

# Progettazione logica: normalizzazione di schemi relazionali

Sistemi Informativi T

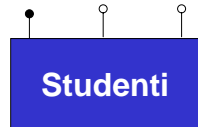
Versione elettronica: [08.3.progLogica.normalizzazione.pdf](#)

## Forme normali

- Una forma normale è una proprietà di uno schema relazionale che ne garantisce la “**qualità**”, cioè l’**assenza di determinati difetti**
- Una **relazione non normalizzata**:
  - **presenta ridondanze**,
  - esibisce **comportamenti poco desiderabili durante gli aggiornamenti**
- Storicamente, le forme normali sono state definite per il modello relazionale, ma hanno senso anche in altri contesti, ad esempio nel modello E/R
- L’attività che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale è detta **normalizzazione**
- La normalizzazione va utilizzata come **tecnica di verifica dei risultati della progettazione** di una base di dati
  - Non costituisce quindi una metodologia di progettazione
- E anche per analizzare la bontà di un DB preesistente

## Una relazione con anomalie

matricola cognome nome



Matricola	Cognome	Nome	Fac_nome	Fac_ind
29323	Bianchi	Giorgio	Ingegneria	Risorgimento 2
35467	Rossi	Anna	Medicina	Massarenti 9
39654	Verdi	Marco	Ingegneria	Risorgimento 2
42132	Neri	Lucia	Agraria	Fanin 50
11274	Gialli	Luca	Agraria	Fanin 50

- L'indirizzo di una facoltà è ripetuto in tutte le tuple dei suoi studenti: **ridondanza**
- Se l'indirizzo di una facoltà cambia, è necessario modificare il valore in diverse tuple: **anomalia di aggiornamento**
- Una nuova facoltà senza studenti non può essere inserita: **anomalia di inserimento**
- ...

Prog. Logica: normalizzazione

Sistemi Informativi T

3

## Un'altra relazione con anomalie

- Lo stipendio di ciascun impiegato è ripetuto in tutte le tuple relative: **ridondanza**
- Se lo stipendio di un impiegato varia, è necessario modificare il valore in diverse tuple: **anomalia di aggiornamento**
- Se un impiegato interrompe la partecipazione a tutti i progetti, dobbiamo cancellarlo: **anomalia di cancellazione**
- Un nuovo impiegato senza progetto non può essere inserito: **anomalia di inserimento**

Impiegato	Stipendio	Progetto	Bilancio	Funzione
Rossi	20	Marte	2	Tecnico
Verdi	35	Giove	15	Progettista
Verdi	35	Venere	15	Progettista
Neri	55	Venere	15	Direttore
Neri	55	Giove	15	Consulente
Neri	55	Marte	2	Consulente
Mori	48	Marte	2	Direttore
Mori	48	Venere	15	Progettista
Bianchi	48	Venere	15	Progettista
Bianchi	48	Giove	15	Direttore

Prog. Logica: t

4

## Analizziamo la relazione...

- Ogni impiegato ha un solo stipendio (anche se partecipa a più progetti)
- Ogni progetto ha un (solo) bilancio
- Ogni impiegato in ciascun progetto ha una sola funzione (anche se può avere funzioni diverse in progetti diversi)
- Ma abbiamo usato un'unica relazione per rappresentare tutte queste informazioni eterogenee:
  - gli impiegati con i relativi stipendi
  - i progetti con i relativi bilanci
  - le partecipazioni degli impiegati ai progetti con le relative funzioni
- E' evidente che la relazione è progettata male
- Quello che vogliamo fare ora è astrarre dal caso specifico e cercare di trovare la ragione più generale che dà luogo alle anomalie viste...

Prog. Logica: normalizzazione

Sistemi Informativi T

5

## Dipendenza funzionale

- Per formalizzare i problemi visti si introduce un nuovo tipo di vincolo, la **dipendenza funzionale**
- Dato uno schema  $R(XYZ)$ , diciamo che in  $R$  vale la **dipendenza funzionale (FD)  $X \rightarrow Y$**  ( $X$  determina funzionalmente  $Y$ ) se e solo se
  - in ogni istanza ammissibile  $r$  di  $R(X)$  non esistono due tuple distinte  $t1$  e  $t2$  tali che  $t1[X] = t2[X]$  e  $t1[Y] \neq t2[Y]$
- Ovvero: se  $t1$  e  $t2$  hanno gli stessi valori su  $X$ , allora hanno gli stessi valori anche su  $Y$

Prog. Logica: normalizzazione

Sistemi Informativi T

6

## Esempi di FD

- Nella relazione vista si hanno diverse FD, tra cui:

Impiegato  $\rightarrow$  Stipendio

Progetto  $\rightarrow$  Bilancio

Impiegato, Progetto  $\rightarrow$  Funzione

- Altre FD sono meno “interessanti” (“banali”), perché sempre soddisfatte, ad esempio:

Impiegato, Progetto  $\rightarrow$  Progetto

- $Y \rightarrow A$  è non banale se A non appartiene a Y

## Anomalie e FD

- Le anomalie viste si riconducono alla presenza delle FD:

Impiegato  $\rightarrow$  Stipendio

Progetto  $\rightarrow$  Bilancio

- Viceversa la FD

Impiegato, Progetto  $\rightarrow$  Funzione

non causa problemi

- Motivo:

- La terza FD ha **sulla sinistra una (super)chiave e non causa anomalie**
- Le prime due FD **non hanno sulla sinistra una chiave e causano anomalie**

- La relazione contiene alcune informazioni legate alla chiave e altre ad attributi che non formano una chiave

## FD e vincoli di chiave

- Le FD sono una generalizzazione dei vincoli di chiave
- Infatti, se si ha uno schema  $R(X)$  con  $X = \underline{K}Y$  allora vale la FD
$$K \rightarrow Y$$
- Siccome è anche vero che  $K \rightarrow K$  si può concludere che
$$K \rightarrow X$$
- In altri termini, **una chiave determina funzionalmente tutti gli attributi dello schema**
  - Questo vale ovviamente anche per le superchiavi ("in ogni istanza ammissibile  $r$  di  $R(X)$  non esistono due tuple distinte  $t_1$  e  $t_2$  tali che  $t_1[K] = t_2[K]$ ")
- E' vero anche il contrario: se un insieme di attributi  $K$  determina funzionalmente tutti gli attributi dello schema, allora  $K$  è superchiave

## Forma normale di Boyce e Codd (BCNF)

- Per evitare le anomalie viste si introduce la

### Forma Normale di Boyce-Codd (BCNF)

Uno schema  $R(X)$  è in forma normale di Boyce e Codd se, per ogni dipendenza funzionale (non banale)  $Y \rightarrow Z$  definita su di esso,  $Y$  è una superchiave di  $R(X)$

- Si noti che, come al solito, il vincolo si riferisce allo schema, in quanto dipende dalla semantica degli attributi
- Un'istanza può pertanto soddisfare "per caso" il vincolo, ma ciò non garantisce che lo schema sia normalizzato
- In altri termini, **le FD non si "ricavano" dall'analisi dei dati, ma ragionando sugli attributi dello schema**

## Normalizzazione in BCNF

- Se uno schema non è in BCNF, la soluzione è **decomporlo**
- L'intuizione è che si devono "estrarre" gli attributi che sono determinati da attributi non chiave ovvero  
**"creare uno schema per ogni funzione"**

Impiegato → Stipendio

Impiegato	Stipendio
Rossi	20
Verdi	35
Neri	55
Mori	48
Bianchi	48

Impiegato, Progetto → Funzione

Impiegato	Progetto	Funzione
Rossi	Marte	Tecnico
Verdi	Giove	Progettista
Verdi	Venere	Progettista
Neri	Venere	Direttore
Neri	Giove	Consulente
Neri	Marte	Consulente
Mori	Marte	Direttore
Mori	Venere	Progettista
Bianchi	Venere	Progettista
Bianchi	Giove	Direttore

Progetto → Bilancio

Progetto	Bilancio
Marte	2
Giove	15
Venere	15

Prog. Logica: normalizzazione

Sistemi Informativi T

11

## Attenzione!

- La soluzione non è tuttavia sempre così semplice, bisogna fare anche altre considerazioni; ad esempio, operando come prima:

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano



Progetto	Sede
Marte	Roma
Giove	Milano
Venere	Milano
Saturno	Milano

Impiegato → Sede

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Progetto → Sede

Impiegato → Sede  
Progetto → Sede

...se proviamo a tornare indietro  
(Join su Sede):



Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano
Verdi	Saturno	Milano
Neri	Giove	Milano

**Diversa dalla relazione di partenza!**

Prog. Logica: normalizzazione

Sistemi Informativi T

12

## Decomposizione senza perdita

- La decomposizione non deve assolutamente alterare il contenuto informativo del DB
- Si introduce pertanto il seguente requisito

### Decomposizione senza perdita (*lossless*)

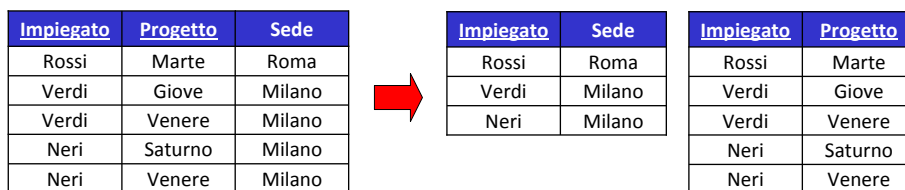
Uno schema  $R(X)$  si decompone senza perdita negli schemi  $R_1(X_1)$  e  $R_2(X_2)$  se, per ogni istanza legale  $r$  su  $R(X)$ , il join naturale delle proiezioni di  $r$  su  $X_1$  e  $X_2$  è uguale a  $r$  stessa:

$$\pi_{X_1}(r) \bowtie \pi_{X_2}(r) = r$$

- Una decomposizione con perdita può generare tuple spurie
- Per decomporre senza perdita è necessario e sufficiente che **il join naturale sia eseguito su una superchiave di uno dei due sottoschemi**, ovvero che valga  $X_1 \cap X_2 \rightarrow X_1$  oppure  $X_1 \cap X_2 \rightarrow X_2$

## Esempio di decomposizione lossless

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano



Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

OK!

... ma i problemi non sono ancora finiti...

## Modifichiamo il DB...

- Supponiamo di voler inserire l'informazione che Neri lavora al progetto Marte:

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere
Neri	Marte

- Ricostruendo la relazione otteniamo:

che però viola la FD **Progetto → Sede**!

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano
Neri	Marte	Milano

## Preservazione delle dipendenze

- Diciamo che una decomposizione preserva le dipendenze se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti
  - Nell'esempio Progetto → Sede non è conservata
- Se una FD non si preserva diventa più complicato capire quali sono le modifiche del DB che non violano la FD stessa
- In generale si devono prima eseguire query SQL di verifica (o, meglio, fare uso di trigger)



## Esempio di query di verifica

- Bisogna verificare che il progetto (**Marte**) sia presso la stessa sede dell'impiegato (**Neri**). A tal fine bisogna trovare un impiegato che lavora al progetto Marte

**Impiegati**

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

**ImpProg**

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere
Neri	Marte

```
SELECT *      -- OK se restituisce una tupla
FROM   Impiegati I
WHERE  I.Impiegato = 'Neri'
      AND I.Sede IN ( SELECT Il.Sede
                      FROM   Impiegati Il, ImpProg IP
                      WHERE  Il.Impiegato = IP.Impiegato
                      AND   IP.Progetto = 'Marte')
```

Prog. Logica: normalizzazione

Sistemi Informativi T

17

## Qualità delle decomposizioni

- Una decomposizione:
  - **deve** essere senza perdita, per garantire la ricostruzione delle informazioni originarie
  - **dovrebbe** preservare le dipendenze, per semplificare il mantenimento dei vincoli di integrità originari

- Nell'esempio, questo suggerisce di inserire anche:

**Progetti**

Progetto	Sede
Marte	Roma
Giove	Milano
Venere	Milano
Saturno	Milano

- Va sempre eseguita una query, ma più semplice

```
SELECT *      -- OK se restituisce una tupla
FROM   Impiegati I, Progetti P
WHERE  I.Impiegato = 'Neri'
      AND P.Progetto = 'Marte'
      AND I.Sede = P.Sede
```

Prog. Logica: normalizzazione

Sistemi Informativi T

18

## Una limitazione non superabile

- In funzione del pattern di FD, può non essere possibile decomporre in BCNF e preservare le FD

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

**Progetto, Sede → Dirigente**  
**Dirigente → Sede**

- **Progetto, Sede → Dirigente** coinvolge **tutti gli attributi** e quindi nessuna decomposizione può preservare tale dipendenza!

## Algoritmi per la normalizzazione

- Quando si hanno diverse FD è difficile ragionare sullo schema, ed è quindi altrettanto difficile operare manualmente buone decomposizioni
- Esiste un algoritmo per normalizzare in BCNF, ma tale algoritmo ha elevata complessità e genera più schemi del necessario
- Viceversa, se si cambia “leggermente” il requisito di BCNF si ha a disposizione un algoritmo efficiente e che risolve la maggior parte dei casi che si presentano nella pratica...
- La **terza forma normale (3NF)**, discussa nel seguito, è un target di normalizzazione che consente di ottenere automaticamente:
  - decomposizioni senza perdita
  - decomposizioni che preservano tutte le dipendenze

## In pratica...

- Se la relazione non è normalizzata si decompone in 3NF
  - Come? Con l'algoritmo descritto nel seguito
- Si verifica se lo schema ottenuto è anche in BCNF
  - Se una relazione ha una sola chiave le due forme normali coincidono
- Se uno schema non è in BCNF si hanno 3 alternative:
  - 1) Si lascia così com'è, gestendo le anomalie residue  
(se l'applicazione lo consente)
  - 2) Si decompone in BCNF, predisponendo trigger o query di verifica
  - 3) Si cerca di rimodellare la situazione iniziale, al fine di permettere di ottenere schemi in BCNF

## 2) Decomposizione dello schema

- È innanzitutto opportuno osservare che {Progetto, Dirigente} è una chiave

- La decomposizione:

non va bene, perché è con perdita!

ProgSedi

Progetto	Sede
Marte	Roma
Giove	Milano
Marte	Milano
Saturno	Milano
Venere	Milano

Dirigenti

Dirigente	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

ProgDir

Dirigente	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Marte
Neri	Saturno
Neri	Venere

Dirigenti

Dirigente	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

- La decomposizione **corretta** è:

**Progetto, Sede → Dirigente**  
**Dirigente → Sede**

### 3) Ridefinizione dello schema

- Nell'esempio, introduciamo il concetto di Reparto per distinguere i dirigenti di una stessa sede (ogni dirigente opera in un reparto di una sede, e viceversa)

Dirigente	Progetto	Sede	Reparto
Rossi	Marte	Roma	1
Verdi	Giove	Milano	1
Verdi	Marte	Milano	1
Neri	Saturno	Milano	2
Neri	Venere	Milano	2

Dirigente → Sede, Reparto  
Sede, Reparto → Dirigente  
Progetto, Sede → Reparto

- È ora possibile operare una decomposizione in BCNF

Dirigente	Sede	Reparto
Rossi	Roma	1
Verdi	Milano	1
Neri	Milano	2

Progetto	Sede	Reparto
Marte	Roma	1
Giove	Milano	1
Marte	Milano	1
Saturno	Milano	2
Venere	Milano	2

Prog. Logica: normalizzazione

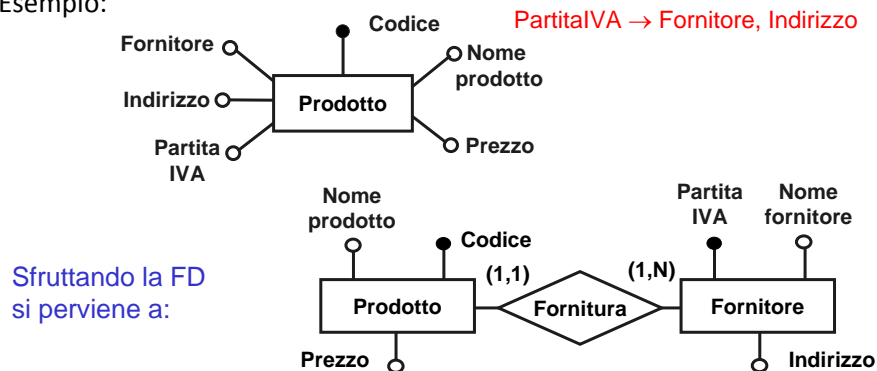
Sistemi Informativi T

23

### Progettazione e normalizzazione

- La teoria della normalizzazione può essere usata nella progettazione logica per **verificare e migliorare lo schema relazionale finale**
- Si può usare anche durante la progettazione concettuale per **verificare la qualità dello schema concettuale**

Esempio:



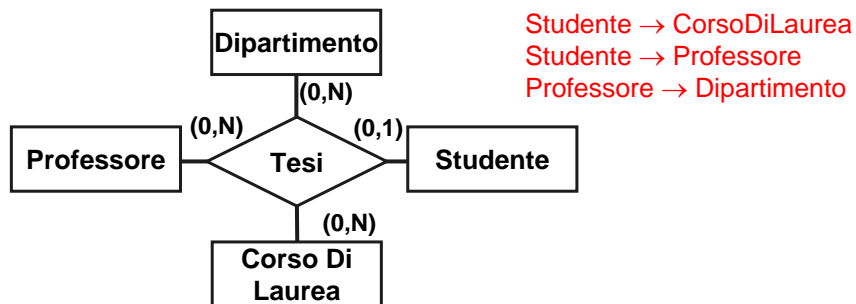
Prog. Logica: normalizzazione

Sistemi Informativi T

24

## Esempio: analisi di associazioni n-arie (1)

- Le associazioni n-arie spesso nascondono FD che possono dar luogo a schemi non normalizzati

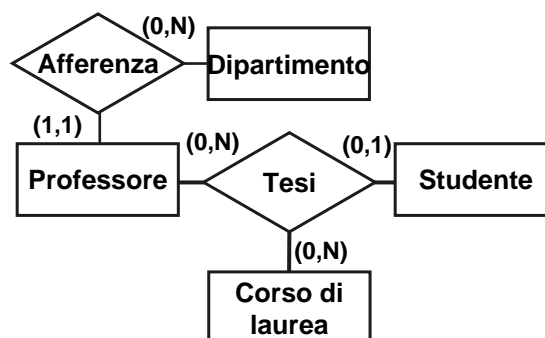


Tesi(Studente, Professore, Dipartimento, CorsoDiLaurea)

non è in 3NF a causa di **Professore  $\rightarrow$  Dipartimento**

## Esempio: analisi di associazioni n-arie (2)

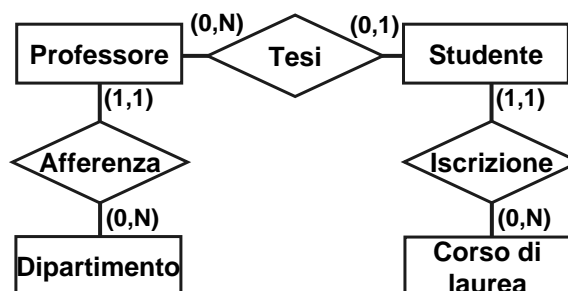
- Si ristruttura lo schema di conseguenza:



Tesi(Studente, Professore, CorsoDiLaurea) è ora in BCNF

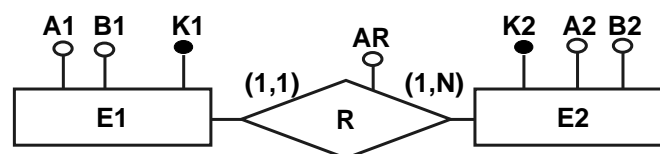
## Esempio: analisi di associazioni n-arie (3)

- L'associazione Tesi in realtà include 2 FD, tra loro indipendenti:  
 $\text{Studente} \rightarrow \text{CorsoDiLaurea}$  (iscrizione)  
 $\text{Studente} \rightarrow \text{Professore}$  (per chi ha un relatore)
- È quindi opportuno procedere a un'ulteriore ristrutturazione:



## FD e modello E/R

- È bene abituarsi a “leggere” uno schema E/R anche in termini di FD
- A tal fine si considerano le **cardinalità massime delle associazioni**:



$K1 \rightarrow A1, B1$

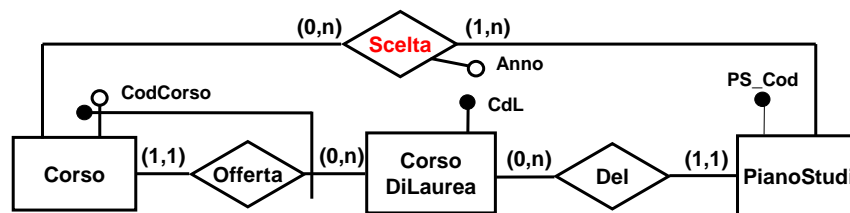
$K2 \rightarrow A2, B2$

$K1 \rightarrow K2, AR$  poiché  $\max\text{-card}(E1, R) = 1$

- Si suggerisce di rivedere le regole per la traduzione delle associazioni in termini di FD tra gli identificatori delle entità e di normalizzazione degli schemi...

## Possiamo fare a meno delle FD?

- Una buona progettazione concettuale rende spesso superfluo ragionare in termini di FD; tuttavia vi sono schemi E/R che, tradotti secondo le regole viste, possono dar luogo a schemi relazionali non normalizzati



- Vincolo: **si possono scegliere solo corsi offerti dal proprio CdL**
- L'associazione **Scelta** genera lo schema: **Scelta(CdL, CodCorso, PS\_Cod, Anno)**
- Quella individuata è solo una superchiave: **Scelta(CdL, CodCorso, PS\_Cod, Anno)** in quanto vale la FD **PS\_Cod → CdL**, e quindi **lo schema non è in 3NF**
- La traduzione corretta è quindi: **Scelta(CodCorso, PS\_Cod, Anno)**

Prog. Logica: normalizzazione

Sistemi Informativi T

29

## Algoritmo di normalizzazione in 3NF

## La terza forma normale

- Una forma normale meno restrittiva della BCNF si definisce come segue:

### Terza Forma Normale (3NF)

Uno schema  $R(X)$  è in terza forma normale se, per ogni dipendenza funzionale (non banale)  $Y \rightarrow Z$  definita su di esso,  $Y$  è una superchiave di  $R(X)$  oppure ogni attributo in  $Z$  è contenuto in almeno una chiave di  $R(X)$

- Una relazione in 3NF può ancora presentare anomalie
- Tuttavia il vantaggio è che **è sempre possibile ottenere schemi in 3NF preservando tutte le dipendenze**
  - Storicamente, esiste anche una “seconda forma normale”, che però non ha interesse pratico
- Vediamo ora come sia possibile ottenere automaticamente schemi in 3NF preservando tutte le dipendenze

## Algoritmo di normalizzazione in 3NF

- L'idea alla base dell'algoritmo che produce una decomposizione in 3NF è **creare una relazione per ogni gruppo di FD che hanno lo stesso lato sinistro (determinante)** e inserire nello schema corrispondente gli attributi coinvolti in almeno una FD del gruppo
- Per far questo è tuttavia **necessario minimizzare** l'insieme di FD individuate, altrimenti il risultato corretto non è garantito  
**Esempio:** Se le FD individuate sullo schema  $R(\underline{A}BCDEFG)$  sono:  
 **$AB \rightarrow CDEF, C \rightarrow F, F \rightarrow G$**   
si genererebbero gli schemi  **$R1(\underline{A}BCDEF), R2(\underline{C}F), R3(\underline{F}G)$**   
**Ma  $R1$  non è in 3NF a causa della FD  $C \rightarrow F$ !**
- Vediamo quindi prima come si può ragionare più precisamente con le FD e come si possono minimizzare



## Innanzitutto: chiusura di X

- Come prima cosa chiediamoci:  
*Se F è un insieme di FD su R(U) e X un insieme di attributi, quali attributi di U dipendono funzionalmente da X?*
  - Ad es., se F include  $A \rightarrow B$  e  $B \rightarrow C$ , allora è anche vero che  $A \rightarrow C$ . Infatti C dipende da B, che a sua volta dipende da A
- Denotiamo con  $X^+$  l'insieme degli attributi di R(U) che dipendono da X
- Calcolare  $X^+$  è semplice:

```
X+ = X;  
Ripeti:  
    Fine = true;  
    Per tutte le FD in F = {Vi → Wi, i=1,...,n}:  
        Se Vi ⊆ X+ e Wi ∉ X+ allora: {X+ = X+ ∪ Wi; Fine = false}  
Fino a che Fine = true o X+ = U
```

## Chiusura di X - esempio

- Supponiamo di avere  $F = \{A \rightarrow B, BC \rightarrow D, B \rightarrow E, E \rightarrow C\}$  e calcoliamo  $A^+$ , ovvero l'insieme di attributi che dipendono da A

```
A+ = A  
A+ = AB          usando A → B  
A+ = ABE         usando B → E  
A+ = ABEC        usando E → C  
A+ = ABECD       usando BC → D
```

- Quindi da A dipendono tutti gli attributi dello schema, ovvero  
**A è superchiave (e anche chiave)**

## Passo 1: FD "semplici"

- Per minimizzare un insieme di FD è innanzitutto necessario scriverle tutte in una forma "standard", in cui **sulla destra c'è sempre un singolo attributo**
- Supponiamo di avere  $F = \{AB \rightarrow CD, AC \rightarrow DE\}$
- Allora riscriviamo F come
$$F = \{AB \rightarrow C, AB \rightarrow D, AC \rightarrow D, AC \rightarrow E\}$$

## Passo 2: attributi "estranei"

- In alcune FD è possibile che sul lato sinistro ci siano degli attributi inutili ("estranei"): **vanno eliminati!**
- Supponiamo di avere  $F = \{AB \rightarrow C, A \rightarrow B\}$  e calcoliamo  $A^+$ 
$$A^+ = A$$
$$A^+ = AB \quad \text{poiché } A \rightarrow B \text{ e } A \subseteq A^+$$
$$A^+ = ABC \quad \text{poiché } AB \rightarrow C \text{ e } AB \subseteq A^+$$
- Quindi **C dipende solo da A**, ovvero **in  $AB \rightarrow C$  l'attributo B è estraneo** (perché a sua volta dipendente da A) e possiamo riscrivere l'insieme di FD più semplicemente come:  $F' = \{A \rightarrow C, A \rightarrow B\}$

*Come facciamo a stabilire che in una FD del tipo  $AX \rightarrow B$  l'attributo A è estraneo?*

- **Calcoliamo  $X^+$  e verifichiamo se include B, ovvero se basta X a determinare B!**

## Passo 3: FD ridondanti

- **Dopo** avere eliminato gli attributi estranei si deve verificare se vi sono intere FD inutili ("ridondanti"), ovvero FD che sono implicate da altre
- Supponiamo di avere  $F = \{B \rightarrow C, B \rightarrow A, C \rightarrow A\}$
- Si vede che  $B \rightarrow A$  è **ridondante** in quanto bastano le altre due per stabilire che A dipende da B, e quindi possiamo riscrivere l'insieme di FD più semplicemente come:  $F' = \{B \rightarrow C, C \rightarrow A\}$   
*Come facciamo a stabilire che una FD del tipo  $X \rightarrow A$  è ridondante?*
- La eliminiamo da F, calcoliamo  $X^+$  e verifichiamo se include A, ovvero se con le FD che restano riusciamo ancora a dimostrare che X determina A!

**NB: NON INVERTIRE I PASSI 2 E 3!**

- Sia  $F = \{AB \rightarrow C, B \rightarrow A, C \rightarrow A\}$ . Con il passo 2 scopriamo che A è estraneo in  $AB \rightarrow C$ , quindi otteniamo  $F' = \{B \rightarrow C, B \rightarrow A, C \rightarrow A\}$  e dopo possiamo eliminare  $B \rightarrow A$ , restando con  $F'' = \{B \rightarrow C, C \rightarrow A\}$
- Se invertiamo i passi non riusciamo più a eliminare  $B \rightarrow A$ !

Prog. Logica: normalizzazione

Sistemi Informativi T

37

## Creazione degli schemi in 3NF

- Avendo minimizzato l'insieme iniziale di FD si può procedere con la creazione degli schemi in 3NF

Passo 4:

Si raggruppano tutte le FD che hanno lo stesso **lato sinistro (determinante)** X e si crea uno schema che ha X come chiave

Passo 5:

Se 2 o più determinanti si determinano reciprocamente, si fondono gli schemi (più chiavi alternate per lo stesso schema)

Passo 6:

Alla fine si verifica che esista uno schema la cui chiave è anche chiave dello schema originario. Se non esiste lo si crea, usando gli attributi che compaiono in tutti i determinanti ed eliminando quelli determinati da altri

Prog. Logica: normalizzazione

Sistemi Informativi T

38

## Esempio 1

Sia  $F = \{A \rightarrow BC, B \rightarrow C, ABE \rightarrow D\}$  e  $R(ABCDE)$

Passo 1:  $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow C, ABE \rightarrow D\}$

Passo 2:  $F' = \{A \rightarrow B, A \rightarrow C, B \rightarrow C, AE \rightarrow D\}$  poiché  $B$  dipende da  $A$

Passo 3:  $F'' = \{A \rightarrow B, B \rightarrow C, AE \rightarrow D\}$  poiché  $A \rightarrow C$  è ridondante

Passo 4: Si generano gli schemi  $R1(\underline{AB})$ ,  $R2(\underline{BC})$  e  $R3(\underline{AED})$

Passi 5 e 6: Le chiusure delle chiavi sono:

$A^+ = ABC$

$B^+ = BC$

$AE^+ = ABCED$  che è quindi anche chiave dello schema non decomposto

## Esempio 2

Sia  $F = \{A \rightarrow BC, B \rightarrow AG, BE \rightarrow D\}$  e  $R(ABCDEFG)$

Passo 1:  $F = \{A \rightarrow B, A \rightarrow C, B \rightarrow A, B \rightarrow G, BE \rightarrow D\}$

Passo 2:  $F' = F$

Passo 3:  $F'' = F$

Passo 4: Si generano gli schemi  $R1(\underline{ABC})$ ,  $R2(\underline{BAG})$  e  $R3(\underline{BED})$

Passi 5 e 6: Le chiusure delle chiavi sono:

$A^+ = ABCG$

$B^+ = BAGC$

$BE^+ = ABCEDG$  che è quindi anche chiave dello schema non decomposto

Tuttavia  $A \rightarrow B$  e  $B \rightarrow A$ , quindi si fondono  $R1$  e  $R2$  in  $R12(\underline{ABCG})$ ,  
con, ad es.,  $A$  chiave primaria e  $B$  chiave alternata

### Esempio 3

Sia  $F = \{A \rightarrow B, B \rightarrow C, D \rightarrow E\}$  e  $R(ABCDE)$

Passo 1:  $F = \{A \rightarrow B, B \rightarrow C, D \rightarrow E\}$

Passo 2:  $F' = F$

Passo 3:  $F'' = F$

Passo 4: Si generano gli schemi  $R1(\underline{AB})$ ,  $R2(\underline{BC})$  e  $R3(\underline{DE})$

Passo 5: Non si fonde nulla

Passo 6: Nessun determinante è chiave dello schema non decomposto; si crea quindi lo schema

$R0(ABD)$

Poiché  $A \rightarrow B$ , si elimina B restando con  $R0(\underline{AD})$

### Esempio 4

$R(ABCDE)$ ,  $F = \{C \rightarrow AB, BC \rightarrow DE, D \rightarrow B\}$

1:  $F = \{C \rightarrow A, C \rightarrow B, BC \rightarrow D, BC \rightarrow E, D \rightarrow B\}$

2:  $F' = \{C \rightarrow A, C \rightarrow B, C \rightarrow D, C \rightarrow E, D \rightarrow B\}$  poiché B dipende da C

3:  $F'' = \{C \rightarrow A, C \rightarrow D, C \rightarrow E, D \rightarrow B\}$  poiché  $C \rightarrow B$  è ridondante

4:  $R1(\underline{CADE})$ ,  $R2(\underline{DB})$

5: nessuna fusione

6:  $C^+ = ABCDE$

$D^+ = BD$

C è quindi anche chiave dello schema non decomposto

## Esempio 5

$R(ABCDEFGH), F = \{ABC \rightarrow DEG, BD \rightarrow ACE, C \rightarrow BH, H \rightarrow BDE\}$

- 1:  $F = \{ABC \rightarrow D, ABC \rightarrow E, ABC \rightarrow G, BD \rightarrow A, BD \rightarrow C, BD \rightarrow E, C \rightarrow B, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E\}$
- 2: Conviene innanzitutto calcolare le chiusure dei lati sinistri delle FD
 

$ABC^+ = ABCDEGH$	$BD^+ = BDACEHG$
$C^+ = CBHDEAG$	$H^+ = HBDEACG$

 Quindi:
 
$$F' = \{C \rightarrow D, C \rightarrow E, C \rightarrow G, BD \rightarrow A, BD \rightarrow C, BD \rightarrow E, C \rightarrow B, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E\}$$
- 3:  $F'' = \{C \rightarrow G, BD \rightarrow A, BD \rightarrow C, C \rightarrow H, H \rightarrow B, H \rightarrow D, H \rightarrow E\}$
- 4,5,6: A questo punto si può osservare che C, BD e H sono tutte chiavi di R, quindi lo schema è in 3NF (e anche in BCNF!) e non va decomposto

Prog. Logica: normalizzazione

Sistemi Informativi T

43

## Esempio 6

$R(ABCDEFGH), F = \{AB \rightarrow CDE, CE \rightarrow AB, A \rightarrow G, G \rightarrow BD\}$

- 1:  $F = \{AB \rightarrow C, AB \rightarrow D, AB \rightarrow E, CE \rightarrow A, CE \rightarrow B, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$
- 2: Chiusure:
 

$AB^+ = ABCDEG$	$CE^+ = CEABGD$
$A^+ = AGBDCE$	$G^+ = GBD$

 Quindi:
 
$$F' = \{A \rightarrow C, A \rightarrow D, A \rightarrow E, CE \rightarrow A, CE \rightarrow B, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$$
- 3:  $F'' = \{A \rightarrow C, A \rightarrow E, CE \rightarrow A, A \rightarrow G, G \rightarrow B, G \rightarrow D\}$
- 4:  $R1(\underline{A}CEG), R2(\underline{C}EA), R3(\underline{G}BD)$
- 5: Si fondono R1 e R2:  $R12(\underline{A}CEG)$ , con CE chiave alternata
- 6: Non esiste nessuno schema che include una chiave di R, in quanto in R c'è anche H!. Si crea quindi lo schema  $R0(\underline{A}H): AH^+ = ABCDEGH$

Prog. Logica: normalizzazione

Sistemi Informativi T

44

## Esempio 7

Sia  $F = \{Imp \rightarrow Stip; Prog \rightarrow Bilancio\}$

Passo 1:  $F = \{Imp \rightarrow Stip; Prog \rightarrow Bilancio\}$

Passo 2:  $F' = F$

Passo 3:  $F'' = F$

Passo 4: Si generano gli schemi  $IMP(\underline{Imp}, Stip)$  e  $PROG(\underline{Prog}, Bilancio)$

Passo 5: Non si fonde nulla

Passo 6: Né Imp né Prog sono chiavi dello schema non decomposto;  
si crea quindi lo schema

$PARTECIPAZIONI(\underline{Imp}, \underline{Prog})$

## Normalizzare o no?

- La normalizzazione non va intesa come un obbligo, in quanto in alcune situazioni le anomalie che si riscontrano in schemi non normalizzati sono un male minore rispetto alla situazione che si viene a creare normalizzando
- In particolare, le cose da considerare sono:
  - Normalizzare elimina le anomalie, ma può appesantire l'esecuzione di certe operazioni (join tra gli schemi normalizzati)
  - La frequenza con cui i dati vengono modificati incide su qual è la scelta più opportuna (relazioni "quasi statiche" danno meno problemi se non normalizzate)
  - La ridondanza presente in relazioni non normalizzate va quantificata, per capire quanto incida sull'occupazione di memoria e sui costi da pagare quando le repliche di una stessa informazione devono essere aggiornate

## Riassumiamo:

- Una forma normale è una proprietà di uno schema relazionale che ne garantisce la “**qualità**”, cioè l’**assenza di determinati difetti**
- Una **relazione non normalizzata** **presenta ridondanze** e dà luogo a **comportamenti poco desiderabili durante gli aggiornamenti**
- La definizione delle **forme normali** (3NF e BCNF) si basa sul vincolo di **dipendenza funzionale (FD)**
- **Normalizzare uno schema significa decomporlo in sottoschemi**
- **Ogni decomposizione deve essere senza perdita**, ovvero deve permettere di ricostruire esattamente la relazione originaria non decomposta
- È anche opportuno che la decomposizione **preservi le FD**, al fine di **evitare** (o ridurre la complessità di) **query di verifica** (o **trigger**) che garantiscano che i vincoli siano rispettati