^+ por aplicação dos Axiomas de Armstrong.

Definição (Axiomas de Armstrong)

- *Se $\beta \subseteq \alpha$, então $\alpha \to \beta$* **(reflexividade)**
- *Se $\alpha \to \beta$, então $\gamma\alpha \to \gamma\beta$* **(aumento)**
- *Se $\alpha \to \beta$, e $\beta \to \gamma$, então $\alpha \to \gamma$* **(transitividade)**

Estes regras são:

- **coerentes**, isto é, só geram dependências que pertencem a F^+
- **completas**, isto é, geram todas as dependências pertencentes a F^+

Exemplo

Sejam

$$R = (A, B, C, G, H, I)$$

e

$$F = \{A \to B, A \to C, CG \to H, CG \to I, B \to H\}$$

.

Podemos obter alguns dos elementos de F^+, aplicando os axiomas de Armstrong.

- $A \to H$, por transitividade a partir de $A \to B$ e $B \to H$.
- $AG \to I$, por aumento de $A \to C$ com G, obtendo-se $AG \to CG$, de seguida, por transitividade com $CG \to I$.
- $CG \to HI$, por aumento de $CG \to I$ inferindo $CG \to CGI$, de seguida por aumento de $CG \to H$ inferindo $CGI \to HI$, e depois transitividade.

Construção de F^+

Para calcular o fecho de um conjunto de dependências F podemos aplicar o seguinte algoritmo:

$F^+ = F$
repete
  para cada `uma das dependências funcionais` $f \in F^+$ **faz**
    `aplicar reflexividade e aumento em` f
    `adicionar os resultados a` F^+
  para cada `par de dependências` $f_1, f_2 \in F^+$ **faz**
    se f_1 `e` f_2 `podem combinar-se por transitividade`
    então `adicionar a dependência resultante a` F^+
até que F^+ `não mude mais`

NOTA: Veremos, mais tarde, outro procedimento para esta problema

Fecho de Dependências

Podemos facilitar a construção de F^+ usando mais algumas regras coerentes:

- Se $\alpha \to \beta$ e $\alpha \to \gamma$, então $\alpha \to \beta\gamma$ **(união)**
- Se $\alpha \to \beta\gamma$, então $\alpha \to \beta$ e $\alpha \to \gamma$ **(decomposição)**
- Se $\alpha \to \beta$ e $\gamma\beta \to \delta$, então $\alpha\gamma \to \delta$ **(pseudo-transitividade)**

Todas estas regras podem-se derivar dos Axiomas de Armstrong.

Fecho de um Conjunto de Atributos

Dado um conjunto de atributos α, define-se o fecho de α sobre F.

Definição (Fecho de um Conjunto de Atributos)

Dado um conjunto de dependências funcionais F, e $\alpha \subseteq F$, define-se o fecho de α sobre F, denotado por α^+, como sendo o conjunto de atributos que dependem funcionalmente de α dado F, isto é:

$$\alpha \to \beta \in F^+ \quad \text{sse} \quad \beta \subseteq \alpha^+$$

Algoritmo para calcular α^+.

$\alpha^+ = \alpha$
repete
  para cada $\beta \to \gamma \in F$ **faz**
    se $\beta \subseteq \alpha^+$ **então** $\alpha^+ := \alpha^+ \cup \gamma$
até que α^+ `não mude mais`

Exemplo de fecho de atributos

- $R = (A, B, C, G, H, I)$
- $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$
- cálculo de $(AG)^+$

1. $(AG)^+ := AG$
2. $(AG)^+ := ABCG$ $\quad (A \rightarrow C$ e $A \rightarrow B)$
3. $(AG)^+ := ABCGH$ $\quad (CG \rightarrow H$ e $CG \subseteq AGBC)$
4. $(AG)^+ := ABCGHI$ $\quad (CG \rightarrow I$ e $CG \subseteq AGBCH)$

$(AG)^+$ já não muda mais dado que já inclui todos os atributos de R.

Uso de fecho de atributos

O cálculo do fecho de atributos pode ser usado para vários fins:

- **Testar super-Chaves:** para testar se α é super-chave, calcular α^+, e verificar se α^+ contém todos os atributos de R.
  - Será AG super-chave?
  - E algum subconjunto próprio de AG é super-chave?
- **Testar dependências funcionais:** para verificar se a dependência $\alpha \rightarrow \beta$ é verdadeira (isto é pertence a F^+), basta verificar se $\beta \subseteq \alpha^+$, para um dado α^+.
- **Cálculo do fecho de F:** para cada $\gamma \subseteq R$, calcular γ^+. Para cada $S \subseteq \gamma^+$, devolver como resultado a dependência $\gamma \rightarrow S$.

Cobertura Canónica

- Um conjunto de dependências, podem conter algumas delas que são redundantes (por se inferirem das outras). Por exemplo:
  $A \rightarrow C$ é redundante em: $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$. Porquê?
- Partes de dependências também podem ser redundantes. Por exemplo:
  - $\{A \rightarrow B, B \rightarrow C, A \rightarrow CD\}$ pode ser simplificado para $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$. Porquê?
  - $\{A \rightarrow B, B \rightarrow C, AC \rightarrow D\}$ pode ser simplificado para $\{A \rightarrow B, B \rightarrow C, A \rightarrow D\}$. Porquê?
- Intuitivamente, uma cobertura canónica de F é um conjunto "minimal" de dependências, equivalente a F, e em que nenhuma dependência tem partes redundantes

Atributos dispensáveis

Considere o conjunto de dependências F e a dependência $\alpha \rightarrow \beta \in F$.

Definição (Atributo dispensável à esquerda)

O atributo A é dispensável à esquerda em α se $A \in \alpha$ e F implica $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.

Definição (Atributo dispensável à direita)

O atributo A é dispensável à direita em β se $A \in \beta$, e o conjunto $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ implica F.

Nota: a implicação na direcção oposta é trivial em ambos os casos.

Exemplos:

- Dado $F = \{A \rightarrow C, AB \rightarrow C\}$, B é dispensável em $AB \rightarrow C$ porque $A \rightarrow C$ implica $AB \rightarrow C$.
- Dado $F = \{A \rightarrow C, AB \rightarrow CD\}$, C é dispensável em $AB \rightarrow CD$ pois com $A \rightarrow C$, $AB \rightarrow CD$ pode ser inferido de $AB \rightarrow D$.

Teste para atributos dispensáveis

Considere o conjunto F de dependências, e a dependência $\alpha \to \beta \in F$.

- Para testar se $A \in \alpha$ é dispensável em α, basta:
  1. calcular $(\alpha - A)^+$ usando as dependências em F;
  2. verificar se $(\alpha - A)^+$ contém A. Se contém, então A é dispensável.
- Para testar se $A \in \beta$ é dispensável em β, basta:
  1. calcular α^+ usando as dependências em $F' = (F - \{\alpha \to \beta\}) \cup \{\alpha \to (\beta - A)\}$;
  2. verificar se α^+ contém A. Se contém, então A é dispensável.

Cobertura Canónica

Definição (Cobertura Canónica)

Uma cobertura canónica de F é um conjunto de dependências F_c tal que:

- *F implica todas as dependências em F_c, e*
- *F_c implica todas as dependências em F, e*
- *Nenhuma dependência em F_c contém atributos dispensáveis, e*
- *O lado esquerdo de cada dependência em F_c é único.*

Uma cobertura canónica de F é o conjunto de dependências funcionais com o mesmo poder expressivo que F e mínimo, isto é com o menor número de dependências funcionais possível.

Cálculo da Cobertura Canónica

Para calcular uma cobertura canónica de F:

$F_c := F$ **repete**
  Usar a regra da união para substituir as dependências em F_c,
    $\alpha_1 \to \beta_1$ e $\alpha_1 \to \beta_2$ por $\alpha_1 \to \beta_1\beta_2$
  enquanto há dependências com atributos dispensáveis **faz**
    Encontrar dependências $\alpha \to \beta$ com atributos dispensáveis (em α ou β)
    Quando se encontra atributo dispensável, apaga-se de $\alpha \to \beta$
  fimenquanto
até que F_c não muda.

Nota: A regra da união pode tornar-se aplicável depois de retirados alguns atributos dispensáveis. Por isso há que re-aplicá-la.

Exemplo de cálculo de cobertura canónica

- $R = (A, B, C)$
- $F = \{A \to BC, B \to C, A \to B, AB \to C\}$
- cálculo de F_c:
  1. Combinar $A \to BC$ e $A \to B$ para obter $A \to BC$;
  2. $F_c = \{A \to BC, B \to C, AB \to C\}$;
  3. A é dispensável em $AB \to C$ porque $B \to C$ implica $AB \to C$;
  4. $F_c = \{A \to BC, B \to C\}$;
  5. C é dispensável em $A \to BC$ pois $A \to BC$ é implicado por $A \to B$ e $B \to C$;
  6. $F_c = \{A \to B, B \to C\}$;
  7. Não há mais atributos dispensáveis. Verifica-se também que F_c não muda mais
- A cobertura canónica é: $F_c = \{A \to B, B \to C\}$.

Objectivos com a Concepção de BDs Relacionais

- Pretende-se encontrar "**bons**" conjuntos de esquemas relações, para armazenar os dados.
- Uma "**má**" concepção pode levar a:
  - Repetição de dados;
  - Impossibilidade de representar certos tipos de informação;
  - Dificuldade na verificação da integridade.
- Objectivos da concepção (para atingir um "bom" esquema):
  - Evitar dados redundantes;
  - Garantir que as relações relevantes sobre dados podem ser representadas;
  - Facilitar a verificação de restrições de integridade.

Exemplo

Concepção de um esquema de base de dados, avaliação do mesmo, e sua (se necessário) transformação num "bom" esquema.

- Concepção: Considere o esquema simples: `Amigos = (nome, telef, codPostal, localidade)`. E uma sua instância:

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

- Redundância: os valores de `(codPostal, 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 hajam amigos dessa localidade.
  - Podem usar-se valores nulos, mas estes são difíceis de gerir.

Objectivos da Normalização

Após a concepção (e antes da implementação num dado SGBD), pretende-se obter um "bom" esquema. Temos então que:

- Avaliar: decidir se o um dado esquema R já está num "bom" formato.
- Transformar (normalizar): se não estiver, decompor R num conjunto de esquemas $\{R_1, R_2, \ldots, R_n\}$ tal que:
  - cada um deles está num "bom" formato;
  - a decomposição é sem perdas.
- A **normalização** é baseada em:
  - dependências funcionais;
  - dependências multi-valor.

Exemplo - Decomposição

- Decompor o esquema Amigos em:

```
Amigos1 = (nome,telef, codPostal)
CPs = (codPostal, localidade)
```

Uma qualquer decomposição tem de preservar a informação, contida no esquema inicial.

- **Não pode haver perda de atributos:** todos os atributos do esquema original (R) têm que aparecer na decomposição (R_1, R_2), isto é, $R = R_1 \cup R_2$.
- **Decomposição sem perdas:** para todas as relações possíveis r sobre o esquema R tem de se verificar que:

$$r = \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r)$$

A decomposição de R em R_1 e R_2 é sem perdas sse pelo menos uma das dependências abaixo pertence a F^+:
  - $R_1 \cap R_2 \rightarrow R_1$
  - $R_1 \cap R_2 \rightarrow R_2$

Exemplo de decomposição sem perdas

Decomposição de Amigos em:

```
Amigos1 = (nome,telef,codPostal)
CPs = (codPostal, localidade)
```

r

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

$\Pi_{\text{Amigos1}}(r)$

nome	telef	codPostal
Maria	1111	2815
João	2222	1000
Pedro	1112	1100
Ana	3333	2815

$\Pi_{\text{CPs}}(r)$

codPostal	localidade
2815	Caparica
1000	Lisboa
1100	Lisboa

Verifica-se que:

$$\Pi_{\text{Amigos1}}(r) \bowtie \Pi_{\text{CPS}}(r) = r$$

Notar que é valida a dependência: `codPostal` $\rightarrow$ `localidade`, isto é, verifica-se $R_1 \cap R_2 \rightarrow R_2$.

Exemplo de decomposição com perdas

Decomposição de Amigos em:

```
Amigos2 = (nome,telef,localidade)
Loc = (localidade,codPostal).
```

r

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

$\neq$

$\Pi_{\text{Amigos2}}(r) \bowtie \Pi_{\text{Loc}}(r)$

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

$\Pi_{\text{Amigos2}}(r)$

nome	telef	localidade
Maria	1111	Caparica
João	2222	Lisboa
Pedro	1112	Lisboa
Ana	3333	Caparica

$\Pi_{\text{CPs}}(r)$

localidade	codPostal
Caparica	2815
Lisboa	1000
Lisboa	1100

Note-se que nenhuma das duas dependências seguintes é válida:

- `localidade` $\rightarrow$ `nome, telefone`, isto é, $R_1 \cap R_2 \not\rightarrow R_1$.
- `localidade` $\rightarrow$ `codPostal`, isto é, $R_1 \cap R_2 \not\rightarrow R_2$.

Normalização por uso de Dependências

Quando se decompõe um esquema R com dependências F, em $R_1, R_2, \ldots, R_n$ quer-se:

- **Decomposição sem perdas**. Por forma a não se perder informação.
- **Não haja redundância**. Ver-se-à mais à frente como ...
- **Preservação de dependências**. Por forma a que verificação das dependências possa ser feita de forma eficiente.

Seja F_i o conjunto de dependências de F^+ que só contêm atributos de R_i.
A decomposição preserva as dependências se

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

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.

Exemplo

- Sejam $R = (A, B, C)$ e $F = \{A \rightarrow B, B \rightarrow C\}$.
- Decomposição 1: $R_1 = (A, B)$, $R_2 = (B, C)$:
  - Decomposição sem perdas: $R_1 \cap R_2 = \{B\}$e $B \rightarrow BC$;
  - Preserva as dependências.
- Decomposição 2: $R_1 = (A, B)$, $R_2 = (A, C)$:
  - Decomposição sem perdas: $R_1 \cap R_2 = \{A\}$ e $A \rightarrow AB$;
  - Não preserva as dependências. Não se pode verificar $B \rightarrow C$ sem calcular $R_1 \bowtie R_2$.