# Lezione 14.Decomposizioni che preservano le dipendenze - Esercizi

Prof.ssa Maria De Marsico
demarsico@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Parte del materiale è tratto dal materiale preparato dalla prof.ssa Moscarini per il canale A-L

# Definizione!

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- **Definizione** Sia ***R*** **uno schema di relazione,** ***F*** **un insieme di dipendenze funzionali su** ***R*** e $\rho=\{R_1, R_2,..., R_k\}$ una **decomposizione** di *R*.
- Diciamo che ***ρ preserva F*** se $F\equiv\cup_{i=1}^{k} \pi_{Ri}(F)$,
- dove $\pi_{Ri}(F) = \{X\rightarrow Y / X\rightarrow Y \in F^{+} \wedge XY\subseteq R_i\}$.

# Verifica

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Supponiamo di **avere già** una **decomposizione** e di voler **verificare se** preserva le dipendenze funzionali.

- Verificare **se** una decomposizione preserva un insieme di dipendenze funzionali *F* richiede che venga verificata **l'equivalenza** dei due insiemi di dipendenze funzionali ***F*** e $G=\cup_{i=1}^{k} \pi_{R_i}(F)$ *e quindi la doppia inclusione* $F^+ \subseteq G^+$ e che $F^+ \supseteq G^+$.

- **Nota: per come è stato definito G in questo caso**, sarà sicuramente $F^+ \supseteq G$
- Infatti $G=\cup_{i=1}^{k} \pi_{R_i}(F)$, dove $\pi_{R_i}(F) = \{X \rightarrow Y / X \rightarrow Y \in F^+ \wedge XY \subseteq R_i\}$
- *Ogni* ***proiezione di F*** *che viene* ***inclusa per definizione in G*** *è un* ***sottoinsieme di F⁺****, quindi* $F^+$ ***contiene*** *G* (che **ovviamente** può anche essere scritto come $G \subseteq F^+$) e per il lemma sulle chiusure questo implica che $G^+ \subseteq F^+$ (che **ovviamente** può anche essere scritto come $F^+ \supseteq G^+$)

- Per il Lemma sulle chiusure è sufficiente quindi verificare che $F \subseteq G^+$ (che poi implica $F^+ \subseteq G^+$ )

# Verifica

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- La verifica che $F \subseteq G^+$ (che poi implica che $F^+ \subseteq G^+$ )
- può essere fatta con l'algoritmo che segue (la cui correttezza è una banale conseguenza del Lemma sulla chiusura di un insieme di attributi e del Teorema sull'uguaglianza $F^+=F^A$).

**Algoritmo - contenimento di F in $G^+$**

- **Input** due insiemi $F$ e $G$ di dipendenze funzionali su $R$;
- **Output** la variabile *successo* (al termine avrà valore ***true*** se $F \subseteq G^+$ , ***false*** altrimenti)

**begin**

*successo:=true;*

**for every** $X \rightarrow Y \in F$

**do** **begin**

*calcola* $X^+_G$;

**if** $Y \not\subset X^+_G$ **then** *successo:=false*

**end**

**end**

**Lemma** Siano $R$ uno schema di relazione ed $F$ un insieme di dipendenze funzionali su $R$. Si ha che: $X \rightarrow Y \in F^A$ se e solo se $Y \subseteq X^+$.

**Teorema** Siano $R$ uno schema di relazione ed $F$ un insieme di dipendenze funzionali su $R$. Si ha $F^+ = F^A$.

- **Se** $Y \not\subset X^+_G$ allora $X \rightarrow Y \notin G^A$ per il lemma e quindi $X \rightarrow Y \notin G^+$ per il Teorema
- Basta verificare che **anche una sola dipendenza** non appartiene alla chiusura di G per poter affermare che **l'equivalenza non sussiste**

# Problema



- **Come calcoliamo $X^+_G$?**
- Se volessimo utilizzare l'Algoritmo già visto per il calcolo della chiusura di un insieme di attributi, dovremmo **prima calcolare $G$ …**
- **…** ma, **per la definizione di $G$**, ciò **richiede** il calcolo di **$F^+$** che richiede tempo **esponenziale**.
- Presentiamo un algoritmo che permette di calcolare $X^+_G$ a **partire da $F$.**

# Algoritmo calcolo $X^+_G$ a partire da $F$.



**Algoritmo -** $X^+_G$ a **partire da** $F$**.**

- **Input** uno schema $R$, un insieme $F$ di dipendenze funzionali su $R$, una decomposizione $\rho=\{R_1, R_2,..., R_k\}$ di $R$, un sottoinsieme $X$ di $R$;
- **Output** la chiusura di $X$ rispetto a $G=\cup_{i=1}^{k} \pi_{R_i}(F)$, (**nella variabile** $Z$);

**begin**

$Z:=X;$

$S:=\varnothing;$

**for** $i:=1$ **to** $k$

**do** $S:=S\cup(Z\cap R_i)^+_F\cap R_i$

**while** $S\not\subset Z$

**do**

**begin**

$Z:=Z\cup S;$

**for** $i:=1$ **to** $k$

**do** $S:=S\cup(Z\cap R_i)^+_F\cap R_i$

**end**

**end**

**Attenzione!!!**
L' intersezione ha **priorità maggiore** dell'unione, quindi $Z\cap R_i)^+_F\cap R_i$ va calcolato **prima dell'unione con S. Se si inverte l'ordine delle operazioni** da S potremmo eliminare ciò che non rientra in $R_i$ (perché è stato inserito in un passaggio precedente grazie alla proiezione su un sottoschema senza intersezioni con $R_i$) ma questo non avrebbe senso perché in S stiamo accumulando gli attributi che sono determinati funzionalmente da X anche se appartengono a sottoschemi diversi

# Attenzione!

- L'algoritmo (per definizione di algoritmo …) **termina sempre**!

- Il fatto che l'algoritmo termini **non indica** che una dipendenza $X \rightarrow Y$ **è preservata!**

- Per **verificare se** $X \rightarrow Y$ è preservata, in base al Lemma sulla chiusura di un insieme di attributi e in base al Teorema sull'uguaglianza $F^+ = F^A$ , **dobbiamo controllare <u>SE</u>** ***Y*** **è contenuto nella copia finale della variabile** ***Z*** (che conterrà la **chiusura di X rispetto a G,** $X^+_G$ )

# Esempio 1



Dato il seguente schema di relazione

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

e il seguente insieme di dipendenze funzionali

$F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$

dire se la decomposizione $\rho = \{ ABC, ABD \}$

preserva le dipendenze in F

In base a quanto visto basta verificare che $F \subseteq G^+$ cioè che **ogni** dipendenza funzionale in F si trova in $G^+$

**NOTA IMPORTANTE** - In effetti è <u>inutile</u> controllare che vengano preservate le dipendenze tali che **l'unione delle parti destra e sinistra è contenuta interamente in un sottoschema**, perché secondo la **definizione**
$\pi_{R_i}(F) = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq R_i\}$.
**tali dipendenze fanno parte per definizione di G**.

**NOTA**: Per come è strutturato l'algoritmo, a Z possono **solo** venire **aggiunti** elementi (cioè non succede mai che un attributo venga eliminato da Z), quindi quando Z arriva a contenere **la parte destra della dipendenza** possiamo essere sicuri che la dipendenza stessa è preservata e sospendere il seguito del procedimento (in un compito scritto questo fatto va giustificato).

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$ $\rho = \{ ABC, ABD \}$

Menzionando esplicitamente l'osservazione riportata nella prima nota, basta verificare che sia preservata la dipendenza $D \rightarrow C$ .

$Z = D$

$S = \varnothing$

ciclo for esterno sui sottoschemi ABC e ABD

$S = S \cup (D \cap ABC)^+_F \cap ABC = \varnothing \cup (\varnothing)^+_F \cap ABC = \varnothing \cup \varnothing \cap ABC = \varnothing$

$S = S \cup (D \cap ABD)^+_F \cap ABD = S \cup (D)^+_F \cap ABD$ ... (continua)

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo $(D)^+_F = DCBA$

... (continua) $S = S \cup (D)^+_F \cap ABD = \varnothing \cup DCBA \cap ABD = ABD$

Attenzione!!!
Ovviamente $(\varnothing)^+_F = \varnothing$ **qualunque** sia F!!!

**prima** l'intersezione!

$\varnothing \cap X = \varnothing$
**qualunque** sia X!!!

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D) F = { AB→ C, D→ C, D→ B, C→ B, D→ A } ρ = { ABC,ABD }

ABD $\not\subset$ D (S $\not\subset$ Z) quindi **entriamo nel ciclo while**

Z = Z∪ S = ABD

prima l' intersezione!

ciclo for interno al while sui sottoschemi ABC e ABD

$S = S \cup (ABD \cap ABC)^+_F \cap ABC = S \cup (AB)^+_F \cap ABC = ABD \cup ABC \cap ABC = ABCD$

$S = S \cup (ABD \cap ABD)^+_F \cap ABD = S \cup (ABD)^+_F \cap ABD = ABCD \cup ABCD \cap ABD = ABCD \cup ABD = ABCD$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo che $(AB)^+_F = ABC$ e $(ABD)^+_F = ABCD$

ABCD $\not\subset$ ABD quindi rientriamo nel ciclo while

Z = Z∪ S = ABCD

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D) F = { AB→ C, D→ C, D→ B, C→ B, D→ A } ρ = { ABC,ABD }

ABCD $\not\subset$ ABD quindi rientriamo nel ciclo while

Z = Z$\cup$ S = ABCD

ciclo for interno al while sui sottoschemi ABC e ABD

S = S $\cup$ (ABCD $\cap$ ABC)$^+_F \cap$ ABC = S $\cup$ (ABC)$^+_F \cap$ ABC = ABCD $\cup$ ABC $\cap$ ABC = ABCD

S = S $\cup$ (ABCD $\cap$ ABD)$^+_F \cap$ ABD = S $\cup$ (ABD)$^+_F \cap$ ABD = ABCD $\cup$ ABCD $\cap$ ABC = ABCD $\cup$ ABC = ABCD

**S $\subset$ Z quindi STOP**

l'algoritmo si ferma, ma **va controllato il contenuto di Z**

**Z= (D)$^+_G$ = ABCD   C $\in$ (D)$^+_G$ quindi la dipendenza è preservata**

Poichè **(D)$^+_G$ = ABCD**
osserviamo che sono preservate anche **D → B** e **D → A**
**Che però per le note precedenti sapevamo comunque già essere in G**

# Esempio 1: svolgimento



- In base alle osservazioni sulle dipendenze sicuramente contenute in G e al fatto di aver verificato che D→ C è preservata (era l'unica in dubbio), possiamo già dire che **la decomposizione preserva le dipendenze**

- **A scopo didattico verifichiamo anche le altre**

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$ $\rho = \{ ABC, ABD \}$

Cominciamo con $AB \rightarrow C$

$Z = AB$

$S = \varnothing$

ciclo for esterno sui sottoschemi ABC e ABD

$S = S \cup (AB \cap ABC)^+_F \cap ABC = S \cup (AB)^+_F \cap ABC$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo $(AB)^+_F = ABC$

$S = S \cup (AB)^+_F \cap ABC = \varnothing \cup ABC \cap ABC = ABC$

$S = S \cup (AB \cap ABD)^+_F \cap ABD = ABC \cup (AB)^+_F \cap ABD = ABC \cup ABC \cap ABD = ABC$

# Esempio 1: svolgimento



$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$ $\rho = \{ ABC, ABD \}$

$ABC \not\subset AB$ quindi entriamo nel ciclo while

$Z = Z \cup S = ABC$

Secondo la nota precedente (a Z possono solo venire aggiunti elementi),**potremmo interrompere l'algoritmo** perché **$C \in Z \subset (AB)^+_G$.**

A scopo didattico continuiamo l'esecuzione. In un esercizio di esame ci si può fermare **fornendo la giusta motivazione**

ciclo for interno al while sui sottoschemi ABC e ABD

$S = S \cup (ABC \cap ABC)^+_F \cap ABC = S \cup (ABC)^+_F \cap ABC$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo $(ABC)^+_F = ABC$

$S = S \cup (ABC)^+_F \cap ABC = ABC \cup ABC \cap ABC = ABC$

$S = S \cup (ABC \cap ABD)^+_F \cap ABD = S \cup (AB)^+_F \cap ABD = ABC \cup ABC \cap ABD = ABC \cup AB = ABC$

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- S $\subset$ Z quindi STOP senza rientrare nel while

l' algoritmo si ferma, ma **va controllato il contenuto di Z**

- **Z= $(AB)^+_G$ = ABC** **C $\in (AB)^+_G$ quindi** la dipendenza è preservata

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$ $\rho = \{ ABC, ABD \}$

Infine verifichiamo che venga preservata $C \rightarrow B$

$Z = C$

$S = \varnothing$

ciclo for esterno sui sottochemi ABC e ABD

$S = S \cup (C \cap ABC)^+_F \cap ABC = S \cup (C)^+_F \cap ABC$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo $(C)^+_F = BC$

$S = S \cup (C)^+_F \cap ABC = \varnothing \cup BC \cap ABC = \varnothing \cup BC = BC$

$S = S \cup (C \cap ABD)^+_F \cap ABD = S \cup (\varnothing)^+_F \cap ABD = BC \cup \varnothing \cap ABD = BC \cup \varnothing = BC$

# Esempio 1: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D)$ $F = \{ AB \rightarrow C, D \rightarrow C, D \rightarrow B, C \rightarrow B, D \rightarrow A \}$ $\rho = \{ ABC, ABD \}$

$BC \not\subset C$ quindi **entriamo nel ciclo while**

$Z = Z \cup S = BC$

ciclo for interno al while sui sottoschemi ABC e ABD

$S = S \cup (BC \cap ABC)^+_F \cap ABC = S \cup (BC)^+_F \cap ABC$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo
$(BC)^+_F = BC$

$S = S \cup (BC)^+_F \cap ABC = BC \cup BC \cap ABC = BC$

$S = S \cup (BC \cap ABD)^+_F \cap ABD = S \cup (B)^+_F \cap ABD$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo
$(B)^+_F = B$

$S = S \cup (B)^+_F \cap ABD = BC \cup B \cap ABD = BC$

**$S \subset Z$ quindi STOP** l'algoritmo si ferma, ma **va controllato il contenuto di Z**

$Z = (C)^+_G = BC$ cioè $B \in (C)^+_G$ quindi la dipendenza è preservata.

# Esempio 2

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Dato il seguente schema di relazione

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

e il seguente insieme di dipendenze funzionali

$$F = \{ AB \rightarrow E,\ B \rightarrow CE,\ ED \rightarrow C \}$$

dire se la decomposizione $\rho = \{ ABE, CDE \}$

preserva le dipendenze in F

In base a quanto visto basta verificare che $F \subseteq G^+$ cioè che **ogni** dipendenza funzionale in F si trova in $G^+$

**NOTA** – come abbiamo verificato, è <u>inutile</u> controllare che vengano preservate le dipendenze tali che **l'unione delle parti destra e sinistra è contenuta interamente in un sottoschema**, perché secondo la **definizione**
$\pi_{R_i}(F) = \{X \rightarrow Y \mid X \rightarrow Y \in F^+ \wedge XY \subseteq R_i\}$.
**tali dipendenze fanno parte per definizione di G**.
In questo esempio, questo vale per $AB \rightarrow E$ e per $ED \rightarrow C$
Quindi verifichiamo solo che venga preservata la dipendenza **B→ CE**

# Esempio 2: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R = (A, B, C, D, E)$ $F = \{ AB \rightarrow E, B \rightarrow CE, ED \rightarrow C\}$ $\rho = \{ ABE, CDE \}$

Attenzione!!!
$(\varnothing)^+_F = \varnothing$
**qualunque** sia F!!!
$\varnothing \cap X = \varnothing$
**qualunque** sia X!!!

Verifichiamo che sia preservata $B \rightarrow C E$.

$Z = B$

$S = \varnothing$

ciclo for esterno sui sottoschemi ABE e CDE

$S = S \cup (B \cap ABE)^+_F \cap ABE = \varnothing \cup (B)^+_F \cap ABE = \varnothing \cup BCE \cap ABE = BE$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo $(B)^+_F = BCE$

$S = BE \cup (B \cap CDE)^+_F \cap CDE = BE \cup (\varnothing)^+_F \cap CDE = BE$

**prima** l'intersezione! … elimineremmo … la B!

$BE \not\subset B$ ($S \not\subset Z$) quindi **entriamo nel ciclo while**

# Esempio 2: svolgimento

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

R = (A, B, C, D, E)    F = { AB→ E, B→ CE, ED→ C}    ρ = { ABE, CDE }

BE $\not\subset$ B (S $\not\subset$ Z) quindi **entriamo nel ciclo while**

Z = Z∪ S = B ∪ BE = BE

prima l' intersezione!

ciclo for interno al while sui sottoschemi ABE e CDE

$S = BE \cup (BE \cap ABE)^+_F \cap ABE = BE \cup (BE)^+_F \cap ABE = BE \cup BCE \cap ABE = BE$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo che $(BE)^+_F = BCE$

$S = BE \cup (BE \cap CDE)^+_F \cap CDE = S \cup (E)^+_F \cap CDE = BE \cup E \cap CDE = BE \cup E = BE$

Applicando l'algoritmo sulla chiusura di un insieme di attributi abbiamo che $(E)^+_F = E$

**BE = BE (S $\subseteq$Z ) quindi STOP**

**Z= $(B)^+_G$ = BE**

l'algoritmo si ferma, ma **controlliamo Z**

$E \in (B)^+_G$ MA    $C \notin (B)^+_G$

**quindi la dipendenza B→ CE non è preservata (nella chiusura manca uno degli attributi che dovrebbero essere determinati funzionalmente da B)**

# Esempi

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Torniamo alle decomposizioni degli esempi, e verifichiamo se l'algoritmo avrebbe rilevato la perdita di alcune dipendenze funzionali.

# Esempi iniziali

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$R=ABC$ con l' insieme di dipendenze funzionali $F=\{A\rightarrow B, B\rightarrow C\}$

Decomponiamo $R$ in $\rho$ = { AB, AC }

Cominciamo a verificare se $\rho$ preserva $A\rightarrow B$

Z = A

S = $\varnothing$

ciclo for esterno sui sottoschemi AB e AC

$S = S \cup (A \cap AB)^+_F \cap AB = S \cup (A)^+_F \cap AB$

Applicando l'algoritmo sulla chiusura di un insieme di attributi

$(A)^+_F = ABC$

$S = S \cup (A)^+_F \cap AB = \varnothing \cup ABC \cap AB = \varnothing \cup AB = AB$

$S = S \cup (A \cap AC)^+_F \cap AC = S \cup (A)^+_F \cap AC = AB \cup ABC \cap AC = AB \cup AC = ABC$

$ABC \not\subset A$ dovremmo continuare ma Z contiene già tutto R e non possiamo togliere attributi, quindi ci fermiamo perché sicuramente $\mathbf{(A)^+_G = R}$

e quindi $\mathbf{B \in (A)^+_G}$

# Esempi iniziali

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

*R=ABC* *F={A→B, B→C}* $\rho$ = { AB, AC }

*Verifichiamo B→C*

Z = B

S = $\emptyset$

ciclo for esterno sui sottochemi AB e AC

$S = S \cup (B \cap AB)^+_F \cap AB = S \cup (B)^+_F \cap AB$

Applicando l'algoritmo sulla chiusura di un insieme di attributi

$(B)^+_F = BC$

$S = S \cup (B)^+_F \cap AB = \emptyset \cup BC \cap AB = \emptyset \cup B = B$

$S = S \cup (B \cap AC)^+_F \cap AC = S \cup (\emptyset)^+_F \cap AC = B \cup \emptyset \cap AC = B \cup \emptyset = B$

$B \subseteq B$ STOP

**$Z=(B)^+_G=B$**

l'algoritmo si ferma, ma **va controllato il contenuto di Z**

e quindi **$C \notin (B)^+_G$**

**La dipendenza *B→C* non è preservata**

**L'algoritmo conferma che la decomposizione $\rho$ non preserva F, come avevamo già scoperto dall'esempio**

# Esempi iniziali

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Consideriamo lo schema *R=(Matricola, Comune, Provincia)* con l'insieme di dipendenze funzionali
  $F=\{Matricola \rightarrow Comune,\ Comune \rightarrow Provincia\}$
- Decomponiamo *R in R1* $\rho$ = *{(Matricola, Comune), (Matricola, Provincia)}*
- Cominciamo a verificare se $\rho$ preserva *Matricola* $\rightarrow$ *Comune* … ma prima di iniziare sostituiamo i nomi lunghi con lettere più «comode» … Magari A, B, C … ma allora torniamo all'esempio di prima …

- $R=ABC$ con l'insieme di dipendenze funzionali $F=\{A\rightarrow B,\ B\rightarrow C\}$
- Decomponiamo *R* in $\rho$ = { AB, AC }

- Abbiamo già verificato che *Matricola* $\rightarrow$ *Comune* $(A\rightarrow B)$ viene preservata, ma *Comune* $\rightarrow$ *Provincia* no … quindi la decomposizione $\rho$ = *{(Matricola, Comune), (Matricola, Provincia)}* **non preserva *F***

# Finito?



- Abbiamo sottolineato varie volte che una buona decomposizione deve soddisfare 3 condizioni
  - ogni sottoschema deve essere in 3NF
  - la decomposizione deve preservare tutte le dipendenze in F+
  - la decomposizione deve permettere di ricostruire una istanza legale decomposta senza perdita di informazione (join senza perdita)

- Abbiamo visto come, data una decomposizione, possiamo verificare che preservi le dipendenze.

- Vedremo come, data una decomposizione, possiamo verificare che abbia un join senza perdita.