Basi di Dati
La teoria della normalizzazione
-- seconda parte --

Proprietà delle decomposizioni

BCNF

Esempio

ESAMI(MATR, NomeS, IndirizzoS, CAPS, CodiceFiscaleS, DataNascitaS, Corso, Voto, Lode, DataEsame, CodProf, NomeProf, Qualifica, TipoUfficio)

Notazione abbreviata:

ESAMI(MATR, NS, IS, CAP, CF, DN, Co, Vo, Lo, DE, CP, NP, Q, TU)

Dipendenze funzionali: F=

{MATR → NS, IS, CF, DN;

CF → MATR;

IS → CAP;

MATR, Co → Vo, Lo, DE, CP;

CP → NP, Q;

Q → TU;

CP → Co}

Outline

- **Normalizzazione**
- Decomposizioni senza perdita
- Decomposizioni che conservano le dipendenze
- Le forme normali
- BCNF

Normalizzazione

- Utilizzeremo le dipendenze funzionali nella normalizzazione.
- La **normalizzazione** consiste nella *decomposizione* di uno schema di relazione in modo da ottenere più schemi che rispettino una forma normale e minimizzino le anomalie.
- Vedremo che esistono più forme normale diverse.

Normalizzazione

- Ora ci concentreremo su una generica decomposizione di uno schema di relazione (non necessariamente legata a una specifica normalizzazione) e mostreremo due proprietà che una decomposizione in forma normale deve garantire:
  - **Decomposizione senza perdita di informazione**
  - **Conservazione delle dipendenze funzionali**

Outline

- Introduzione
- **Decomposizioni senza perdita**
- Decomposizioni che conservano le dipendenze
- Le forme normali
- BCNF

Decomposizione senza perdita: Esempio

Consideriamo una versione semplificata della relazione ESAMI chiamata *S*

S

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	03	Anselma

Immaginiamo di avere, invece di

S(Matr, NomeS, Voto, Corso, CodC, Titolare),

la sua decomposizione in

S_1(Matr, NomeS, Voto, Corso)

S_2(Corso, CodC, Titolare)

Esempio

- Relazione S_1

MATR	NomeS	Voto	Corso
31	Piero	27	Basi Dati
31	Piero	26	Algoritmi
37	Giovanni	25	Basi Dati

- Relazione S_2

Corso	CodC	Titolare
Basi Dati	01	Pensa
Algoritmi	02	Damiani
Basi Dati	03	Anselma

(per ottenere le relazioni S_1 e S_2 è sufficiente proiettare S sugli schemi di S_1 e di S_2)

Problema…

Le informazioni che rappresento usando le relazioni decomposte sono le stesse che rappresento nella base di dati originale?

…Problema…

Per esempio la prima tupla di S ci dice che lo studente Piero ha sostenuto l'esame di Basi di Dati con il prof. Pensa.

Partendo da S_1 posso ottenere la stessa informazione coinvolgendo anche S_2 e usando Corso come "ponte" per aggiungere le informazioni mancanti.

Posso quindi "ricomporre" le informazioni utilizzando un **natural join**.

Problema

Calcolo quindi $S_1 \bowtie S_2$

S_1

MATR	NomeS	Voto	Corso
31	Piero	27	Basi Dati
31	Piero	26	Algoritmi
37	Giovanni	25	Basi Dati

$\bowtie$

S_2

Corso	CodC	Titolare
Basi Dati	01	Pensa
Algoritmi	02	Damiani
Basi Dati	03	Anselma

$S_1 \bowtie S_2$

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	27	Basi Dati	03	Anselma
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	01	Pensa
37	Giovanni	25	Basi Dati	03	Anselma

Problema

Confrontiamo le due relazioni

S

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	03	Anselma

$S_1 \bowtie S_2$

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	27	Basi Dati	03	Anselma
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	01	Pensa
37	Giovanni	25	Basi Dati	03	Anselma

Tuple spurie

- Se ho una base di dati composta da *S* e la sostituisco con le due relazioni S_1 e S_2, il natural join **perde** informazioni perché introduce le cosiddette **tuple spurie**.

- Una volta che abbiamo sostituito a S le relazioni S_1 e S_2, le tuple spurie sono indistinguibili da quelle originarie e non so più decidere se Piero ha sostenuto l'esame con Anselma o Pensa: ho perso questa informazione.

- (Si noti che si dice **perdita di informazione** pur avendo tuple in più)

Tuple spurie

Considero una relazione r(A) e una sua decomposizione in

$$r_1(A_1) = \pi_{A_1}(r(A))$$

$$r_2(A_2) = \pi_{A_2}(r(A))$$

con $A_1 \subseteq A$, $A_2 \subseteq A$ e $A_1 \cup A_2 = A$

La differenza insiemistica

$$(r_1(A_1) \bowtie r_2(A_2)) \quad - \quad r(A)$$

consiste nell'insieme delle tuple spurie che ottengo dalla decomposizione. Se la differenza è vuota, ottengo la definizione di join senza perdita di informazione…

Decomposizione con join senza perdita di informazioni (definizione)

Data uno schema di relazione *R(A)*, dati due sottoinsiemi di attributi $A_1 \subseteq A$ e $A_2 \subseteq A$, con $A_1 \cup A_2 = A$, $\{R_1(A_1), R_2(A_2)\}$ è una **decomposizione senza perdita di informazione** se e solo se **per ogni istanza** *r(A)* di *R(A)* vale

$$r(A) = r_1(A_1) \bowtie r_2(A_2)$$

dove

- $r_1(A_1) = \pi_{A_1}(r(A))$
- $r_2(A_2) = \pi_{A_2}(r(A))$

Esempio

Immaginiamo che il codice del corso identifichi il "concetto" di corso

S

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	03	Anselma

Consideriamo quindi un nuovo schema per S'_1 con CodC al posto di Corso:

MATR	NomeS	Voto	CodC
31	Piero	27	01
31	Piero	26	02
37	Giovanni	25	03

Esempio

Calcolo quindi nuovamente $S'_1 \bowtie S_2$

MATR	NomeS	Voto	CodC
31	Piero	27	01
31	Piero	26	02
37	Giovanni	25	03

$\bowtie$

Corso	CodC	Titolare
Basi Dati	01	Pensa
Algoritmi	02	Damiani
Basi Dati	03	Anselma

MATR	NomeS	Voto	Corso	CodC	Titolare
31	Piero	27	Basi Dati	01	Pensa
31	Piero	26	Algoritmi	02	Damiani
37	Giovanni	25	Basi Dati	03	Anselma

Questa volta la relazione originale è ricostruita senza perdita di informazione, cioè senza tuple spurie.

Criterio di decomposizione senza perdita: Teorema

Sia R(A) uno schema con dipendenze funzionali F decomposto in $\{R_1(A_1), R_2(A_2)\}$ dove $A_1 \cup A_2 = A$.

La decomposizione di **$R(A)$ in $\{R_1(A_1), R_2(A_2)\}$** è **senza perdita di informazione** per ogni istanza che soddisfa le dipendenze funzionali F se e solo se

$$(A_1 \cap A_2 \text{ è superchiave di } A_1) \vee (A_1 \cap A_2 \text{ è superchiave di } A_2)$$

Intuitivamente: $A_1 \cap A_2$ sono gli attributi usati dal natural join per ricomporre R e, se sono una superchiave, individuano una sola tupla di R_1 (o di R_2) senza generare tuple spurie.

Criterio di decomposizione senza perdita: Teorema (riformulazione)

Considerando la nuova definizione di superchiave
(cioè K è superchiave di R(A) che ha dipendenze funzionali F se e solo se $K \rightarrow A \in F^+$), possiamo riformulare così:

Sia R(A) uno schema con dipendenze funzionali F decomposto in $\{R_1(A_1), R_2(A_2)\}$ dove $A_1 \cup A_2 = A$.

La decomposizione di **$R(A)$ in $\{R_1(A_1), R_2(A_2)\}$** è **senza perdita di informazione** per ogni istanza che soddisfa le dipendenze funzionali F se e solo se

$$A_1 \subseteq (A_1 \cap A_2)_F^+ \quad \vee \quad A_2 \subseteq (A_1 \cap A_2)_F^+$$

Esempio

Consideriamo *S(Matr, NomeS, Voto, Corso, CodC, Titolare)*
e la sua decomposizione in
S'_1(Matr, NomeS, Voto, CodC) e *S_2(Corso, CodC, Titolare)*
e intersechiamo gli attributi di *S'_1* e *S_2*:

{Matr,NomeS,Voto,CodC} ⋂ {CodC,Corso,Titolare} = {CodC}

CodC è superchiave di S_2 perché CodC→Corso,Titolare ∈ F

Quindi la decomposizione è senza perdita di informazione.

Criterio di decomposizione senza perdita: Dimostrazione

Dimostreremo solo la condizione di sufficienza.

Ipotesi: $A_1 \subseteq (A_1 \cap A_2)_F^+ \ \vee \ A_2 \subseteq (A_1 \cap A_2)_F^+$, cioè

$(A_1 \cap A_2) \rightarrow A_1 \in F^+ \ \vee \ (A_1 \cap A_2) \rightarrow A_2 \in F^+$

Tesi: $r(A) = r_1(A_1) \bowtie r_2(A_2)$ ovvero

1. $r(A) \subseteq r_1(A_1) \bowtie r_2(A_2) \quad \wedge$
2. $r_1(A_1) \bowtie r_2(A_2) \subseteq r(A)$

Dimostriamo solo che $r_1(A_1) \bowtie r_2(A_2) \subseteq r(A)$

($r(A) \subseteq r_1(A_1) \bowtie r_2(A_2)$ vale anche sotto ipotesi meno restrittive)

Dimostrazione

Ipotesi: $(A_1 \cap A_2) \rightarrow A_1 \in F^+ \vee (A_1 \cap A_2) \rightarrow A_2 \in F^+$

Tesi: $r_1(A_1) \bowtie r_2(A_2) \subseteq r(A)$

Preso un elemento qualsiasi dell'insieme a sinistra $r_1(A_1) \bowtie r_2(A_2)$ dobbiamo dimostrare che è incluso nell'insieme di destra $r(A)$.

Seguiamo le definizioni degli operatori e introduciamo una rappresentazione grafica per facilitare la dimostrazione.

Dimostrazione

Costruiamo una generica tupla t appartenente al natural join $r_1(A_1) \bowtie r_2(A_2)$.

Lo schema del natural join, che per ipotesi è uguale allo schema di R(A), comprenderà tutti gli attributi di A_1 e di A_2 di cui alcuni (quelli in $A_1 \cap A_2$) in comune.

$r_1 \bowtie r_2$

	A1 – A2	A1 ∩ A2	A2 – A1
t	- - -	///	+ + +

Per dimostrare la tesi, dobbiamo verificare che la tupla *t* sia contenuta in *r(A).*

Dimostrazione

Per definizione di natural join, esisterà una tupla t_1 in $R_1(A_1)$ corrispondente alla tupla t.

$R_1(A_1)$ ha alcuni attributi nell'intersezione $A_1 \cap A_2$ e altri attributi fuori, in $A_1 - A_2$

r_1	A1 – A2	A1 ∩ A2
t_1	- - -	///

Per definizione di natural join, esisterà una tupla t_2 in $R_2(A_2)$ corrispondente alla tupla t.

$R_2(A_2)$ ha alcuni attributi nell'intersezione $A_1 \cap A_2$ e altri attributi fuori, in $A_2 - A_1$

r_2	A1 ∩ A2	A2 – A1
t_2	///	+ + +

Dimostrazione

Partizioniamo gli attributi di R(A) in tre gruppi: attributi solo di A_1, attributi solo di A_2, attributi in comune.

Dobbiamo ricordare che

- $r_1(A_1) = \pi_{A_1}(r(A))$
- $r_2(A_2) = \pi_{A_2}(r(A))$

Dato che r_1 è una proiezione di r, t_1 di $r_1(A_1)$ deriverà da una (o più) tuple di $r(A)$ fatte come t'_1.

r	A1 – A2	A1 ∩ A2	A2 – A1
t'_1	- - -	///	???

Dimostrazione

Partizioniamo gli attributi di R(A) in tre gruppi: attributi solo di A_1, attributi solo di A_2, attributi in comune.

Dobbiamo ricordare che

- $r_1(A_1) = \pi_{A_1}(r(A))$
- $r_2(A_2) = \pi_{A_2}(r(A))$

Dato che r_2 è una proiezione di r, t_2 di $r_2(A_2)$ deriverà da una (o più) tuple di $r(A)$ fatte come t'_2.

r	A1 – A2	A1 ∩ A2	A2 – A1
t'_1	- - -	///	???
t'_2	***	///	+ + +

Dimostrazione

Ricordiamo l'ipotesi: $(A_1 \cap A_2) \rightarrow A_1 \in F^+ \vee (A_1 \cap A_2) \rightarrow A_2 \in F^+$

Consideriamo $(A_1 \cap A_2) \rightarrow A_2 \in F^+$ (l'altro caso è analogo).

La relazione r(A) per ipotesi rispetta i vincoli, quindi il vincolo $A_1 \cap A_2 \rightarrow A_2$ deve essere verificato in r(A)

r	A1 – A2	A1 ∩ A2	A2 – A1
t'_1	- - -	///	???
t'_2	***	///	+ + +

Questo vuol dire che, per definizione di dipendenza funzionale, se $t'_1[A_1 \cap A_2] = t'_2[A_1 \cap A_2]$, allora $t'_1[A_2] = t'_2[A_2]$. Di conseguenza ??? = + + +

r	A1 – A2	A1 ∩ A2	A2 – A1
t'_1	- - -	///	+ + +
t'_2	***	///	+ + +

Dimostrazione

Confrontiamo $r(A)$ e $r_1(A_1) \bowtie r_2(A_2)$:

r	A1 – A2	A1 ∩ A2	A2 – A1
t'_1	- - -	///	+ + +
t'_2	***	///	+ + +

$r_1 \bowtie r_2$	A1 – A2	A1 ∩ A2	A2 – A1
t	- - -	///	+ + +

Ho verificato che una qualunque tupla t di $r_1(A_1) \bowtie r_2(A_2)$ è contenuta in $r(A)$.

[CVD]

Outline

- Introduzione
- Decomposizioni senza perdita
- **Decomposizioni che conservano le dipendenze**
- Le forme normali
- BCNF

Restrizione

- Consideriamo una relazione R(A) abbinata alle sue dipendenze funzionali F.
- Decomponiamo $R(A)$ in $R_1(A_1)$ e $R_2(A_2)$.
- Riscriviamo F in modo che ogni relazione abbia le proprie dipendenze funzionali che discendano dalla relazione di partenza
  $(R_1(A_1), F_1)$ e $(R_2(A_2), F_2)$.
- F_1 è la *restrizione* di F in R_1
- F_2 è la *restrizione* di F in R_2

Restrizione (definizione)

Dato uno schema *R(A*) su cui valgono le dipendenze funzionali *F* e una decomposizione $\boldsymbol{R_i(A_i)}$ di *R(A*),
la **restrizione** F_i di F su R_i è definita come

$$F_i = \{ X \rightarrow Y \mid (X \rightarrow Y \in F^+) \wedge (X,Y \subseteq A_i)\}$$

Nella definizione usiamo F^+ e non semplicemente *F*.
Perché non possiamo semplicemente prendere le d.f. in F e eliminare quelle con attributi non in R_i?

(Contro)Esempio

- *Supponiamo di non considerare F^+ ma solo F…*
- Consideriamo una relazione *R(A,B,C)* con l'insieme di d.f. *F = {A → B, B → C}*
- Consideriamo una decomposizione: $R_1(A,C)$ e $R_2(B,C)$
- La restrizione di F a R_2(B,C) sarebbe $F_2 = \{B \rightarrow C\}$.
- Ma la restrizione di F a R_1(A,C) sarebbe vuota.
- È un **errore**: nella relazione originale R(A,B,C), per via delle d.f. *A → B* e *B → C*, non sono ammesse due tuple diverse t_1 e t_2 con $t_1[A]=t_2[A]$ e $t_1[C]\neq t_2[C]$. Non devono essere ammesse neanche nella decomposizione in R_1 e R_2.
- Se, invece, nel calcolo delle restrizioni partiamo non da F, ma dalla chiusura $F^+=\{A \rightarrow B, B \rightarrow C, \mathbf{A \rightarrow C}\}$, ricaviamo correttamente che $F'_1 = \{A \rightarrow C\}$.

Decomposizione che conserva le dipendenze (definizione)

Data una relazione $R(A)$ con dipendenze funzionali F, decomposta in $R_1(A_1)$ con la restrizione F_1 e $R_2(A_2)$ con la restrizione F_2, la decomposizione $\{R_1,R_2\}$ **conserva le dipendenze** quando $F_1 \cup F_2 \vdash F$.

Decomposizione che conserva le dipendenze

- Per conservare le dipendenze è sufficiente calcolare correttamente le restrizioni oppure in alcune decomposizioni non è possibile conservarle?

- La risposta è: alcune decomposizioni non conservano le dipendenze.
  Ad esempio…

(Contro)Esempio

- Consideriamo una relazione *R(A,B,C)* con l'insieme di d.f.
  $F = \{A \rightarrow C,\ B \rightarrow C\}$ e decomponiamola in
  $R_1(A,B)$ e $R_2(A,C)$
- La decomposizione è senza perdita perché R_1 e R_2 hanno in comune l'attributo A, che è chiave di R_2.
- Calcoliamo le restrizioni:
  $F_1=\emptyset$ e $F_2=\{A \rightarrow C\}$
- Non è possibile portare $B \rightarrow C$ né in F_1 né in F_2 perché mancherebbe o l'attributo C o l'attributo B.
- La d.f. $B \rightarrow C$ non è deducibile da $F_1 \cup F_2$ e quindi la decomposizione non conserva le dipendenze.
- Quindi, se la d.f. $B \rightarrow C$ andasse persa, nel database potremmo ammettere tuple $t_1, t_2 \in R_1 \bowtie R_2$ con $t_1[B]=t_2[B]$ ma $t_1[C] \neq t_2[C]$.

Decomposizioni che conservano le dipendenze

- Le dipendenze perse nella decomposizione andrebbero ri-aggiunte come *vincoli globali* (vincoli, cioè, che coinvolgono più relazioni).
- Se non lo facessimo, la base di dati ammetterebbe istanze inconsistenti rispetto ai vincoli originali.
- Ma la verifica dei vincoli globali è poco praticabile perché è costosa.
- Infatti a ogni modifica di una tupla di una delle relazioni coinvolte, bisognerebbe eseguire un join per riunire tutti gli attributi coinvolti dal vincolo.

Outline

- Introduzione
- Decomposizioni senza perdita
- Decomposizioni che conservano le dipendenze
- **Le forme normali**
- BCNF

Le forme normali

- Le **forme normali** sono delle "ricette" di buona progettazione di basi di dati volte alla minimizzazione delle ridondanze e delle anomalie.
- A una forma normale può essere associato un algoritmo di **normalizzazione**, che specifica come decomporre uno schema relazionale per renderlo in forma normale.
- Esistono diverse forme normali, che corrispondono a compromessi diversi tra proprietà delle decomposizioni, compattezza della rappresentazione e minimizzazione delle anomalie.

Le forme normali

Principali forme normali

Normal form	Defined by	Brief definition
First normal form (1NF)	Two versions: E.F. Codd (1970), C.J. Date (2003)[9]	Table faithfully represents a relation and has no *repeating groups*
Second normal form (2NF)	E.F. Codd (1971)[2]	No non-prime attribute in the table is functionally dependent on a proper subset of any candidate key
Third normal form (3NF)	E.F. Codd (1971);[2] see also Carlo Zaniolo's equivalent but differently expressed definition (1982)[10]	Every non-prime attribute is non-transitively dependent on every candidate key in the table. The attributes that do not contribute to the description of the primary key are removed from the table. In other words, no transitivity dependency is allowed.
Elementary Key Normal Form (EKNF)	C.Zaniolo (1982)[10]	Every non-trivial functional dependency in the table is either the dependency of an elementary key attribute or a dependency on a superkey
Boyce–Codd normal form (BCNF)	Raymond F. Boyce and E.F. Codd (1974)[11]	Every non-trivial functional dependency in the table is a dependency on a superkey
Fourth normal form (4NF)	Ronald Fagin (1977)[12]	Every non-trivial multivalued dependency in the table is a dependency on a superkey
Fifth normal form (5NF)	Ronald Fagin (1979)[13]	Every non-trivial join dependency in the table is implied by the superkeys of the table
Domain/key normal form (DKNF)	Ronald Fagin (1981)[14]	Every constraint on the table is a logical consequence of the table's domain constraints and key constraints
Sixth normal form (6NF)	C.J. Date, Hugh Darwen, and Nikos Lorentzos (2002)[15]	Table features no non-trivial join dependencies at all (with reference to generalized join operator)

(da George Kollios, Introduction to Database Systems, http://www.cs.bu.edu/faculty/gkollios/cs460f17/)

Le forme normali

- La 1NF è già stata spiegata quando abbiamo introdotto il modello relazionale: le relazioni hanno valori con domini semplici (non composti o con ripetizioni)
- La 2NF, 3NF e BCNF sono forme normali basate sulle dipendenze funzionali (la 2NF ha solo valore storico)
- La 4NF è stata introdotta su una classe di vincoli di cui non abbiamo parlato, i vincoli di dipendenza funzionale multi-valore
- La 5NF si basa su vincoli di decomposizione con join senza perdita
- La DKNF si basa su un ragionamento su chiavi e domini

Le forme normali

Tratteremo solo le due forme normali più usate, entrambe basate sulle dipendenze funzionali:

- Boyce-Codd Normal Form (BCNF)
- Terza forma normale (3NF)

Outline

- Introduzione
- Decomposizioni senza perdita
- Decomposizioni che conservano le dipendenze
- Le forme normali
- **BCNF**

Boyce-Codd Normal Form (BCNF)

Data una relazione *R(A)* in 1NF e un insieme di dipendenze funzionali *F*, la relazione è in **BCNF** se e solo se
per ogni $X \rightarrow Y \in F$ si verifica **almeno una** delle seguenti condizioni:

1. $Y \subseteq X$ (cioè $X \rightarrow Y$ è una dipendenza **riflessiva**)
2. *X* è **superchiave** di *R*

Esempio

La relazione ESAMI con le d.f. F

{ MATR → NS, IS, CAP, CF, DN;

CF → MATR;

IS → CAP;

MATR, Co → Vo, Lo, DE, CP, NP;

CP → NP, Q;

Q → TU;

CP → Co }

non è in BCNF: le chiavi sono {MATR,Co}, {CF,Co}, {MATR,CP}, {CF,CP} e ad es. la prima d.f. non è riflessiva e MATR non è una superchiave.

Esempio

Consideriamo invece un sottoinsieme dello schema:

STUDENTE(MATR,NS,CF,DN) con le d.f.

F = {MATR → CF,NS,DN; CF → MATR}

Analizzando la relazione troviamo due chiavi candidate: MATR oppure CF.

Le dipendenze funzionali sono tutte del tipo 2 perché sia MATR che CF sono superchiavi, quindi STUDENTE è una relazione in BCNF.

Intuizione

- Una relazione deve contenere un unico concetto ancorché identificato con chiavi diverse.

- Esempio: la relazione STUDENTE coglie l'unico concetto di studente, ancorché identificato da MATR o da CF.

- È per questo motivo che, escludendo le d.f. banali (riflessive), la BCNF ammette solo d.f. che dipendono da superchiavi.

BCNF e ridondanze

Perché la BCNF evita le ridondanze indotte dalle d.f.?

Informalmente:

- Una ridondanza è una ripetizione che posso evitare.
- Le ripetizioni sono evitabili se possono essere dedotte da altre informazioni.
- Ogni d.f. introduce, per definizione, una ripetizione. Per es. MATR→NS,DN dice che sono obbligato ad avere NS e DN uguali quando MATR è uguale.
- Ma se ogni d.f. non banale X→Y è BCNF, non ho ripetizioni perché per definizione X è una superchiave, quindi non posso avere due tuple uguali su X e non ho ripetizioni indotte dalle d.f.

BCNF e anomalie

Perché la BCNF evita le anomalie?

- *Anomalie di aggiornamento.*
- Causate dalle ridondanze.
- Ad esempio della relazione non in BCNF
  ESAMI(MATR, NS, IS, CAP, CF, DN, Co, Vo, Lo, DE, CP, NP, Q, TU)
  i dati dello studente o del professore sono ripetuti in ogni tupla dei loro esami e un aggiornamento dei dati può richiedere l'aggiornamento di molte tuple.
- In BCNF non ci sono ridondanze, quindi non presenta anomalie di aggiornamento.

BCNF e anomalie

- *Anomalie di inserimento/cancellazione.*
- Consideriamo la relazione non in BCNF E(MATR,NS,DN,Co,Vo,DE) con d.f. F={MATR→NS,DN; MATR,Co→Vo,DE}.
- Anomalia di inserimento: se inserisco un nuovo studente, cioè MATR, NS e DN, devo inserire tutta la chiave, cioè anche un suo esame (MATR e CORSO), ma non posso perché sarebbero nulli.
- Intuitivamente: se, come succede in BCNF, ogni relazione ha solo d.f. banali o con attributi che dipendono solo dalle superchiavi, ogni concetto viene rappresentato in una relazione a sé.
- Per le anomalie di cancellazione il discorso è analogo.

Normalizzazione in BCNF

Boyce e Codd hanno proposto anche un algoritmo di normalizzazione che decompone uno schema di una base di dati producendo uno schema in BCNF.

Idea dell'algoritmo:

1. *Per ogni dipendenza $X \rightarrow Y$ in R che vìola la BCNF*
2. *Decomponi R definendo una nuova relazione su $R_X(XY)$ (che ha chiave X ed è quindi BCNF)*
3. *Elimina Y dalla relazione originaria (e quindi anche la d.f. che non è BCNF)*

Non vedremo nel dettaglio questo algoritmo.

Esempio

Consideriamo ora lo schema

CC(Titolare,NConto,NAgenzia,CittaAgenzia,Saldo)

F = {NConto→NAgenzia,CittaAgenzia,Saldo;

Titolare,NConto →NAgenzia,CittaAgenzia,Saldo}

- *CC* non è in BCNF perché la prima d.f. non è riflessiva e *NConto* non è superchiave.
- Intuitivamente, le d.f. ci fanno capire che la relazione rappresenta due concetti: conto e titolare. Se spostiamo il concetto di conto in una nuova relazione…

Esempio

CC'(<u>NConto</u>,NAgenzia,CittaAgenzia,Saldo)

$F_{CC'}$ = *{NConto → NAgenzia,CittaAgenzia,Saldo}*

TITOLARI(<u>Titolare,NConto</u>)

$F_{TITOLARI} = \varnothing$

(non riportiamo le d.f. riflessive, dette anche **banali**)

Si può dire che il database è in BCNF?

Sì, l'unica dipendenza è del tipo 2

Complessità di calcolo

- La complessità dell'algoritmo è *esponenziale*
- Nel passo 2 devo calcolare le restrizioni $F_{i'}$ e $F_{i''}$
- Il calcolo delle restrizioni, dal punto di vista algoritmico, è *esponenziale*: richiede il calcolo di F^+ per estrarre tutte le dipendenze in $A_i - Y$ e XY

Limitazioni di BCNF (esempio)

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

Con due vincoli di dipendenza funzionale:

f_1: Progetto, Sede → Dirigente

f_2: Dirigente → Sede

La relazione non è in BCNF: in f_2 Dirigente non è superchiave

Però:

Progetto, Sede → Dirigente coinvolge tutti gli attributi e quindi nessuna decomposizione può conservare tale d.f.

Limitazioni di BCNF

- Quindi esistono schemi che vìolano la BCNF e per cui non esiste alcuna decomposizione in BCNF che conservi le dipendenze.
- In alcuni casi la strada della normalizzazione in BCNF non è praticabile.